Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

Untersuchung von Verwaltungsschalen als Werkzeuge zum Testen von Komponenten in wandelbaren Produktionsanlagen

Abschlussarbeit

zur Erlangung des akademischen Grades

Master of Engineering (M.Eng.)

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich I - Energie und Information
Studiengang Informations- und Kommunikationstechnik

1. Prüfer:	Prof. Dr.-Ing. S. Schäfer
2. Prüfer:	Dipl.-Ing. D. Schöttke

Eingereicht von:	Vasil Denkov
Matrikelnummer:	559026
Eingereicht am:	26.10.2022

Kurzfassung

Die folgende Abschlussarbeit umfasst die Thematik der softwaretechnischen Automatisierung des industriellen Testprozesses durch die Kontextualisierung traditioneller Verifikationsstrategien aus dem Bereich der Softwareentwicklung in der Domäne der Industrie 4.0 Vision. Im Rahmen der Untersuchung wurden mehrere die Thematik adressierende Literaturquellen in Betracht gezogen. Darauffolgend wurden unterschiedliche Möglichkeiten zur Integration eines Testdurchführungsprozesses auf Basis der Verwaltungsschale diskutiert. Als Ziel galt der Entwurf, die partielle Implementierung und vollständige Auswertung einer flexiblen und generischen Softwarearchitektur, welche als Testautomatisierungslösung dient. Die Architekturdienste werden durch das Metamodell der Verwaltungsschale und mittels der Programmierumgebung *Eclipse BaSyx* zur Verfügung gestellt. Die spezifizierte Softwarearchitektur und die Programmierschnittstellen der synthetisierten Softwarekomponenten ermöglichen die Definition und Durchführung von Verifikationsprozessen für einzelne Anlagenkomponenten, bei welchen die Verwaltungsschale als zentrale Auskunftsstelle gilt. Die einzelnen Funktionalitäten jeder Softwarekomponente wurden nachvollziehbar beschrieben und ausführlich getestet. Zum Zweck der Konzeptprüfung fand die Implementierung von zwei Test-Stufen statt, welche eine strukturelle Verifikation von Verwaltungsschalen und zustandsorientierte Validierung von Führungskomponenten über den digitalen Vertreter ermöglichen.

Abstract

The following thesis covers the topic of automating industrial test processes by contextualising traditional software verification strategies within the Industrie 4.0 domain. Throughout the study, multiple literary sources are considered and different designs for integrating a test execution process based on the concept of an asset administration shell are discussed. A flexible and generic software architecture is designed that serves as an automated solution for testing arbitrary characteristics of individual manufacturing components, whose single point of information and communication is the asset administration shell. The architecture is brought to life through a fully-tested and documented proof-of-concept implementation based on the Eclipse BaSyx programming environment. The implementation is scoped but not limited to handle structural and state machine verification of asset administration shells and their underlying assets, respectively.

Publikationen

Ausgewählte Ideen und Abbildungen wurden im Rahmen der Bearbeitung des Themas als Teil der folgenden Publikationen veröffentlicht:

[Sch22a] S. Schäfer, D. Schöttke, T. Kämpfe, V. Denkov und B. Tauber. „Verwaltungsschale: Einsatz in wandelbaren Produktionsanlagen“. In: *atp magazin 8/2022: Flexibilität und Effizienz in der Prozessindustrie*. Aug. 2022.

[Sch22b] S. Schäfer, D. Schöttke, T. Kämpfe, V. Denkov und B. Tauber. „Verwaltungsschalen zum anlagenweiten Testen wandelbarer Produktionsanlagen“. In: Jan. 2022, S. 59–74. ISBN: 9783181023990. DOI: 10.51202/9783181023990-59.

Inhaltsverzeichnis

Publikationen **iii**

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Zielsetzung . . . 2
1.3 Vorgehensweise und Aufbau der Arbeit . . . 3

2 Grundlagen und Stand der Technik **4**
2.1 Softwaretest . . . 4
2.1.1 Definitionen und Ziele . . . 4
2.1.2 Testaktivitäten . . . 5
2.1.3 Testprinzipien . . . 6
2.1.4 Teststufen . . . 7
2.1.5 Testtypen . . . 9
2.2 Testautomatisierung . . . 10
2.2.1 Definitionen und Ziele . . . 11
2.2.2 Generische Architektur . . . 11
2.2.3 Berichte und Metriken . . . 13
2.2.4 Verifizierung . . . 13
2.3 Domänenspezifische Testverfahren . . . 14
2.3.1 Virtuelle Inbetriebnahme . . . 14
2.3.2 Modellbasiertes Testen . . . 16
2.4 Industrie 4.0 . . . 17
2.4.1 Verwaltungsschalen . . . 17
2.4.2 Fähigkeiten . . . 18
2.4.3 Skill-Based Production . . . 19

3 Verwandte Arbeiten **20**

4 Konzeption **26**
4.1 Einführung . . . 26
4.1.1 Ziel des Systems . . . 26
4.1.2 Systemumgebung . . . 27
4.2 Soll-Konzept . . . 28
4.2.1 Systemfunktionalität . . . 28
4.2.2 Systemkontext . . . 36
4.2.3 Anforderungen . . . 42
4.2.4 Primäre APIs . . . 46
4.2.5 Komponenten . . . 54
4.3 Zusammenfassung . . . 71

5 Konzeptprüfung **72**
5.1 Entwicklungsansatz . . . 72
5.2 Vorgehensweise . . . 73
5.3 Kernkomponenten . . . 74
5.3.1 Descriptor Factory . . . 74
5.3.2 Test Ticket . . . 84

5.3.3 Ticket Builder . . . 86
5.3.4 Test Object . . . 97
5.3.5 Test Runner . . . 100
5.3.6 Ticket Repository . . . 103
5.3.7 Test Object Provider . . . 111
5.3.8 Test Orchestrator . . . 119
5.4 Dienste . . . 129
5.4.1 Ticket Repository Service . . . 129
5.4.2 Test Orchestration Service . . . 142
5.5 Verwaltungsschale . . . 147
5.5.1 Ticket Management Submodel . . . 148
5.5.2 Test Execution Submodel . . . 154
5.6 Testklassen . . . 155
5.6.1 Strukturtests . . . 155
5.6.2 Zustandsbezogene Tests . . . 157
5.7 Inbetriebnahme . . . 160
5.8 Zusammenfassung . . . 162

6 Fazit **164**
6.1 Zusammenfassung . . . 164
6.2 Kritischer Rückblick . . . 165
6.3 Ausblick . . . 166

Abbildungsverzeichnis **I**

Tabellenverzeichnis **III**

Source Code Content **VII**

Glossar **VIII**

Abkürzungsverzeichnis **X**

Literaturverzeichnis **XII**

Onlinereferenzen **XVII**

Anhang A **XIX**

Eigenständigkeitserklärung **XCVI**

Kapitel 1

Einleitung

1.1 Motivation

Nach Chryssolouris wird das Zeitalter der Massenproduktion durch die Ära der Marktnischen abgelöst [Chr06]. Während in der Vergangenheit die Produktentwicklung und die Auftragsbearbeitung vor der Auftragserfüllung und der Distributionslogistik priorisiert wurden, richtet sich heutzutage der Fokus auf die zuverlässige Lieferung kundenspezifischer Produkte in global verteilten Märkten [Wie07]. Dieser Fokuswechsel bringt eine Reihe von global-wirtschaftlichen Veränderungen und Herausforderungen mit sich. Als Beispiele dienen neben der explodierenden Anzahl von Produktmodellen und den daraus resultierenden mutierten Produktlebenszyklen, der Bedarf an Agilität der Industrieanlagen zur effizienten Individualisierung der Produktvarianten und die standortübergreifende bzw. dezentralisierte Produktion [Wie07], [Sch09]. In [Sch09] informiert Janorschke, dass die nachhaltige Wettbewerbsfähigkeit im Kontext der Wertschöpfungskette zusammen mit den verkürzten Lebens- und Innovationszyklen eine fortlaufende Umstellung der bereits etablierten Produktionsphasen und ihrer Strukturen verlangen. Demzufolge werden Unternehmen und Organisationen, deren Produktionseinrichtungen sich nicht an die fluktuierenden Rahmenbedingungen, die sog. Turbulenzen, adaptieren können, es zunehmend schwer haben auf die sich ständig verändernden Marktbedürfnissen zu reagieren [WZ09].

Laut [Wie07] spielen die ressourceneffizienten Produktentwicklungszyklen, welche qualitativ hochwertige Produkte in ausreichender Menge produzieren, eine Schlüsselrolle für das Erfüllen der Anforderungen eines diversifizierten Kundenstamms. Wie bereits erwähnt wurde, gilt u.a. die anforderungsorientierte Wandlungsfähigkeit der Produktionsanlagen und ihrer Komponenten als zentrales Erfolgskriterium. Gemäß der Spezifikation der Industrie 4.0 Plattform entspricht die Wandlungsfähigkeit der "*Eigenschaft eines Systems, es mit geringem Arbeitsaufwand an Änderungen, die zum Planungszeitpunkt nicht bekannt waren, anzupassen*"[Bos22]. Wiendahl definiert diese als "*characteristics to accomplish early and foresighted adjustments of the factory's structures and processes on all levels to change impulses economically*" [Wie07]. Der Bedarf an agilen Produktionsumgebungen und die fortschreitende Verbreitung der Informations- und Kommunikationstechnologien haben wachsende Anforderungen an die Anlagen und ihre Komponenten zur Folge. Laut [Wey18] lässt sich die daraus resultierende Komplexitätssteigerung durch eine Unterteilung der betrachteten Systeme in deren Bestandteile reduzieren. Hierfür ist eine klare Definition der entsprechenden Schnittstellen erforderlich.

Es lässt sich behaupten, dass zur Umsetzung einer wandelbaren Produktionsanlage die Möglichkeit des effizienten Austauschs oder Rekonfiguration bzw. Aktualisierung jeder variationsfähigen oder ersetzbaren Entität vorausgesetzt wird. Sowohl im Bereich der Softwareentwicklung als auch für die Realisierung wandelbarer Produktionsanlagen dient die Abstraktion als eine wichtige Voraussetzung zur Bewältigung der Komplexität. Jede Entität in einem agilen Produktionsprozess muss einen virtuellen Vertrag erfüllen, der auch als Interface bezeichnet wird. Dieses Interface stellt das gemeinsame Verhalten bzw. Informationsbasis aller Entitäten vom gleichen Typ dar, wodurch sie unter Einhaltung bestimmter Bedingungen dynamisch ein- und ausgewechselt werden können, ohne den Produktionszyklus negativ zu beeinflussen. Dies ermöglicht es den Herstellern, sich über dynamisch austauschbare Entitäten schnell an sich ändernde Kunden- bzw. Marktanforderungen anzupassen. Des Weiteren spielt für die agile Produktion auch die Gestaltung des allgemeinen Engineering-Prozesses einer industriellen Anlage eine wichtige Rolle. Dieser

ist so zu organisieren, dass auf Basis des entsprechenden Anforderungs- bzw. Fähigkeitsprofils eine dynamische Reorganisation und Parametrierung der Produktionseinrichtung zum Zweck der Herstellung eines kundenspezifischen Produktes stattfinden kann. Hierfür sind neben den notwendigen Interfaces u.a. generische Modelle für die maschinenlesbare Spezifikation von Anforderungen, Produktionsrezepten und Fähigkeiten sowie geeignete Methoden für ihre Ableitung und Distribution erforderlich.

Laut [DW16] dient die Industrie 4.0 Vision als eine technologische Antwort der aktuellen globalen und wirtschaftlichen Veränderungen. Das Konzept basiert auf die Integration der modernen Informations- und Kommunikationstechnologien in der Produktion und spezifiziert die Verwaltungsschale (VWS, engl. AAS) als einen wesentlichen Bestandteil jedes Produktionsgegenstandes, der als Basis für die Interoperabilität einer Industrieanlage gilt. Des Weiteren definiert die Industrie 4.0 Plattform Engineering-Prozesse und Instrumentierungen als potenzielle Methoden und Werkzeuge für die Umsetzung flexibler Produktionsumgebungen. Als Beispiel gilt den Begriff *Skill-Based Production*, welcher im Kontext der deutschen Produktion immer häufiger als die "*Produktionsarchitektur der Zukunft*", siehe [Mar22], verwendet wird und auf dem Konzept der Verwaltungsschale basiert. In einem Livestream der SmartFactory-KL informiert Peschke, dass als Hauptmotivation für die skill-basierten Produktionssysteme die Flexibilität des Produktionsprozesses und die schnelle Reaktion auf Anforderungen ist [Mar22].

Als Folge des wachsenden Bedarfs an die anforderungsorientierte und autonome Wandelbarkeit der Produktionsanlagen und der daraus resultierenden Komplexitätssteigerung, gewinnen ganzheitliche Lösungsansätze zur Qualitätssicherung der Automatisierungskomponenten und ihrer funktionalen Eigenschaften immer mehr an Bedeutung. Während das RAMI 4.0 Modell zu einer Komplexitätsreduktion führt und die wichtigsten Aspekte der Industrie 4.0 Vision in der Form einer Schichtenarchitektur veranschaulicht, beschreibt das Referenzmodell keine Methoden zur Validierung der korrekten Funktionsweise der bestehenden Anlagenressourcen [Wey18]. Es stellt sich die Frage, wie und warum sich die Verwaltungsschalen für die automatisierte Validierung der funktionalen und nicht-funktionalen Eigenschaften der bestehenden Anlagenkomponenten verwenden lassen. Hierfür wird auf die Methoden der Softwaretestung zurückgegriffen und die Rolle der Verwaltungsschalen als Werkzeug zum Testen von Komponenten in wandelbaren Produktionsanlagen untersucht.

1.2 Zielsetzung

Ziel dieser Arbeit ist es herauszufinden, ob und wenn ja, warum die Nutzung von Verwaltungsschalen in flexiblen Produktionsanlagen positive Auswirkungen auf den industriellen Validierungsprozess zur Folge hat und wie sich diese darstellen. Des Weiteren soll untersucht werden, welche Verfahren bzw. Formen der Testdurchführung bereits in den unterschiedlichen Domänen existieren, inwiefern bestehende Methoden und Technologien für das Testen I4.0 konformer Anlagenkomponenten auf Basis der Verwaltungsschale (VWS) geeignet bzw. adaptierbar sind und was unter „Fähigkeiten" im Zuge der Automatisierungstechnik und Industrie 4.0 verstanden wird. Als Schwerpunkt der Arbeit gilt die Auseinandersetzung mit den Literaturquellen, welche die Thematik der Validierung von Industrie 4.0 Komponenten behandeln, und die darauffolgende Konzeption und Synthese einer erweiterbaren Softwarearchitektur, welche als Basis für die Realisierung einer prototypischen Testautomatisierungslösung gilt. Die zu erarbeitende Testmethode soll eine semi-automatisierte Validierung Industrie 4.0 konformer Anlagenkomponenten gewährleisten, indem sie die Kernkonzepte der Industrie 4.0 Vision mit traditionellen Softwareteststrategien kombiniert und die VWS in den Prozess der Automatisierung und Auswertung der Testaktivitäten einbezieht.

1.3 Vorgehensweise und Aufbau der Arbeit

Im zweiten Kapitel werden die zentralen Begriffe der Abschlussarbeit erläutert. Zu diesen gehören die Aspekte des Softwaretests mit Fokus auf Testmethoden im Bereich der eingebetteten Systeme und der Testautomatisierung. Weiterhin werden die Qualitätsmerkmale und Metriken im Bereich der Softwareentwicklung sowie die etablierten Testverfahren in der Automobilindustrie und der Automatisierungstechnik diskutiert. Im Mittelpunkt der Diskussion liegt die Verifizierung von Anlagenkomponenten durch die Methoden der virtuellen Inbetriebnahme. Als weiteren in Zusammenhang mit dem Thema grundlegenden Aspekten gelten die Verwaltungsschalen, ihre semantische Architektur und Schnittstellen sowie die Fähigkeiten als abstrakte und in der VWS enthaltene Beschreibungen von konkreten ausführbaren Einheiten.

Im Rahmen des dritten Kapitels werden existierende Lösungsmethoden zur Automatisierung des industriellen Testprozesses beschrieben und kritisch bewertet. Es muss also eine Auseinandersetzung mit den in der Literatur vorgeschlagenen Testansätze erfolgen, ihre Übertragbarkeit in Zusammenhang mit der VWS analysiert und die Reproduzierbarkeit untersucht werden. Ein weiterer wichtiger Punkt befasst sich mit der Testvision des Industrie 4.0 Konzeptes und den Voraussetzungen, welche zum Zweck ihrer Realisierung erfüllt werden müssen.

Darauf basierend wird in Kapitel 4 eine Abgrenzung stattfinden und eine Teil- oder Komplettlösung erarbeitet, die für bestimmte Aspekte der Aufgabenstellung qualifiziert ist. Als mögliche Aspekte, welche durch die Lösung adressiert werden können, dienen Struktur-, Verhalten-, Schnittstellen- und Interoperabilitätstests. Zum Zweck der Nachvollziehbarkeit muss das erarbeitete Lösungskonzept mit den entsprechenden Softwaredesign-Tools UML-konform dokumentiert werden.

Kapitel 5 wird der Beschreibung der testgetriebenen Entwicklung gewidmet. Im Rahmen des Kapitels werden also sowohl die Szenarien zur Validierung der Softwarearchitektur, als auch ihre Entwicklung beschrieben. Der Testaufbau wird zusammen mit den Funktionstests im gleichen Kapitel behandelt.

Zum Abschluss wird das Konzept und dessen Implementierung kritisch diskutiert. Der Nutzwert der Lösung wird analysiert und potenzielle Verbesserungen bzw. zugelassene Kompromisse werden dargelegt. Die Vor- und Nachteile der Lösung werden ebenfalls behandelt. Die Arbeit wird mit einem Resümee schließen.

Kapitel 2

Grundlagen und Stand der Technik

In diesem Kapitel werden die zentralen Begriffe der Abschlussarbeit erläutert. Im Fokus liegen die etablierten Test-Verfahren im Bereich der Softwareentwicklung und die Strategien zur Testautomatisierung. Des Weiteren werden auch Test-Qualitätsmerkmale diskutiert und Verifikationsmethoden aus der Automobilindustrie und der Automatisierungstechnik behandelt. Zudem erfolgt eine Konkretisierung der Begriffe „Verwaltungsschale“, „Fähigkeit“ und „Skill“ im Kontext der Industrie 4.0 Vision.

2.1 Softwaretest

Es lässt sich behaupten, dass eine Erhöhung der Konfigurationsfähigkeit der Produktionsgegenstände auf Basis der Softwaretechnik, durch geeignete Softwarelösungen und Softwareinfrastrukturen, als Voraussetzung für eine agile Produktionsanlage gilt. Der zunehmende Fokus auf Software-Tests lässt sich dadurch erklären, dass alle Komponenten einer Anlage von Programmcode betrieben werden.

2.1.1 Definitionen und Ziele

Laut [Raz18] entspricht das Software-Testing einem bedeutenden Bestandteil des Softwareentiwcklungsprozesses. Nach [UA19] ist die Testbarkeit eines der wichtigsten Softwarequalitätsmerkmale, die den Erfolg des Softwareproduktes bestimmen. Als typische Ursachen für inkorrekte Software gelten neben Zeitdruck, schlechter Dokumentation und klassische Programmierfehler, fehlerhafte oder fehlende Kommunikation im Rahmen der Anforderungsspezifikation, sich ständig ändernde Anforderungen und die steigende Softwarekomplexität [Grü13].

In DIN EN ISO9000:2015-11 wird unter Test die Tätigkeit zur Ermittlung von Merkmalen bzw. Merkmalswerten unter Einhaltung der Anforderungen an einen spezifischen beabsichtigten Gebrauch oder eine konkrete und vordefinierte Anwendung verstanden [Sta15]. Laut dem Standard lassen sich Testergebnisse, die Konformität zeigen, als objektive Nachweise zum Zweck einer Validierung verwenden. Im Rahmen des Validierungsprozesses findet die Bestätigung statt, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine konkrete und vordefinierte Anwendung erfüllt worden sind.

Es stellt sich die Frage, anhand welcher Informationen es sichergestellt werden kann, dass die aufgezeichneten Ergebnisse ein korrektes Verhalten nachweisen und welche Aktivitäten mit dieser Tätigkeit assoziiert werden können. Konkrete Antworten liefert die Definition des International Software Testing Qualifications Board (ISTQB) [Mul10].

Das ISTQB definiert als Testung den Prozess, der alle statischen und dynamischen Lebenszyklusaktivitäten umfasst, die sich mit der Planung, Vorbereitung und Bewertung eines Gegenstands und der zugehörigen Arbeitsergebnisse befassen [Mul10]. Ein Ziel dieses Prozesses ist es, sowohl sicherzustellen, dass das Verhalten des Testobjekts im operativen Betrieb der Anforderungsspezifikation entspricht, als auch Abweichungen von der Spezifikation zu erkennen, damit Defekte und potenzielle Fehlerzustände rechtzeitig eliminiert werden können. In diesem Sinne lässt sich eine Softwarekomponente, die sich für ein

bestimmtes Szenario gemäß der Spezifikation verhält, in allen anderen Fällen aber davon abweicht, für das entsprechende Szenario als gültig betrachten.

Nach der allgemeinen Definition des Testbegriffs erfolgt eine Konkretisierung des Begriffs Softwaretest. Laut [V14] entspricht ein Softwaretest dem Test, welcher im Laufe eines Softwareentwicklungsprozesses durchgeführt wird, auf Basis der Anforderungsspezifikation die Softwarefunktionalität bewertet und die Qualität der Software misst. Durch die Behebung der im Rahmen der Softwaretests identifizierten Fehler lässt sich diese Qualität verbessern. Als Basis für die Softwaretests dient die Anforderungsspezifikation. Im Rahmen des Softwareengineeringsprozesses wird aus der Anforderungsspezifikation eine Softwarearchitektur bzw. ein Softwaredesign abgeleitet. Die Testobjekte werden gegen ihren im Rahmen des Softwaredesigns spezifizierten Zweck verifiziert [Grü13]. Aus diesem Grund informiert Grunfelder, dass Fehler im Rahmen der Planungsphasen von den Testverfahren der Softwareentwicklung nicht adressiert werden können. Während solche Probleme durch Methoden des Projektmanagements vermeidbar sind, eignen sich die Verfahren aus dem Softwaretest nur für klassische Programmier- und Designfehler.

Es ist hervorzuheben, dass die Durchführung von Softwaretests keine Abwesenheit von Fehlern, sondern ein korrektes Verhalten bei vorher festgelegten Eingangsdaten für bestimmte Testszenarien garantieren kann [Dij70]. Der Grund dafür liegt darin, dass die Verifikation einer Softwarekomponente mit allen möglichen Eingangsdaten oft unpraktisch und ressourcenineffizient ist. Laut [Aru19] entspricht eine der Herausforderungen des Testens der Komprimierung einer hohen Anzahl von Tests in einem überschaubaren Test-Set. Kleuker kommentiert, dass als weiteres Ziel die Lokalisierung von möglichst vielen Fehlern mit möglichst wenigen Testfällen gilt [Kle19].

2.1.2 Testaktivitäten

Neben der Testdurchführung existieren laut [Mul10] auch andere Tätigkeiten, die für den Testprozess bedeutend sind. Das ISTQB definiert die folgenden Aktivitäten als Teil des fundamentalen Testprozesses: Testplanung und Kontrolle, Analyse und Design, Implementierung und Ausführung, Bewertung von Ausgangskriterien und Protokollierung sowie Aktivitäten zum Testabschluss.

Laut [HK19] umfasst die Planungsphase u.a. die Definition der konkreten Testziele, die Spezifikation welche Testarten für die Verifikation und die Validierung des entsprechenden Systems geeignet sind, welche Voraussetzungen für ihre Anwendung erfüllt werden müssen, auf welche Instrumentierung die Testdurchführung basieren soll und wie die Ergebnisse des Testprozesses zu protokollieren sind. Als weitere Testplanungsaktivitäten gelten die Definition der Zulassungs- und Ausgangskriterien für jede Teststufe und die Durchführung von Risikoanalysen, für mehrere Information siehe [Mul10, S. 49]. Das ISTQB definiert als Kontrolle des Testprozesses die Durchführung von Aktivitäten zur Einhaltung des Testplans. Dazu gehört u.a. die Überwachung des gesamten Testprozesses anhand geeigneter Metriken, die Erstellung von Testreports und die strategische Anpassung des Testplans auf Basis der gesammelten Informationen.

Im Rahmen des Analyse- und Designvorgangs werden die festgelegten Testziele auf Basis von high-level Testszenarien konkretisiert. Eine Voraussetung dafür ist die Auseinandersetzung des Testers mit der Architektur der zu validierenden Software, den Anforderungen, der Spezifikation der Schnittstellen und den weiteren im Rahmen des Softwareentwicklungsprozesses erstellten Artefakten [Mul10, S. 15]. Laut Grünfelder [Grü13, S. 46] gilt als Ziel dieses Analysevorgangs die Verifikation, dass die Softwarearchitektur für das entsprechende Anwendungsszenario optimal ist und einem bestimmten Testbarkeitsgrad entspricht. Als Voraussetzung für eine hohe Test- und Wiederverwendbarkeit der Software dient der angemessene Einsatz von Design Patterns. Als weitere Aktivitäten des Analyse- und Designvorgangs definiert ISTQB die Identifikation von Testkonditionen in Abhängigkeit von der Struktur der Testobjekte, die Spezifikation und Priorisierung von Testszenarien und die Vorbereitung von Testdaten [Mul10].

Voraussetzung für die Implementierung und Ausführung der Testfälle ist die Spezifikation der Daten, mit welchen die Tests operieren und anhand welcher die Funktionsweise der Testobjekte verifiziert wird, die Priorisierung der Szenarien, und die Vorbereitung der Testumgebung. Innerhalb der Testumgebung werden ausführbaren Testklassen (Test-Suites) definiert, die eine Sammlung von Testobjekten beinhalten.

Im Rahmen der Ausführung werden die erwarteten mit den tatsächlichen Ergebnissen jedes Testobjekts verglichen. Zu den Aktivitäten der Implementierungs- und Ausführungsphase zählen weiterhin die Erstellung von Skripten zur Automatisierung des Ausführungsprozesses, die Protokollierung der Testergebnissen zusammen mit der verwendeten Softwareversion und den eingesetzten Test-Tools u.v.m., siehe [Mul10, S. 16].

Die letzten zwei Phasen beschäftigen sich mit der Auswertung des Testprozesses. Dies umfasst das Abgleichen der protokollierten Testergebnissen mit den Ausgangskriterien und die Zusammenfassung der erreichten Testziele. Die Abschlussaktivitäten konzentrieren sich auf die Analyse der Testzyklen und der gewonnenen Erkenntnisse.

2.1.3 Testprinzipien

Aus der Beschreibung und der Allgemeinheit der Testaktivitäten geht hervor, dass es keinen konkreten und allgemeingültigen, standardisierten Testprozess gibt, auf welcher Basis jedes Testobjekt auf gleicher Art und Weise verifiziert und validiert werden kann. Aus diesem Grund besagt eines der sieben Test-Prinzipien, dass das Testing kontextabhängig ist, siehe [Bor17]. Muller u. a. argumentieren dieses Prinzip mit der Aussage, dass sicherheitskritische Software anders als eine E-Commerce-Webseite getestet wird, siehe [Mul10, S. 14].

Laut dem zweiten Prinzip ist eine vollständige Testung eines komplexen Testobjekts nahezu unmöglich [Mul10, S. 14]. Als Grund dafür gilt die gegen unendlich konvergierende Anzahl von möglichen Pre- bzw. Postkonditionen, im Kontext der Softwareentwicklung Ein- und Ausgangsdaten, welche eine Basis für die Durchführung unterschiedlicher Testfällen darstellen. Die Durchführung eines zielorientierten Testprozesses setzt die Durchführung von Risikoanalysen zum Zweck der Spezifikation von kritischen Testfällen voraus [Bor17].

Da die Folgen einer verspäteten Identifikation eines Softwarefehlers in Abhängigkeit von der Applikationsdomäne sehr stark variieren können, besagt das dritte Test-Prinzip, dass den Testprozess möglichst frühzeitig beginnen und ergebnisorientiert durchgeführt werden muss [Bor17].

Laut dem vierten Grundsatz, auch als *Defect Clustering* bekannt, werden Softwarefehler in den meisten Fällen nicht gleichmäßig verteilt. Das Prinzip besagt, dass i.d.R. eine niedrige Anzahl von Softwarekomponenten als Ursache für einen großen Anteil von Qualitätsproblemen gilt [Mul10, S. 14]. Die Idee des Grundsatzes ist mit dem sog. Paretoprinzip vergleichbar. Laut diesem sind 80% der Folgen auf 20% der Ursachen zurückzuführen.

Das fünfte Prinzip - *Pesticide Paradox* - besagt, dass die Verifikation einer sich ständig weiterentwickelnden Software durch die Wiederholung der gleichen Testfälle, ihre Entwertung garantiert. Die Entwicklung und Integration weiterer Komponenten und Funktionalitäten setzt die Implementierung neuer und die Modifikation der bestehenden Testroutinen voraus. Ein Testprozess lässt sich somit als eine kontinuierliche Durchführung von Testaktivitäten bezeichnen. Als Ziel gilt die Identifikation von möglichst vielen Softwarefehlern durch die permanente Erstellung von weiteren Testszenarien [Mul10].

Gemäß dem sechsten Grundsatz kann durch den Testprozess gezeigt werden, dass Defekte vorhanden sind. Die Aktivität garantiert jedoch die Abwesenheit von Fehlern nicht [Bor17].

Laut dem siebten Test-Prinzip gilt die Testdurchführung als kein Hilfsmittel gegen inkorrektes Softwaredesign.

Für weitere Informationen siehe [Mul10, S. 14] und [Bor17].

2.1.4 Teststufen

Zur Reduktion des Komplexitätsgrades eines Gegenstands, ist seine Aufteilung in Komponenten erforderlich, die über Schnittstellen interagieren, vgl. mit [Wey18]. Dieser allgemeingültige Ansatz korrespondiert mit der Vorgehensweise in der Softwareentwicklung, wo das Gesamtprodukt systematisch, durch Entwicklung und Integration kleinerer Softwareeinheiten, realisiert wird. Die korrekte Definition und fehlerfreie Implementierung dieser Einheiten und ihrer Schnittstellen spielen eine kritische Rolle für die Interoperabilität der einzelnen Subsysteme bzw. für die korrekte Funktionsweise des Gesamtsystems. Aufgrund dieses hierarchischen und allgemeingültigen Softwareentwicklungsansatzes und seiner Unabhängigkeit vom ausgewählten Entwicklungsmodell ist eine Unterteilung der unterschiedlichen Softwaretests in Teststufen sinnvoll. Während sich die Unit-, Integrations-, System- und Akzeptanztests als die primären Testmethoden etabliert haben, unterscheidet man auch zwischen anderen Softwareteststrategien.

Unit-Testing

Die Durchführung von Unit- bzw. Komponenten- oder Modultests ist der erste Verifikationsprozess im Rahmen des Entwicklungszyklus einer Software und gilt als Voraussetzung für alle nachfolgenden Testphasen. Laut [SIG17] ist der Zweck dieser Verifikation einen Nachweis zu erbringen, ob die Softwareeinheiten mit dem detaillierten Software-Entwurf und den nicht-funktionalen Anforderungen übereinstimmen. Der internationale Standard IEC 61508 erfordert die Verifikation der Software durch den Einsatz von Unit-Tests [Raz18]. Im Rahmen der Modultests werden einzelne Softwareeinheiten, Funktionen bei den prozeduralen und Methoden bei den objektorientierten Programmiersprachen, isoliert gegen der Spezifikation nach dem Eingabe-Verarbeitung-Ausgabe Prinzip getestet. Dies erfolgt in den meisten Fällen durch eine Äquivalenzklassenbildung und darauffolgende Grenzwertanalyse [Grü13]. Für unterschiedliche Kombinationen von Eingangsgrößen muss der Unit ein bestimmtes und im Rahmen der Anforderungsanalyse spezifiziertes Verhalten aufweisen. Ob das Verhalten der entsprechenden Funktion bzw. Methode mit diesem, welches spezifiziert wurde, übereinstimmt, wird anhand der Testdaten verifiziert. Als Grundlage für den Testprozess gelten die Anforderungen an die Komponenten, das detaillierte Software-Design und die Code-Basis.

Die Modultests charakterisieren sich durch ein bestimmtes Entwurfsmuster - das sog. *3A-Pattern*. Das Ziel des Musters ist es, den Testcode organisieren, indem ein Testfall in drei aufeinander folgende Phasen unterteilt wird - *Arrange*, *Act* und *Assert*. Im Rahmen der ersten Phase werden die Bedingungen erfüllt, die als Voraussetzungen für die Ausführung des Testobjekts dienen. Laut Microsoft umfasst die *Arrange*-Sektion die Initialisierung von Daten und Objekten, die dem Testobjekt übergeben werden. Khorikov definiert die Phase als das Bringen des zu testenden Systems und seiner Abhängigkeiten in einem gewünschten Zustand [Kho20]. Im Rahmen der nächsten Phase wird das Testobjekt mit den entsprechenden Daten aufgerufen. Der *Assert*-Vorgang verifiziert, dass die Ergebnisse aus der Ausführung der SUT dem erwarteten Zustand entsprechen. Dies kann anhand einer Überprüfung des Rückgabewertes oder der unterschiedlichen Attributen erfolgen.

Da mittels Unit-Tests die Testobjekte unabhängig von der Gesamtsoftware verifiziert werden müssen, sind konkrete Methoden für die Isolation der zu testenden Einheiten erforderlich. Laut [Kho20] besteht der Vorteil dieses Ansatzes in der klaren Identifikation der Defektursache bei nicht bestandenen Testfällen. Außerdem diskutiert Khorikov, dass diese Vorgehensweise eine klare Strukturierung der Tests garantiert. Der Grund dafür liegt daran, dass jeder Klasse der Applikation eine korrespondierende Testklasse zugeordnet werden kann, was eine Erhöhung der Übersichtlichkeit bei komplexeren Projekten zur Folge hat. Die Isolation des Testobjekts erfolgt mittels der sog. *Test-Doubles* bzw. *Stubs*, die nur zum Zweck der Verifikation verwendet werden und das Verhalten von realen Objekten simulieren. Eine Voraussetzung für die Kontrolle der Abhängigkeiten und die hohe Testbarkeit ist die Verwendung des Prinzips *Dependency Inversion*, siehe [Cla13].

Integration-Testing

Im Rahmen der Integrationstests werden die Units als Gruppe nach dem 3A-Muster getestet und die Schnittstellen der voneinander abhängigen Komponenten verifiziert. Es findet keine Isolation der Units statt. Das ISTQB definiert als Basis für diesen Testprozess das Software- bzw. das Systemdesign, die Softwarearchitektur zusammen mit den dokumentierten Abläufen und die Anwendungsszenarien [Mul10]. Laut [SIG17] umfasst den Testvorgang die Integration der im Rahmen der Unit-Testung verifizierten Einheiten in größere Softwarekomponenten, welche einer vollständig integrierten und mit dem Architekturdesign korrespondierenden Software entsprechen, sowie das Sicherstellen, dass die zusammengesetzten Komponenten und ihre Schnittstellen mit dem Softwarearchitekturdesign übereinstimmen. Der Fokus liegt auf den Schnittstellen der Units und dem korrekten Datenaustausch. Eine praxisnahe Definition liefert Osherove. Der Autor definiert die Integrationstestung als "*testing a unit of work without having full control over all of it and using one or more of its real dependencies, such as time, network, database, threads, random number generators, and so on.*" [Osh13, S. 8].

Der Nachteil dieser Teststufe besteht darin, dass im Rahmen der Integrationstests viele Abhängigkeiten realisiert werden, z.B. bei Nutzung externer Bibliotheken, Datenbanken, Durchführung von Systemaufrufen zur Netzwerkkommunikation u.a., was zu einer zeitlichen Ineffizienz und niedrigeren Wartbarkeit im Vergleich zu den Unit-Tests führt [Kho20]. Der entscheidende Vorteil dieser Teststufe liegt darin, dass auf diese Weise eine deutlich größere Code-Basis abgedeckt und die gesamte Softwareinfrastruktur verifiziert wird. Laut [Jor13] schlagen die meisten Software-Engineering Texte vier Integrationsstrategien vor: *Top-Down*, *Bottom-Up*, *Sandwich* und *Big-Bang*. Das International Software Testing Qualifications Board informiert, dass die auszuwählende Vorgehensweise u.a. von der Softwarearchitektur und den Systemfunktionalitäten abhängig ist [Mul10].

Die Top-Down Integration charakterisiert sich dadurch, dass zum Beginn die hierarchisch priorisierten Units integriert werden. Jorgensen kommentiert, dass zum Zweck der Kompilierbarkeit alle anderen Units als Test-Doubles auftreten [Jor13]. Auf Basis des Abhängigkeitsbaums werden dann die Komponenten der untergeordneten Softwareebenen sukzessiv integriert und nach jeder Komponentenintegration die Kommunikation zwischen der aufrufenden und der aufgerufenen Methode verifiziert. Die Systematik dieser Vorgehensweise garantiert, dass ein potenzielles Problemauftritt mit dem zuletzt integrierten Unit zu assoziieren ist. Während die frühzeitige Testung der Schnittstellen, der Logik und des Datenflüsses aller Komponenten auf der höchsten Softwareebene als Argument für dieses Verfahren gilt, führt die Notwendigkeit zur Erstellung von Test-Doubles zu einer Erhöhung des Testaufwands [SK12].

Bei der Bottom-Up Integration werden zuerst die Units der niedrigsten Softwareebene integriert und die sog. Test-Drivers zur Simulation des Verhaltens der aufrufenden Komponenten verwendet. Grünfelder beschreibt den Testtreiber als eine Umgebung, die "*[..] die Funktionen der Module aufruft und das erwartete Ergebnis mit dem tatsächlichen Ergebnis vergleicht*" [Grü13]. Im nächsten Schritt erfolgt die Ersetzung der Test-Drivers durch die realen Komponenten, die Erstellung von weiteren Treibern für die bereits integrierten Units und die Wiederholung des Testvorgangs. Der Integrationsprozess erfolgt bis alle Subsystemen getestet sind. Laut Singh und Khan ist ein Vorteil der Bottom-Up Integration das frühzeitige Sicherstellen, dass die grundlegenden Softwareeinheiten korrekt miteinander interagieren [SK12]. Ein weiteres Argument für diese Integrationsart ist die Abwesenheit von Test-Doubles. Dagegen sprechen die verzögerte Validierung der höchsten Ebenen und die Notwendigkeit zur Implementierung von Testtreibern [SK12].

Voraussetzung für die Big-Bang Integrationsstrategie ist, dass alle Module bereit für Integration sind [Jor13]. Im Gegensatz zu den anderen Verfahren ist hier keine systematische Vorgehensweise erkennbar, da die Verifikation erst nach der Integration aller Komponenten stattfindet. Während diese Strategie aufgrund der zahlreichen Abhängigkeiten zu einer sehr schweren Fehlerlokalisierung führt und von dem International Software Testing Qualifications Board abgeraten wird, besteht hier den Vorteil, dass keine Test-Doubles oder Test-Drivers von den Testern bereitgestellt werden müssen [Mul10].

Das Sandwich-Verfahren kombiniert die Bottom-Up mit der Top-Down Integration, indem es sich auf die Konstruktion von end-to-end Testpfaden orientiert [Jor13]. Es werden also ganze Pfade von der höchsten bis zur niedrigsten Softwareschicht gebildet und die Schnittstellen und Funktionsweise der integrierten

Units verifiziert. Diese Integrationsart erfordert ebenfalls keine Implementierung von Test-Doubles oder Testtreibern.

System-Testing

Der IEEE 620.12 Standard definiert das Systemtesting als die Testung eines integrierten Systems zum Zweck der Verifikation, dass dieses den spezifizierten Anforderungen entspricht, vgl. mit [IEE90, S. 74]. Jorgensen kommentiert, dass das Ziel der Systemtests der Demonstration des korrekten Systemsverhaltens entspricht [Jor13]. Das International Software Testing Qualifications Board gibt als Testbasis für diesen Prozess die System- und Softwareanforderungsspezifikation, die Anwendungsszenarien des Systems zusammen mit der funktionalen Spezifikation sowie Berichte über Risikoanalysen an [Mul10].

Laut dem International Software Testing Qualifications Board fokussieren sich die Systemtests auf das Verhalten des Gesamtsystems indem sie die funktionale bzw. nicht-funktionale Systemanforderungen und die Qualitätsmerkmale überprüfen. Eine Voraussetzung für die sinnvolle Durchführung der Funktionstests ist also ihre Zuordnung zu den entsprechenden Anforderungen. Als funktionale Anforderungen gelten alle, die Funktionalitäten spezifizieren, welche ein System oder eine Systemkomponente ausführen können muss [III17]. Des Weiteren definiert das ISO/IEC/IEEE 24765:2017 Standard als Funktionstestung den Testvorgang, bei dem die interne Mechanismen eines Systems oder einer Komponente ignoriert werden und der Fokus auf den Ausgaben liegt [III17]. In diesem Zusammenhang wird oft der Begriff *Black-Box-Testing* verwendet, der im folgenden Abschnitt zusammen mit weiteren Testtypen erläutert wird. Als nicht-funktionale Anforderungen gelten i.a. Qualitätsmerkmale, Performanzanforderungen, Schnittstellenspezifikationen u.a. Diese müssen im Rahmen des Systemtestvorgangs auch berücksichtigt und protokolliert werden.

2.1.5 Testtypen

Es existieren unterschiedliche Typen von Testmethoden, die sich auf unterschiedliche Softwareaspekte orientieren. Das International Software Testing Qualifications Board gibt folgende Aspekte an: Funktionalität, nicht-funktionale Qualitätsmerkmale und Softwarestruktur, vgl mit [Mul10, S. 27]. Während die funktionale und nicht-funktionale Anforderungen mittels der Methoden des *Black-Box-Testing*, auch Spezifikationsbasiertes Testing genannt, verifizierbar sind, eignen sich für die Übeprüfung der Softwarearchitektur die sog. *White-Box-Tests* - Strukturbasierte Tests. In [Grü13] erläutert Grünfelder unterschiedliche Methoden zur Verifikation funktionaler und nicht-funktionaler Anforderungen. Als Beispiele gelten die Black-Box-Testmethoden, welche auf Verfahren wie Äquivalenz-Partitionierung, Grenzwertanalyse, Entscheidungstabellen, Zustandstransitionen, Anwendungsszenarien u.a basieren. Die nicht-funktionale Anforderungen werden z.B. durch Performanz- und Stresstests verifiziert [Mul10]. Das strukturbasierte Testen erfolgt durch das Messen der Testabdeckung. Als etablierte Methoden dienen die Anweisungs- und Entscheidungsüberdeckung.

Black-Box Tests

Der Ansatz der Äquivalenz-Partitionierung, auch Äquivalenzklassenbildung genannt, ist für alle Teststufen relevant und besteht in der Aufteilung der Menge der Eingangsdaten in unabhängigen Klassen [Kle19]. Das Verhalten des Testobjekts unterscheidet sich in Abhängigkeit davon, aus welcher Klasse die Eingangsdaten stammen und muss bei Input-Informationen, die zu *Exceptions* führen bzw. für welche die Software nicht geeignet ist, ebenfalls verifiziert werden. Jeder Menge aus Eingangsdaten kann eine Menge aus Ausgangsdaten zugeordnet werden. Zum Zweck des hohen Abdeckungsgrads bei gleichzeitig reduzierten Testfällen werden aus jeder Klasse die Grenzwerte entnommen und dem Testobjekt übergeben. In diesem Fall handelt es sich um eine Äquivalenzklassenbildung mit Grenzwertanalyse, für mehrere Informationen siehe [Grü13, S. 87].

Im Fall wo ein Testobjekt mehrere Kombinationen aus Eingangsdaten zum Zweck der vollständigen Funktionstestung übergeben bekommen muss, eignet sich der Einsatz von Entscheidungstabellen. Diese dienen zur Darstellung aller zweckmäßigen Kombinationen aus unterschiedlichen Eingangsparametern,

die eine spezifizierte Aktion auslösen. Laut dem International Software Testing Qualifications Board identifizieren die Entscheidungstabellen die komplexeren Geschäftsregeln, welche vom Testobjekt gemäß der Anforderungsspezifikation implementiert werden müssen, und formulieren Kombinationen aus logischen Eingabeargumenten, welche die durchzuführende Aktion entscheiden [Mul10]. Jede Spalte der Tabelle definiert die Eingabebedingungen und welche Aktion vom Testobjekt durchgeführt werden muss. Da Testobjekte mit einer hohen Anzahl von Input-Argumenten viele Testszenarien erfordern, lässt sich die Anzahl der Testfälle durch Verfahren wie Paarweises Testen und Klassifikationsbäume verringern, siehe [Grü13, S. 160].

Bei den zustandsbasierten Tests wird das System als ein Zustandsautomat betrachtet. Im Fokus stehen die Zustände, die Transitionen und die Ereignisse, welche diese uslösen, sowie die jedem Zustand zugeordneten Aktionen [Mul10]. Laut Grünfelder charakterisiert sich das zustandsbasierte Testverfahren durch die Initiierung von Transitionen zum Zweck der Abdeckung von möglichst vielen Pfaden und Zuständen eines Systems. Das International Software Qualifications Board definiert als weitere Testszenarien die Verifikation typischer Pfaden, die Überprüfung jedes Zustands und jeder Transition, die Simulation ungültiger Transitionen u.a., vgl. mit [Mul10, S. 40].

Die Tests auf Basis von Anwendungsszenarien stellen die korrekte Funktionsweise des Systems aus Sicht des Anwenders sicher und verifizieren das sog. *Happy Path.* Muller u. a. informieren, dass jedes Anwendungsszenario durch bestimmte Vor- und Nachbedingungen charakterisiert wird. Eine Basis für die Entscheidung, ob der Test erfolgreich durchgeführt wurde, kann von den Nachbedingungen gebildet werden. Die Anwendungsszenarien können sowohl in Form von System- als auch mittels Integrationstests validiert werden. Da durch die Use-Cases die direkte Interaktion des Benutzers mit dem System dargestellt wird, spielt das Use-Case-Testing eine Schlüsselrolle für die Qualitätssicherung und die Kundenzufriedenheit.

White-Box Tests

Das Ziel der Anweisungsüberdeckung (engl. *Statement Coverage*) ist es, eine Metrik zu geben, anhand welcher die Anzahl der ausgeführten Anweisungen eines Testobjekts gemessen wird. Der Überdeckungsgrad hängt hier von der Anzahl der abgedeckten Anweisungen geteilt durch ihre Gesamtanzahl ab. Laut Grünfelder, Muller u. a. ist die Anweisungsüberdeckung allein kein aussagekräftiges Verfahren [Grü13], [Mul10].

ISO/IEC/IEEE 29119-1:2022 definiert die Entscheidungsüberdeckung als strukturbasiertes Testing, welches auf die Ausübung von Entscheidungsausgängen im Kontrollfluss des Testobjekts basiert, vgl mit [III22]. Als Voraussetzung für die Nutzung der Methode dient also die Verwendung von Konstrollstrukturen für die Implementierung des Testobjekts. Muller u. a. geben an, dass im Rahmen dieser Technik Testdaten spezifiziert und dem Testobjekt übergeben werden, durch welche möglichst viele Pfade innerhalb der internen Struktur des Testobjekts verifiziert werden. Der Überdeckungsgrad hängt von der Anzahl der abgedeckten Entscheidungsergebnisse geteilt durch ihre Gesamtanzahl ab. Diese Methode ist besonders bei den Unit-Tests von Bedeutung, jedoch für alle Teststufen relevant [Mul10], [Kle19].

Laut dem ISTQB existieren auch weitere Techniken für die Durchführung von White-Box Tests, für mehrere Informationen siehe [Grü13] und [Jor13].

2.2 Testautomatisierung

Eine wichtige Rolle für die effiziente Validierung der Funktionsweise einer industriellen Anlage nach durchgeführter Rekonfiguration oder Komponentenintegration spielt die Testautomatisierung. Weyrich und Zeller sprechen in diesem Zusammenhang über die sog. Selbsttestfunktionalitäten, die als Basis für eine effiziente Migration der Verifikationsphasen dienen [WZ16]. Laut [WZ16] verlangen die agilen Produktionsanlagen die Integration von automatisierten Testprozessen, die im operativen Betrieb ausgeführt werden. Es stellt sich die Frage, was genau unter Testautomatisierung verstanden wird, welche Ziele diese hat, und wie sie sich umsetzen lässt.

2.2.1 Definitionen und Ziele

In [Boa16] ist die Testautomatisierung als die Verwendung von Softwarewerkzeugen zur Automatisierung der Testaktivitäten definiert. Es handelt sich also um eine Automatisierung des i.d.R. manuell durchgeführten Testprozesses durch den Einsatz von Technologien, welche sich mit dem Design, der Kontrolle, Implementierung, Ausführung, Bewertung und Protokollierung von Testszenarien beschäftigen. Baumgartner u. a. kommentieren, dass die Aufgabe des Testautomatisierungsprozesses der Bereitstellung aller zum Zweck der Verifizierung notwendigen Testmitteln wie Software, Dokumentation, Testfälle, Testdaten und Testumgebungen entspricht [Bau21]. Das International Software Testing Qualifications Board definiert als primäre Ziele der Testautomatisierung die Effizienzsteigerung des allgemeinen Validierungsprozesses in Hinsicht auf Zeit und Kosten zusammen mit der Verbreiterung des Testspektrums und der steigenden Testdurchführungsfrequenz [Mul10].

Die Vorteile der Testautomatisierung liegen in der flexiblen Gestaltung und Ergänzung der Regressionsteststrategie, parallelisierbaren Testdurchführung, Möglichkeit der automatisierten Wiederholung von sehr komplexen Testsequenzen und der effizienten Ausnutzung verfügbarer Testmitteln, vgl. mit [Bau21, S. 12] und [Bak16, S. 12]. Bakker u. a. definieren als Nachteile dieses Ansatzes die Kosten für die Realisierung der Testinfrastruktur, das angeforderte Fachwissen für ihre zuverlässige Ausnutzung und die potenzielle Komplexitätssteigerung der Testfälle. Durch die limitierten Interpretationsfähigkeiten der Testumgebung in Hinsicht auf Anforderungen und Testszanerien lässt sich die Testautomatisierung als keine Ersatzstrategie der manuellen Testung etablieren [Bau21, S. 13].

2.2.2 Generische Architektur

Die generische Testautomatisierungsarchitektur (TAA) veranschaulicht die wichtigsten Aspekte bei der Realisierung einer Testautomatisierungslösung (engl. *Test Automation Solution - TAS*). Beispielhafte und universelle Testsystem-Architekturen werden von Bakker u. a. sowie Korotkiy und Bender erläutert, wobei die erstere zusätzlich von Baumgartner u. a. in [Bau21] weiter diskutiert wird, siehe [KB10] und [Bak16, S. 23–27].

Laut Baumgartner u. a. besteht eine TAS aus einer Testumgebung bzw. den damit assoziierbaren ausgewählten Testmitteln. Aus diesem Grund lassen sich die TASs als eine Sammlung von Daten bzw. Dateien und miteinander interagierenden Softwarekomponenten und Testwerkzeugen betrachten. Der Entwurf eines Testsystems ist als ein rein softwaretechnisches Projekt anzusehen und erfordert die für diese Disziplin etablierte systematische Vorgehensweise [Bau21, S. 71]. Eine wichtige Voraussetzung für die Implementierung einer qualitativen Softwarearchitektur ist die Einhaltung der sog. SOLID-Prinzipien, siehe [Gre11]. Wichtig ist, dass die konkrete Implementierung der generischen TAA sowohl vom konkreten Anwendungsfall als auch von der Klasse potenzieller Testobjekten und der Fachdomäne abhängig ist. Der Umfang der implementierten Softwareebenen kann in Korrelation mit der entsprechenden Lösung variieren. Die im Lehrplan des ISTQB vorgestellte generische Architektur charakterisiert sich durch vier Softwareebenen, siehe [Bak16, S. 25] und A.1. Diese korrespondieren mit den im Abschnitt 2.1.2 erläuterten Testaktivitäten.

Die höchste Schicht, die Testgenerierungsebene (engl. *Test Generation Layer*), ist für die manuelle oder automatisierte bzw. modellbasierte Erstellung der Testszenarien verantwortlich und bietet Verwaltungsfunktionalitäten an wie z.B. Zuordnung von Testfällen zu Anforderungen, Navigation, Anpassung von Testsequenzen und Dokumentation. Des Weiteren informieren Bakker u. a. sowie Baumgartner u. a., dass unter Einhaltung bestimmter Voraussetzungen, wie z.B. Definition von Modellierungs-, Konfigurations- bzw. Parametrierungs-, Überwachungsfunktionalitäten und Generierungsalgorithmen, eine modellgetriebene Testfallgenerierung stattfinden kann.

Die Testdefinitionsebene (engl. *Test Definition Layer*) beschäftigt sich mit der Konkretisierung der Testszenarien. Diese Ebene beinhaltet alle Informationen, die für die konkrete Definition bzw. Implementierung eines Testfalls oder einer *Test Suite* erforderlich sind, und bietet Funktionalitäten, durch welche diese zusammen mit Testdaten, Strukturen zur Definition des Testablaufs, Testgruppen und Testskripte, verwaltet

und bereitgestellt werden [Bau21, S. 82]. Laut dem Lehrplan des International Software Testing Qualifications Board, siehe [Bak16, S. 26], beinhaltet diese Architekturebene Testtools zur Spezifikation von Testfällen, Testdaten, Testprozeduren zur Konkretisierung des Testablaufs und der damit assoziierbaren Vor- bzw. Nachbedingungen, Skripte für die Testausführung u.a.

Laut Bakker u. a. übernimmt die Ausführungsebene (engl. *Test Execution Layer*) die Aufgaben der automatischen Ausführung der Testfälle, ihrer Protokollierung und der Dokumentation der Testergebnisse. Laut Literaturangaben beschäftigt sich diese Schicht mit der Durchführung von Pre- und Post-Testausführungsaktionen, Konfiguration und Parametrisierung der Testausführungswerkzeuge, Analyse von SUT-Reaktionen während der Testausführungsphasen u.a. Baumgartner u. a. geben an dass, durch das Vergleichen des erwarteten mit dem tatsächlichen Verhalten des Testobjekts die Ausführungsebene Testergebnisse produzieren muss, welche im einfachsten Fall durch logische Größen repräsentiert werden.

Während alle übergeordneten Ebenen einem bestimmten Abstraktionsniveau entsprechen, charakterisiert sich die Adaptierungsebene durch konkrete, technologiespezifische Implementierungen von Adaptern, die eine Kommunikation mit dem SUT ermöglichen. Diese Schicht ist also von den Schnittstellen des Testobjekts abhängig und dient der Stimulation und der Beobachtung des Testobjekts. Laut Bakker u. a. ist die Adaptierungsebene auch für die Simulation bzw. Emulation der Testobjektumgebung geeignet. Es lässt sich vermuten, dass während die Schnittstelle der Adaptierungsebene zur übergeordneten Schicht keine Anpassungen erfordert, ihr Interface zum Testobjekt aus Wartbarkeitsgründen das *Open-Closed Principle* einhalten muss.

Die von Korotkiy und Bender in [KB10] vorgeschlagene und allgemeingültige Architektur zur Testautomatisierung konzentriert sich auf die Funktionstests mechatronischer Produkte und umfasst zwei Hauptkomponenten. Der Testautomat dient als Verallgemeinerung der Generierungs- und der Definitionsebene aus der bereits beschriebenen Architektur und definiert die Architekturebenen Testmanagement sowie Testausführung und Auswertung. Das ist die Schicht auf dem höchsten Abstraktionsniveau und gilt deshalb als Schnittstelle zum Anwender. Die zweite Komponente - das Testbett - gilt als Mediator zwischen dem Testautomaten und der SUT und spezifiziert die Ebenen Testbettsteuerung, Testobjektstimulation und -beobachtung sowie Testobjektumgebung.

Die Testmanagementebene orientiert sich u.a. nach der Erstellung, Strukturierung und Parametrisierung von Testfällen, der Protokollierung der Ausführungsphasen und der Dokumentation der Testergebnisse. Die Testausführungs- und -auswertungsschicht sind für die Übergabe der Testfällen an dem Testbett verantwortlich. Laut Korotkiy und Bender werden hier die Ablaufsphasen des Testfalls anhand der Testfallbeschreibung identifiziert und der Testbettsteuerung übergeben. Der Grund dieser Identifikation bzw. der Zerlegung der Testfälle in Testschritte besteht in der sukzessiven Testdurchführung und Protokollierung. Die Reaktionen des Testobjekts werden nach jedem Testschritt mit dem erwarteten Verhalten verglichen. Hierfür spezifizieren Korotkiy und Bender als Testergebnisse *PASS*, *FAIL* und *INCONCLUSIVE*. Laut der Architekturbeschreibung ist die Testbettsteuerung sowohl für die Sicherung der korrekten Ausführung der Testschritte als auch für die Transformation der Testschritte in Stimuli verantwortlich [KB10]. Dadurch erfolgt eine Stimulation und Beobachtung der SUT und ihrer Umgebung. Die mittlere Ebene des Testbetts - Testobjektstimulation und -beobachtung - dient als Schnittstelle zwischen der Steuerungsebene und der Testobjektumgebung. Sie ist mit der Adaptierungsebene der bereits vorgestellten Architektur vergleichbar. Die Testumgebung hat die Aufgabe der Durchführung der spezifizierten Pre- und Post-Testausführungsaktionen und der Simulation der Produktumgebung.

Die zwei Architekturen sind konzeptionell miteinander vergleichbar. Während der Ansatz von Korotkiy und Bender zur Sicherstellung der korrekten Ausführung der einzelnen Testschritte sinnvoll und für eine detaillierte Protokollierung und Lokalisierung von Defekten geeignet ist, stellt es sich die Frage wie genau aus den Testschritten passende Stimulationssignale generiert und in welcher Form die Ergebnisse aus der Beobachtung der Reaktionen des Testobjekts dem übergeordneten Layer zur Verfügung gestellt werden. Konkrete Definitionen bzw. standardisierte Implementierungsmöglichkeiten der kritischen Architekturkomponenten, z.B. der Testausführungskomponente, auf Basis von Entwurfsmustern wären für die ressourceneffiziente Realisierung von Testumgebungen sinnvoll und lassen sich als Gegenstände weiterer Untersuchungen bezeichnen.

2.2.3 Berichte und Metriken

Wie bereits erwähnt wurde, ist die geeignete Protokollierung und Auswertung der Testergebnisse eine bedeutende Voraussetung für die Messung der Qualität eines Testobjekts. Witte bezeichnet diese Qualität als "*[..] Abwesenheit von Fehlern im Systemverhalten*", die durch die Testaktivitäten nachgewiesen werden kann [Wit18]. Hierfür ist die Nutzung passender Metriken erforderlich, anhand welcher konkrete Aussagen über die Übereinstimmung des Ist- mit dem Soll-Zustands formuliert werden können. Baumgartner u. a. kommentieren, dass jeder Vorgang, der auf Basis aussagekräftiger Regeln einen formal gültigen Zahlenwert für ein bestimmtes Attribut produziert, als Metrik gilt [Bau21, S. 203]. Im Rahmen dieses Abschnitts werden anforderungsbasierte und softwareorientierte Metriken im Kontext des Testobjekts und der TAS diskutiert.

Laut [Wit18, S. 65] definiert die Funktionsmessung nach Gilb bestimmte Kriterien, auf welcher Basis die allgemeingültige Softwarequalitätsmerkmale eines Testobjekts verifiziert werden können. Zu diesen Zählen die Funktionalitätsmessung, welche sich aus dem Verhältniss der erfüllten Anforderungen zu ihrer Gesamtzahl ergibt [Wit18, S. 66]. Ein weiteres Aspekt ist die Performanz. Die Performanzmessung ist von der maximalen Reaktionszeit des Testobjekts abhängig und umfasst Messung von Antwortzeiten bei konstantem und anforderungsgemäß beherrschbarem Systemlast [Grü13, S. 168]. Andere Kriterien wie z.B. Zuverlässigkeit, Wartbarkeit u.a. sind in [Wit18, S. 66] definiert. In [Wit19, S. 299] definiert der gleiche Autor unterschiedliche Arten von Metriken. Als Beispiele dienen die testfallorientierte Metriken, welche i.A. mit der Anzahl von Testläufen, bestandenen Tests und Testwiederholungen assoziiert werden, die fehlerbasierte Metriken (z.B. die Anzahl der Fehlermeldungen) sowie Metriken wie die Codeüberdeckung, welche auf das Testobjekt basieren.

Baumgartner u. a. teilen die TAS-Metriken in interne und externe Metriken auf. Während sich die internen Metriken auf die interne Funktionsweise und Eigenschaften der Testautomatisierungslösung beziehen, orientieren sich die externen Metriken nach dem Einfluss der Testlösung auf andere, externe Prozesse im gesamten Lebenszyklus [Bau21, S. 204]. Als Beispiele für externe Metriken dienen den Nutzwert der Testautomatisierung zusammen mit dem Aufwand für die Erstellung und Wartung automatisierter Tests, die Zeit der Ausführungsphasen, den Anforderungs- und Codeüberdeckungsgrad u.a. Als interne Metriken spezifizieren Baumgartner u. a. die Fehlerdichte des Automatisierungscodes und die Geschwindigkeit bzw. Effizienz der TAS-Komponenten. Eine konkrete Vorgehensweise bei der Berechnung des Nutzwertes u.a. als Metriken bezeichnete Merkmale ist in [Bau21] jedoch nicht angegeben. Der Grund dafür liegt darin, dass diese kontextspezifisch bzw. von anderen Faktoren wie Kosten und weitere Ressourcen abhängig sind.

Weiterhin liefern Baumgartner u. a. Informationen über die Protokollierung der Testergebnisse. Laut [Bau21, S. 211] können im Laufe der Protokollierung sowohl Informationen über die Funktionsweise der TAS als auch solche über die Korrektheit des Testobjekts dokumentiert werden. Als Ziel der Protokollierung der TAS gilt die Wartung des Testautomatisierungsprozesses und die Reliabilität der Testergebnisse. Ein Testbericht muss alle Informationen zu den ausgeführten Testfällen beinhalten, inkl. welche Daten bei der Ausführung zum Einsatz kamen und welche Schritte durchgeführt wurden. Laut Witte müssen die abgedeckten Anforderungen zusammen mit den Programmethoden ebenfalls protokolliert werden [Wit19]. Dies ist jedoch von dem Umfang der Testautomatisierungslösung abhängig und erfordert ein automatisiertes Mapping zwischen Anforderungen und Testszenarien. Als weitere zu protokollierende Informationen spezifizieren Baumgartner u. a. die Start- und Endzeit der durchgeführten Testfällen, den Status der Testdurchführung, die im Rahmen der Testphasen durchgeführten Aktionen u.a. [Bau21, S. 212–213]. Für weitere Informationen siehe [Bak16, S. 53–59].

2.2.4 Verifizierung

Wie bereits erwähnt wurde lässt sich die Realisierung einer Testautomatisierungslösung als ein softwaretechnisches Projekt bezeichnen. Es stellt sich die Frage wie diese verifiziert werden kann. Das International Software Testing Qualifications Board gibt unterschiedliche Strategien an, siehe [Bak16, S. 70–73]. Die Variante, welche für ein Softwareprojekt typisch wäre, ist die Testung der für die Konstruktion der konkreten

TAA verwendete Werkzeuge und Softwarekomponenten. Diese sind sowohl isoliert als auch in Abhängigkeit voneinander zu verifizieren. Im Kontext der generischen Testautomatisierungsarchitektur wäre eine Bottom-Up Testung zum Zweck der Fehlerlokalisierung sinnvoll. Eine weitere Möglichkeit entspricht der automatisierten Ausführung von manuell erstellten und verifizierten Testszenarien mit bekannten Ausgangskriterien. Wird eine Differenz zwischen dem Ausgang des manuell und des automatisch ausgeführten Testfalls festgestellt, so müssen die protokollierten Informationen zusammen mit den Pre- und Post-Testausführungsaktionen überprüft werden. Ein Vorgehen bei der Verifikationsstrategie ist außerdem die Überprüfung der Reproduzierbarkeit des korrekten Startvorgangs bzw. des Heruntervahrens der Test Automation Solution auf unterschiedliche Umgebungen. Eine weitere wichtige Rolle spielt die Erstellung von Vorbedingungen für die Testausführung. Dadurch kann es sichergestellt werden, dass alle notwendigen Verbindungen für die Durchführung der Tests eingehalten sind.

2.3 Domänenspezifische Testverfahren

2.3.1 Virtuelle Inbetriebnahme

Matysczok und Seewaldt definieren als Ziel der digitalen Fabrik die "*frühzeitige Parallelisierung und möglichst komplette digitale Bearbeitung von Produktentwicklung und Produktionsplanung bis hin zum virtuellen Anlauf und Betrieb*", siehe [MS19, S. 1] und vgl. mit [BGW18]. Als erster Schritt zum digitalen Fabrikbetrieb geben weiterhin Matysczok und Seewaldt die virtuelle Inbetriebnahme (VIBN) an. Laut Weyrich erfordert eine dynamische Produktionsumgebung die kontinuierliche Durchführung von Tests im operativen Betrieb mit Fokus auf die Kooperation des Testobjekts mit seinem Umfeld [Wey15]. In diesem Zusammenhang beschreiben Oppelt und Urbas die Idee der integrierten VIBN [OU14, S. 5]. Die virtuelle Inbetriebnahme etabliert sich als wichtiger Teil der Verifizierung von Komponenten und Systemen, deren Erprobung in einer realen Produktionsumgebung und bei nicht validiertem Systemverhalten für die aktuelle Infrastruktur und die umgebenden Objekten als kritisch betrachtet wird. Laut [Rie20] trägt diese zur Sicherung von System-Konfigurationen bei und dient als Anwendung des digitalen Zwillings. Des Weiteren weisen Riedel u. a. darauf hin, dass eines der Ziele bei der Entwicklung adaptiver Produktionsanlagen in der Integration der virtuellen Inbetriebnahme in den gesamten Lebenszyklus besteht [Rie20]. Dies gilt außerdem als Voraussetzung für die Umsetzung eines kontinuierlichen und fähigkeitsorientierten Engineering- und Betriebsprozesses, siehe [Bay22, S. 7].

Laut Barth u. a. definiert den GMA FA 6.11 als Virtuelle Inbetriebnahme (VIBN) "*das entwicklungsbegleitende Testen einzelner Komponenten und Teilfunktionen des AS mithilfe von auf die jeweilige Aufgabenstellung abgestimmten Simulationsmethoden und -modellen.*" [Bar15, S. 4]. Im Fokus liegt also die simulative Verifikation von Funktionalitäten und Komponenten eines Automatisierungssystems. Wichtig ist, dass das Testobjekt, unabhängig davon, ob dieses von einer Softwarekomponente, einem elektronischen System oder einem Modell instanziiert wird, mit einer virtuellen Umgebung in Verbindung steht. Darauf basierend wird sein Verhalten in Zusammenspiel mit anderen Anlagenkomponenten verifiziert. In diesem Zuge wird es zwischen drei unterschiedliche Basiskonfigurationen unterschieden - *Software-in-the-Loop* (SIL), *Hardware-in-the-Loop* (HIL) und *Model-in-the-Loop* (MIL) [OU14]. Die MIL-Konfiguration wird i.d.R. in den früheren Entwicklungsetappen eingesetzt, um prototypische Implementierungen zu validieren [Bar15]. Als Grundlage dienen Modellierungsformen wie z.B. Zustandsautomaten, welche das Verhalten des Testobjekts spezifizieren. Während bei der HiL-Methode ein elektronisches Steuergerät (ECU) als Testobjekt eingesetzt wird, dient als Testbasis der SiL-Konfiguration die Ausführung der Steuerungscode auf eine emulierte Steuerung (z.B. eine *SoftPLC*), welche die entsprechende Software ausführt [Kam20]. Beide Konfigurationen erfordern die Kommunikation des Testobjekts mit der virtuellen Umgebung über geeignete Schnittstellen und Kommunikationskanäle [Bar15]. Die eigentliche Verifikation des Testobjekts findet nach dem Eingabe-Verarbeitung-Ausgabe (EVA) Prinzip statt, indem das Testobjekt auf die Signale der virtuellen Umgebung reagiert und seine Reaktion simulativ ausgewertet wird.

In Zusammenhang mit der VIBN wird oft der Begriff digitaler Zwilling (engl. *Digital Twin*) verwendet, weil dieser als eine Kombination aus Simulationsmodellen und digitalen Artefakten bzw. Messdaten,

auch digitale Schatten genannt, die den aktuellen Systemzustand repräsentieren, betrachtet wird [Ros20, S. 5]. Als potenzieller Entwicklungsschritt des digitalen Zwillings geben Rosen u. a. die automatische Ableitung von ausführbaren Simulationen an. Durch dieses Zukunftsbild kann eine virtuelle Verifikation des realen Zwillings im Rahmen der Entwicklungsetappen oder betriebsbegleitend von einer multifunktionalen Plattform, also eine um PC-Funktionalitäten erweiterte SPS, durchgeführt werden [Ros20, S. 5].

Die Vorteile der VIBN sind vielfältig. Ihre Zuordnung zu den Aktivitäten des Systemlieferanten führt zu einem Zeit- und Qualitätsgewinn [Rie20, S. 7] [Kam20, S. 3]. Der Grund dafür liegt daran, dass die Modellierung und Erstellung eines Simulationsmodells weniger Zeit im Vergleich zum Zusammenbau einer realen Anlage in Anspruch nimmt, für Konzeptprüfungen geeignet ist und eine Validierung des Engineering-Prozesses zusammen mit einem Soll-/Ist-Abgleich ermöglicht. Des Weiteren bekommt der Betreiber ein Simulationsmodell bereitgestellt, durch welches eine Betrachtung der Funktionsweise der entsprechenden Komponente nach einer Adaptierung der Anlage an weitere Anwendungsfälle ermöglicht wird. Weitere Argumente, die für den Einsatz der VIBN sprechen, sind kürzere Markteinführungszeiten aufgrund ermöglichter Qualitätssicherung während des Engineering-Prozesses und erhöhte Reaktionsfähigkeit durch bessere Abstimmung zwischen Systemlieferanten und Anlagenbetreiber auf Basis der Simulationsressourcen [Rie20, S. 15]. Oppelt und Urbas informieren, dass auch wenn die Vorteile der VIBN vielversprechend sind, hat sich diese in der Industrie nicht standardisiert. Als Herausforderungen bei der Verifizierung von Anlagenkomponenten auf Basis der Methoden der VIBN gelten die manuelle Erstellung und die Pflege der Simulationsmodelle, ihre steigende Komplexität und die vorausgesetzten Kompetenzen [OU14].

Weiterhin beschreibt [Rie20] auch die prozessuale Integration der VIBN. Diese korrespondiert mit einem automatisierten Testprozess, der aus zwei Phasen besteht - *Pull* und *Push*. Im Rahmen der ersten Phase werden Daten aus dem Engineering-Prozess, wie z.B. Anforderungsspezifikation, SPS-Programme, Konfigurationsdaten u.a. der VIBN übergeben. Laut [Rie20, S. 9] erfoglt auf Basis dieser Informationen die Erstellung eines Simulationsmodells und eines Testaufbaus. Im Rahmen des Push-Vorgangs werden die Testergebnisse in Form von Reports und Datenaufzeichnungen entsprechend zurückgeliefert. In diesem Zuge verhält sich der Prozess der virtuellen Inbetriebnahme wie eine Testautomatisierungslösung. Die Fragen wie aus den Eingangsdaten das Simulationsmodell und den Testaufbau realisiert werden, wie es sichergestellt wird, dass diese dem Anwendungsfall entsprechen, wie ihre Qualifizierung erfolgt und wer die Testdurchführung koordiniert, bleiben jedoch in [Rie20] unbeantwortet. Eine weitere Frage besteht in der Spezifikation und der Persistenz der Testdaten.

Laut Oppelt und Urbas können diese als Teil wiederverwendbarer Simulationstemplates integriert werden, siehe [OU14, S. 5]. Glock u. a. stellen in [Glo18] eine modellbasierte Testmethode auf Basis des HIL-Verfahrens vor. Die Authoren definieren als Voraussetzungen für die Durchführung des Testprozesses die Existenz eines Anlagemodells, welches sowohl das Komponentennetzwerk als auch die Funktionalitäten und ihre Hierarchie zusammen mit den Schnittstellen und den Kommunikationskanälen beschreibt. Darauf basierend erfolgt die manuelle Erstellung abstrakter Testszenarien für jede Funktion. Jedes Testszenario beinhaltet eine Referenz der zu testenden Funktion, eine Beschreibung, die Anforderung, welche mit der Funktion assoziiert wird, die durchzuführende Test-Schritte und den erwarteten Zustand des Testobjekts [Glo18, S. 4]. Ein Konzept für die dynamische Generierung solcher Testszenarien wird ebenfalls vorgestellt. Im nächsten Schritt erfolgt eine Parametrisierung der Testumgebung und die Durchführung der Tests, siehe [Glo18].

Diverse Faktoren und Kriterien, die direkte Auswirkungen auf den Nutzen und den Aufwand von VIBN-Projekten haben, sind in [Kam20] identifiziert worden. Ein wichtiger Schritt für die Umsetzung einer VIBN-Lösung entspricht der Auswahl des Simulationswerkzeugs. Riedel u. a. definieren hierfür unterschiedliche Kriterien, die diesen Prozess unterstützen. Als essenzielle Kriterien, welche ein Simulationswerkzeug zum Zweck seiner Einführung im Unternehmen gerecht werden muss, dienen die Erfüllung von Schnittstellenstandards zur Kommunikation mit Testobjekten und anderen Werkzeugen, die korrekte Abbildung zeitkritischer Komponenten bzw. die Erfüllung bestimmter Zeitanforderungen u.a. [Rie20, S. 26]. Eine wichtige Basis für den interoperablen Austausch und die Ausführung von Simulationsdateien wird durch das *Functional Mock-up Interface*[1] angeboten.

[1] `https://fmi-standard.org/`

2.3.2 Modellbasiertes Testen

Laut [Lig09] dienen Modelle wie z.B. Zustandsautomaten und Kontrollflussgraphen als Instrumentierung für viele struktur- und funktionsorientierte Testtechniken. Des Weiteren informiert Liggesmeyer, dass das modellbasierte Testen (MBT) zwei Ziele verfolgt. Während das erste Ziel in der Erstellung von Testfällen auf Modellbasis besteht, entspricht dem Zweiten die Auswertung von Testläufen mithilfe von Modellen [Lig09, S. 216]. In [Lig09] werden zwei Umsetzungsansätze für MBT vorgestellt. Der erste Ansatz umfasst die Erstellung eines Systemmodells aus formulierten Anforderungen. Im Rahmen der zweiten Variante wird sowohl ein System- als auch ein Testmodell realisiert. Laut Liggesmeyer liegt der Fokus des Testmodells in der Überprüfung, ob die Systemanforderungen erfüllt sind. In [Mal10] kommentieren Malik u. a. die Unterschiede zwischen den beiden Modellarten. Während das Systemmodell das Verhalten des Testobjekts beschreibt und als Verhaltensmodell bzw. Strukturmodell bezeichnet wird, liegt der Fokus des Testmodells auf die Funktionsweise des Testobjekts aus externer Sicht [Mal10].

Laut Utting und Legeard assoziiert man mit dem MBT die Generierung von Eingangsdaten aus domänenspezifischen und Testszenarien aus Umgebungs- und Verhaltensmodellen [UL07]. Des Weiteren informieren Utting und Legeard, dass ein weiteres Aspekt der modellbasierten Testung in der Generierung von Testskripten aus abstrakten Testszenarien besteht. Während die Generierung von Eingangsdaten und Testszenarien aus Umgebungmodellen laut den Autoren keine Erwartungen an das Testobjekt, also keine Ausgangsdaten zum Zweck der Verifikation produziert, führt die Erzeugung von Testszenarien aus Verhaltensmodellen zu einer Spezifikation von Soll-Werten, welche als Basis für einen objektiven Nachweis gelten. Die Erzeugung von Testskripten aus abstrakten Testfällen ist mit dem Prozess der Ableitung ausführbarer Aktionen aus Modellen (z.B. Sequenzdiagrammen) assoziierbar [UL07]. Utting und Legeard bezeichnen das MBT als "*the automation of the design of black-box tests*" und beschreiben als Teil des modellbasierten Testprozesses unterschiedliche Aktivitäten. Die erste Aktivität besteht in der Realisierung von Modellen, welche bestimmte Aspekte des zu testenden Systems exakt beschreiben. Durch den Einsatz der entsprechenden Softwarewerkzeuge folgt eine Ableitung von abstrakten Testszenarien. Im nächsten Schritt findet eine Konkretisierung der abstrakten Testszenarien statt. Diese Transformation produziert eine ausführbare Datei (*Test Suite*).

In [Ham11] wird ein Ansatz zur automatisierten Testfallgenerierung vorgestellt. Im Fokus des Beitrags liegen Automatisierungsanwendungen, die durch UML-Zustandsdiagramme beschrieben werden. Die Autoren stellen drei Ansätze für die Bereitstellung von Testszenarien vor. Während im Fokus des ersten Ansatzes die Spezifikation von Testfällen durch die manuelle Definition von Vorbedingungen, Aktionen, Postkonditionen und erwarteten Ergebnissen liegt, ist vom besonderen Interesse der dritte und modellbasierte Ansatz. Als Eingangsinformation für den Generierungsprozess gelten UML-Spezifikationen des Testobjekts. Das Ergebnis des Generierungsvorgangs entspricht einem ausführbaren Netzwerk aus Funktionsblöcken. Die durch das Zustandsdiagramm definierten Testszenarien werden auf Basis einer Modell-zu-Modell-Transformation als Instanz des Metamodells eines Klassendiagramms gespeichert, das ein *Test Suite* repräsentiert. Als Werkzeug für die Definition der Transformationsregeln verwenden die Autoren eine Erweiterung der Programmiersprache *Java*. Die Durchführung einer weiteren Modell-zu-Modell-Transformation generiert aus dem Klassendiagramm ein weiteres Modell, das dem IEC 61499 bzw. IEC 61131-3 XML-Schema entspricht. Für mehrere Informationen siehe [Ham11]. Weitere Beiträge, welche die Transformation von UML-Modellen in Funktionsblöcken adressieren und die MBT-Thematik berücksichtigen befinden sich unter [PF06] und [HF06]. In [Kre15] wird ein Werkzeug für die Durchführung modellbasierter Tests vorgestellt.

2.4 Industrie 4.0

Die vierte industrielle Revolution kombiniert die sog. operative Technologie, die zur Steuerung von Systemen und Komponenten eingesetzt wird, mit der Informationstechnologie, bei welcher den elektronischen Austausch von Daten und die Vernetzung von Komponenten und Systemen im Fokus liegen [DW16]. Daraus entstehen Technologiekonzepte wie Cyber Physical Systems (CPS) und Cyber Physical Production Systems (CPPS), welche durch ihre Sensoren, Aktoren und softwaretechnische Intelligenz die digitale und die analoge Welt als Teil eines gemeinsamen Netzwerks (Industrial Internet of Things) fusionieren [Bou19]. Laut Kuhn, Schnicke und Ziesche und wie auch im ersten Kapitel erwähnt wurde, liegt der Fokus der Industrie 4.0 Vision v.a. auf der Wandelbarkeit der Produktionsanlagen und der Herstellungsphasen zum Zweck der ressourcen- und kosteneffizienten Produktion von Losgröße eins [KSZ19]. Während die Anpassung der herkömmlichen Produktionssysteme an neuen Produktanforderungen mit der Durchführung neuer Entwicklungszyklen und Programmieraktivitäten assoziiert wird, fachspezifische Kompetenzen erfordert und i.A. als ineffizient und aufwändig betrachtet werden kann, umfasst die Industrie 4.0 Vision Konzepte und Lösungen, durch welche diese Lücke überbrückbar ist [Sch21]. Als beispielhafte Lösung stellen in [Sch21] Schäfer u. a. das *BaSys 4* Middleware vor, welches in Kapitel 4.1.2 behandelt wird. Als Beispiel für ein Konzept zum Erreichen der Wandelbarkeit dient die fähigkeitsbasierte Produktion, siehe Abschnitt 2.4.3. Eine bedeutende Voraussetzung für die Realisierung wandelbarer Anlagen ist die digitale Transformation der Komponenten und Systeme. Diese wird zusammen mit anderen Aspekten durch das RAMI 4.0 Modell beschrieben und basiert auf das Konzept der Verwaltungsschale.

2.4.1 Verwaltungsschalen

Laut [Sch21] definiert das RAMI 4.0 Modell die VWS als die digitale Basis für die Interoperabilität der Industrie 4.0 Systeme und ihre Komponenten. In einem Tech-Podcast von Bosch Rexroth, siehe [Zie22], definiert Ziesche die Verwaltungsschale als Selbstbeschreibung einer physikalischen oder logischen, individuell verwaltbaren und für die Produktion wertvollen Entität, also eines Assets, die als standardisierte bzw. herstellerunabhängige und zentrale Auskunftsstelle (*Single Point of Information*) jeder Industrie 4.0 konformen Komponente gilt. In [Bad20] ist die VWS als standardisierte digitale Repräsentation eines Assets definiert, welche digitale Modelle von unterschiedlichen und i.d.R. auch interdisziplinären Aspekten beinhaltet und technische Funktionalität bereitstellt.

In diesem Sinne ist die VWS mit einem Meta-Modell assoziierbar, durch welches die Aspekte und Charakteristiken eines Assets beschrieben werden können. Eine Verwaltungsschale dient also als eine konkrete Instanz dieses Metamodells bezogen auf ein bestimmtes Asset und setzt sich laut [4018] aus *Header* und *Body* zusammen. Während der *Header* Identifikationsdaten beinhaltet, administriert das *Body* individuelle Submodelle, die bestimmte Aspekte des Assets auf Basis von hierarchisch organisierten Submodell-Elementen beschreiben. Eine allgemeine Beschreibung der Struktur einer VWS ist in Abb. 2.1 exemplarisch dargestellt. Aus der Kompositionsbeziehung geht hervor, dass jedes Submodell unterschiedliche Eigentschaften (engl. *Properties*) bereitstellen kann, wobei als Beispiele Produktmerkmale, Prozessvariablen, Fähigkeiten, Operationen, Entitäten und andere Elemente dienen [HDM21]. Durch die Aggregationsrelation wird ausgedrückt, dass ein Teilmodell unabhängig von einer VWS existieren kann. Eine VWS stellt also sowohl statische als auch ausführbare Elemente zur Verfügung. Während in [Bad20] eine Operation als ausführbare Realisierung einer Funktion definiert wird, sind die Fähigkeiten bis dato ein Gegenstand der Forschung. Eine detaillierte Beschreibung des Metamodells liefern Bader u. a., siehe [Bad20].

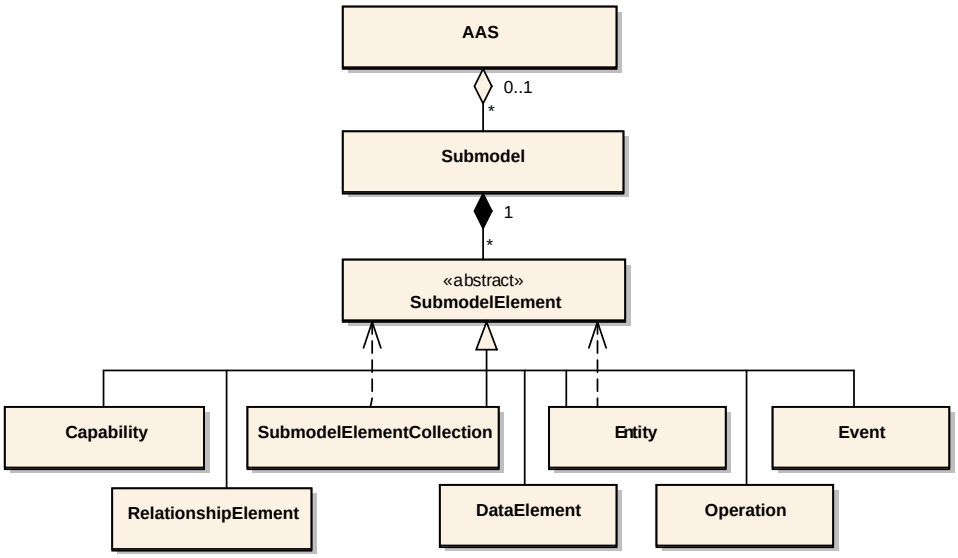


Abbildung 2.1: Allgemeine Struktur der Verwaltungsschale [HDM21] [Bad20]

Laut [4018] können die Informationen der VWS von unterschiedlichen Akteuren aktualisiert werden. Als Beispiele dienen Systemdesigner, Asset-Benutzer, Applikationen, Prozesse und das Asset selbst. Des Weiteren existieren bereits standardisierte Submodelle, die bei der Konzeption einer Industrie 4.0 Produktionsanlage im Kontext der Ressourcen instanziierbar sind. Während das von der Industrial Digital Twin Association (IDTA) entwickelnden Submodell *Simulation* als Beispiel für ein in Zusammenhang mit der VIBN integrierbares Template dient, existieren auch weitere Teilmodell-Templates, die für ein breites Spektrum von Anwengungsszenarien anwendbar sind. Beispiele für Standards und Spezifikationen, die Teilmodelle bereitstellen, sind *eCl@ss*[2] und IEC61987.

Laut [Fou22] existieren drei Typen von Verwaltungsschalen. Die Typ 1 VWS werden als passive Verwaltungsschalen bezeichnet und beinhalten statische Informationen. Die Repräsentanten dieses Typs werden in Form von Dateien verteilt, wobei als geeignete Formate für ihre Erstellung *JSON* und *XML* gelten. Die Typ 2 VWS werden laut [Fou22] als Laufzeit-Instanzen erstellt und auf Servern gehostet. Dadurch lassen sich die reaktiven Vertreter als Teilnehmer von Kommunikationsinfrastrukturen einsetzen. Des Weiteren können diese Daten Sammeln, Information bereitstellen, mit anderen Komponenten interagieren und unterschiedliche Dienste anbieten. Die Typ 3 Verwaltungsschalen werden als proaktive digitale Vertreter bezeichnet und ergänzen die Typ 2 VWS. Die proaktiven Komponenten charakterisieren sich durch die Implementierung eines spezifischen Kommunikationsverhaltens und sind in der Lage VWS-Netzwerke auf Basis der in VDI/VDE 2193 beschriebenen Sprache zu orchestrieren.

2.4.2 Fähigkeiten

[Bay22] gilt als grundlegende Referenz zum Thema Fähigkeiten im Kontext der wandelbaren Produktionsanlagen. Laut dem Dokument gelten Fähigkeiten als abstrakte und generische, also vom Asset unabhängige, Beschreibungen potenziell erzielbarer und domänenspezifischer Wirkungen. Die technische Umsetzung einer Wirkung ist im Kontext der Vision durch die sog. Skills realisierbar. In diesem Zusammenhang dient ein *Skill* als konkrete Asset-spezifische Implementierung des Potenzials einer Komponente, einen bestimmten Effekt in der physikalischen oder virtuellen Welt zu erzielen, siehe [Bay22] und [PSB13]. Eine weitere Beschreibung der Begrifflichkeiten ist zusammen mit dem Metamodell einer Fähigkeitsontologie in [Wes20] zu finden. Weser u. a. beschreiben als *Skill* "*[..] the parametrizable and executable functionality*

[2] `https://eclass.eu/en`

of hardware or software components“ [Wes20]. Im Beitrag werden Fähigkeiten als Metabeschreibungen von *Skills* angesehen.

Bayha u. a. unterscheiden zwischen atomaren und zusammengesetzten sowie prozess-, produkt- und ressourcenspezifischen bzw. -unabhängigen Fähigkeiten, siehe [Bay22, S. 11]. Von besonderer Bedeutung sind die atomaren und zusammengesetzten Capabilities. Während die atomaren Fähigkeiten durch eine niedrigere Abstraktionsebene charakterisiert werden, lassen sich diese auf Basis unterschiedlicher technischen Implementierungen realisieren. Dementsprechend können mehrere *Skills* einer atomaren Fähigkeit zugeordnet werden. Das Mapping zwischen einer Fähigkeit und einer Fertigkeit (engl. *Skill*) findet durch das sog. *RelationshipElement* statt, siehe Abb. 2.1 und A.2.

In [Bay22, S. 15] werden fünf Varianten für die Realisierung von Fertigkeiten definiert - Zustands- und Trigger-Variablen, Operationen, Funktionsblöcke und semantische Protokolle bzw. REST-Dienste. Es wird informiert, dass die Realisierung eines *Skill* durch die Implementierung von Funktionsblöcken als potenziell standardisierbare Option gilt. Es stellt sich die Frage, wie dei Fähigkeiten im Kontext der VWS umgesetzt werden. Anhand der Informationen in [Bay22] wurde das durch Abb. A.2 dargestellte Klassendiagramm erstellt. Aus diesem geht hervor, dass die Umsetzung von Skills, welche auf Steuerungsebene durch Funktionsblöcke implementiert werden, im Kontext der VWS auf Basis von Submodellen stattfindet.

Die Abstraktion von *Skills* durch *Fähigkeiten* bildet die Basis für das im nächsten Unterabschnitt beschriebene PPR-Modell (Process, Product, Ressource).

2.4.3 Skill-Based Production

In [Bay22] beschreiben Bayha u. a. als Hauptziel des PPR-Modells die dynamische Buchung und Anwendung von Ressourcen, deren Fähigkeitsprofil den Anforderungen des durchzuführenden Produktionsprozesses entspricht. Wird der Produktionsprozess durch eine Verwaltungsschale repräsentiert, so gilt also als Voraussetzung für die Skill-basierte Produktion die Spezifikation der Fähigkeiten der notwendigen Ressourcen mithilfe des VWS-Metamodells. Laut dem Beitrag sind die Merkmale der drei Hauptbestandteile des PPR-Konzeptes entscheidend, ob die gewünschte Wirkung tatsächlich erzielt werden kann.

Der Engineering-Prozess setzt sich aus drei Phasen zusammen. Im Rahmen der ersten Phase findet das sog. *Capability Check* statt. Durch dieses werden die vom Prozess angeforderten mit den von den Ressourcen angebotenen Fähigkeiten abgeglichen [HDM21]. Laut Bayha u. a. eignet sich für diesen Abgleich die Beschreibung von Fähigkeiten durch den Einsatz von Ontologien, siehe [Wes20]. Ein potenzielles Werkzeug in diesem Zusammenhang ist *Protégé*[3].

Die Existenz und die Verfügbarkeit der Ressourcen, welche die notwendigen Fähigkeiten durch die Implementierung der entsprechenden Skills bereitstellen, dienen als Voraussetzung für die zweite Phase - das *Feasibility Check*. Die Aufgabe dieses Vorgangs besteht in der Prüfung, dass alle Bedingungen für die Durchführung der Fähigkeiten erfüllt sind. In Bayha u. a. informieren die Autoren, dass diese sowohl von dem Produkt, dem Prozess und den Ressourcen, als auch vom Kontext definiert werden können. Aus diesem Grund muss das Engineering-Werkzeug für die Durchführung der Prüfung eine Schnittstelle zum Bereitstellen von Umgebungsmodellen anbieten. Im Rahmen des *Feasibility Check* erfolgt eine Parametrierung der identifizierten Ressourcen. Die Vor- bzw. Nachbedingungen können zusammen mit den Invarianten simulativ, auf Basis einer VIBN, überprüft werden.

Die dritte Phase wird als *Skill Execution* bezeichnet und umfasst die Parametrierung der Ressourcen und das Aufrufen der für die Erstellung des Produktes notwendigen Fähigkeiten. Auch im Rahmen dieser Phase findet die Überprüfung der erwarteten Konditionen statt.

Die Ideen, welche als Grundlage des PPR-Modells und des fähigkeitsorientierten Engineeringsprozesses dienen, werden durch Abb. A.3 illustriert. Für weitere Informationen siehe [PSB13], [Bay22], [Per19], [HDM21] und [Wes20].

[3] `https://protege.stanford.edu/`

Kapitel 3

Verwandte Arbeiten

Wie bereits im Rahmen des ersten Kapitels erwähnt wurde, gibt Weyrich u. a. an, dass das RAMI 4.0 Modell durch seine Schichtenarchitektur zu einer Komplexitätsreduktion dient und keine standardisierte Verifikationsmethoden beschreibt, siehe [Wey18]. Arbeiten wie [Sch22b], [Glo18], [Wey15], [Wey18], [KB10] und [Löc21] dienen als Argument für die Notwendigkeit der Testautomatisierung im Bereich der industriellen Produktion.

In [MRK16] illustrieren Magnus, Russ und Krause Engineering-Prozesse für die automatisierte Testdurchführung und modellbasierte Testgenerierung auf Basis der Formalisierung anforderungsbasierter Systembeschreibungen. Der Schwerpunkt des Beitrags liegt auf der anforderungsorientierten Spezifikationsmodellgenerierung. Von besonderem Interesse ist der vorgestellte Formalisierungsansatz. Hierfür schlagen die Autoren die Verwendung von Sequenzdiagrammen vor, da diese für eine detaillierte Beschreibung funktionaler Anforderungen geeignet sind. Darauffolgend werden diese mit einer internen und textbasierten Anforderungsbeschreibungssprache des Instituts für Automatisierung und Kommunikation (IFAK) beschrieben und in einer späteren Phase mittels geeigneter Algorithmen zu einem Spezifikationsmodell synthetisiert, für mehrere Informationen siehe [MRK16]. Obwohl sich die Transformation der einzelnen Anforderungen in Sequenzdiagrammen als eine schwer automatisierbare Aktivität bezeichnen lässt, bietet die vorgestellte Methode eine gute Basis für die Transformation textbasierter Anforderungen in einer maschineninterpretierbaren Form an. Andere Möglichkeiten zur Formalisierung von Anforderungen bieten die sog. *Gherkin*[1] Editoren, welche im Rahmen der verhaltensgetriebenen Softwareentwicklungsprozessen und zur Spezifikation von Testszenarien im Format *Given-When-Then* eingesetzt werden.

In [Glo18] bezeichnen Glock u. a. die dezentrale Distribution von Aufgaben und Funktionalitäten als ein bedeutender Aspekt der Industrie 4.0 Vision und beschreiben modellbasierte Testprozesse, Validierungsmethoden, die zur Sicherstellung der korrekten Funktionalität vernetzter Komponenten beitragen. Des Weiteren prognostizieren die Autoren eine Veränderung der Verantwortlichkeiten der Komponentenhersteller, die sich in der Trennung der Applikationen von den physikalischen Komponenten äußert und zur Entwicklung neuer Geschäftsmodelle führen würde. In diesem Sinne wird die Möglichkeit der Fähigkeitsverteilung eine lose Kopplung zwischen Soft- und Hardware ermöglichen. Die Entwicklung der Applikationen und Hardwarekomponenten kann dann unabhängig und von unterschiedlichen Anbietern durchgeführt werden. Die Entscheidung, welche Funktionalität von welcher Komponente unterstützt werden muss, wäre in diesem Szenario eine Aufgabe des Integrators. Dieser übernimmt weiterhin die Verantwortung für das Sicherstellen der korrekten Funktionsweise der Anlage und ihrer Komponenten, für mehrere Informationen siehe [Rie20, S. 6]. Das vorgestellte Testverfahren basiert auf die HIL-Methode und erfordert ein Modell der Produktionsanlage, welches Informationen über das gesamte Netzwerk aus Funktionsblöcken und physikalischen Komponenten liefert. Auf Basis der Anforderungsspezifikation werden abstrakte Testszenarien formuliert und jeder zu testenden Funktion zugeordnet. Jedes abstrakte Testszenario beinhaltet eine Beschreibung, die Bezeichnung des Testobjekts, die korrespondierende Anforderung sowie die spezifizierte Testschritte und die erwartete Ausgangswerte. Mittels des vom Modell beschriebenen Funktionsnetzwerks erfolgt die Identifikation des Testobjekts. Darauf basierend werden alle anderen Funktionsblöcke identifiziert, die direkt oder indirekt dem Testobjekt Daten bereitstellen. Im nächsten Schritt erfolgt die Identifizierung der Komponenten, die durch diese Funktionsblöcke repräsentiert werden, und die Erfassung der entsprechenden Parameterwerte. Auf diese Informationsbasis

[1] `https://www.gherkineditor.com/`

erfolgt die Generierung eines Testfalls. Darauffolgend findet die Vorbereitung der Testumgebung statt. Als Grundlage hierfür dient ebenfalls das Modell der Anlage. Am Ende findet die Verifikation nach der klassischen HIL-Methode statt. Obwohl die VWS im Rahmen des von Glock u. a. beschriebenen Konzepts nicht betrachtet wird, lässt sich der vorgeschlagene Ansatz durch bestimmte Anpassungen auch mittels Verwaltungsschalen anwenden. Als Voraussetzung dafür gilt die konkrete Spezifikation der Metamodelle *Capability* und *Skill* zusammen mit einem beispielhaften Szenario, anhand welches diese weiter analysiert werden können. Eine Grundlage für die Migration des Verfahrens im Kontext der I4.0 Vision bietet die konkrete Spezifikation, wie Funktionsblöcke im Kontext der VWS implementierbar sind. Eine weitere wichtige Rolle spielt die Anwendung geeigneter Algorithmen zur Zerlegung und Traversierung zusammengesetzter Fähigkeiten, also von Funktions- bzw. Skill-Netzwerken, im Kontext der VWS sowie die klare Spezifikation von Testdaten im Rahmen der Anforderungsanalyse.

In [Bay22] werden Basiskonzepte für die Umsetzung von Fähigkeiten und Skills vorgestellt, wobei eine genaue Spezifikation der entsprechenden Metamodelle z. Zt. nicht verfügbar ist. Anhand dieser Konzepte schlagen Huang, Dhouib und Malenfant modellbasierte Techniken für die Realisierung fähigkeitsorientierter Anlagen basierend auf Verwaltungsschalen vor, siehe [HDM21]. Der vorgeschlagene Ansatz orientiert sich nach der kontinuierlichen Rekonfiguration von Produktionsressourcen und stellt eine Ergänzung des Modellierungswerkzeugs *Eclipse Papyrus*[2] dar. Die Autoren beschreiben die Modellierung von Verwaltungsschalen, Fähigkeiten, Skills, VWS-Netzwerken sowie Produktionsprozessen und erstellen eine Modellierungsumgebung, welche die Realisierung von Verwaltungsschalen ohne tiefe technische Kompetenzen ermöglicht und als Basiswerkzeug für die modellbasierte Umsetzung der Vision *Skill-Based Production* gilt. Obwohl die Durchführung des sog. Feasibility Check keinem Gegenstand des Beitrags entspricht, definieren die Autoren die Entwicklung eines Orchestrators für die Ausführung des gesamten Engineering-Prozesses als künftige Arbeit.

In [Per19] beschreiben Perzylo u. a. Methoden zur semantischen Beschreibung und Orchestrierung von Skills und Fähigkeiten unter Berücksichtigung der *BaSys 4.0* Initiative. Die Autoren betrachten die Skills aus unterschiedlichen Perspektiven, beschreiben bestimmte Charakteristiken eines Skills und diskutieren die Informationen, welche bei einer Skill-Definition betrachtet werden müssen. Des Weiteren stellen Perzylo u. a. eine Methode für die Beschreibung von Fertigkeiten auf Basis semantischer Notationen vor. Im Beitrag werden zusammengesetzte Fähigkeiten diskutiert und ein prototypischer Orchestrator vorgestellt, der für die Durchführung eines Capability Check geeignet ist und die Synchronisation sekundärer Skills übernimmt. Zum Zweck des Engineerings von komplexen zusammengesetzten Fähigkeiten und für den Abgleich der nachgefragten mit den verfügbaren Fähigkeiten wird eine Beschreibung von Skills benötigt. Aus diesem Grund ist von den Autoren auch eine domänenspezifische Sprache vorgestellt worden, durch welche die Interaktion der Fertigkeiten mit der Produktionsumgebung beschrieben wird. Abschließend wird im Beitrag die Notwendigkeit der Definition standardisierter Beschreibungsformen von Skills und Fähigkeiten hervorgehoben. Eine Spezifikation der entsprechenden Metamodelle würde die Grundlage für die Entwicklung einer fähigkeitsorientierten Testlösung bilden.

Gampig, Terzimehić und Dorofeev definieren Funktionsblöcke zur Interaktion der Steuerungskomponenten mit der I4.0 Middleware, siehe [GTD21]. Als mögliche Kommunikationspartner spezifizieren Gampig, Terzimehić und Dorofeev Verwaltungsschalen, Registry- und Discovery-Server sowie weitere I4.0 konforme Infrastrukturkomponenten. Im Fokus der Betrachtung liegt das Konzept vom *BaSys 4.0* Projekt. Die Autoren stellen Funktionsbausteine vor, durch welche die SPS als Server und Client im Kontext von Verwaltungsschalen und Submodellen agieren kann. Zusätzlich sind im Beitrag Funktionalitäten beschrieben, welche die Kommunikation mit einer Registry ermöglichen. Durch die Spezifikation eines Submodells zur Definition von Testszenarien und Testdaten innerhalb der VWS und mithilfe der vorgestellten Funktionsblöcke wäre die Initiierung eines Testvorgangs im Rahmen der Identifikationsphase der Komponente von der Steuerung selbst möglich. Mithilfe des Submodells *Simulation*[3] und unter Einhaltung der obengenannten Voraussetzungen könnte eine VIBN vom Testobjekt getriggert werden.

Löcklin u. a. stellen Lösungsansätze für das automatisierte Testen industrieller Anlagenkomponenten unter Berücksichtigung der agilen Produktion vor, siehe [Löc21]. Im Rahmen des VDI-Berichts werden Maßnah-

[2] `https://www.eclipse.org/papyrus/`
[3] `https://github.com/admin-shell-io/submodel-templates/tree/main/development/Simulation/1/0`

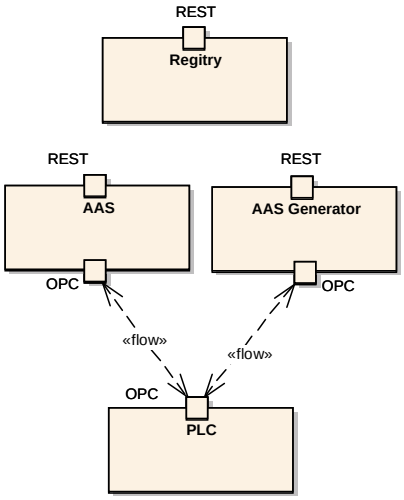


Abbildung 3.1: Generierungsumgebung nach [Sch21]

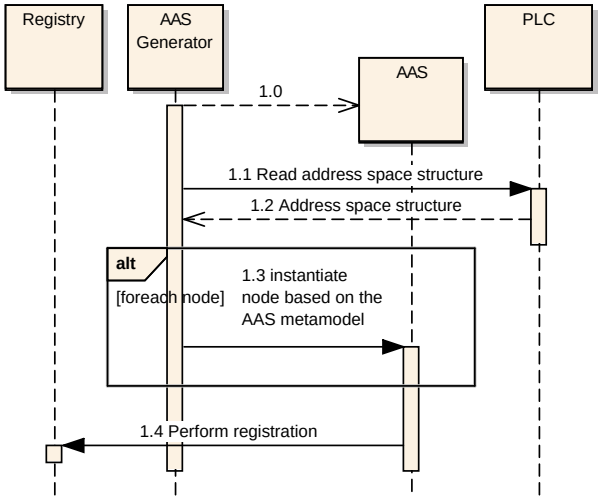


Abbildung 3.2: Ablauf nach [Sch21] (vereinfacht)

men diskutiert, die eine Bewältigung der immer komplexer werdenden Testaktivitäten ermöglichen. Es werden Ansätze zur Reduktion des Testaufwands und der Testdurchführung vorgeschlagen, die in gleicher oder ähnlicher Form auch in [Wey18] betrachtet werden und mit einer Modularisierung und Wiederverwendbarkeit von Hardware-, Softwarekomponenten und Funktionalitäten assoziierbar sind. Neben der Einbindung von Testexperten im Engineering Prozess beschreiben Löcklin u. a. den Einsatz digitaler Zwillinge als eine weitere Maßnahme zur Effizienzsteigerung des industriellen Testprozesses. Im Rahmen des Berichts wird die Verwaltungsschale aufgrund ihrer Rolle als standardisierte Auskunftsstelle als potenzielles Testwerkzeug angesehen und das modellbasierte Testen im Kontext der automatisierten Testfallgenerierung diskutiert. Anhand einer Demonstrationsanlage aus der Prozessindustrie beschreiben die Autoren die Synergieeffekte, welche unter Berücksichtigung aller vorgestellten Ansätze im Rahmen eines Testprozesses entstehen. Laut Löcklin u. a. bestehen die Vorteile der Verwendung von Verwaltungsschalen im Rahmen des Testprozesses in der Bereitstellung umfassender Informationen, z.B. Entwicklungs-Artefakte, auf Basis domänenspezifischer Submodelle sowie durch die Kommunikation mit mehreren externen Kommunikationspartnern. Zum Zweck des praktischen Nachweises haben die Autoren eine VWS für eine Komponente der Demonstrationsanlage mittels des *AASX Package Explorer*[4] entwickelt, die UML-Modelle beinhaltet. Durch die Methoden des modellbasierten Testens werden aus den (Zustands-) Modellen Testfälle generiert und in der VWS gespeichert. Hierfür verwenden Löcklin u. a. die Testwerkzeuge *MoMuT*[5] und *Eclipse Papyrus*. Nach der Testgenerierung erfolgt die Testausführung und Auswertung. Als mögliche Konzepte schlagen die Autoren HIL und SIL vor. Eine vollständige Automatisierung des vorgeschlagenen Testansatzes ist aufgrund der modellbasierten Testfallgenerierung und der damit assoziierbaren manuellen Modellerstellung als nicht realistisch zu betrachten. Der Beitrag stellt eine wesentliche technologische Antwort der steigenden Anforderungen an dem industriellen Testprozess dar und gilt als Impuls für die Konzeption weiterer Testmethoden auf Basis der VWS.

In [Sch21] stellen Schäfer u. a. eine Methode zur automatischen Generierung von Verwaltungsschalen vor, die auf das *BaSys 4.0*[6] Middleware basiert. Wie Glock u. a. in [Glo18] ziehen auch die Autoren von [Sch21] die Kapselung von Funktionalitäten, also die Modularisierung, und das Design auf Basis verifizierter Funktionstemplates als Ansatz für die digitale Transformation von Produktionanlagen in Betracht. Des Weiteren beschreiben Schäfer u. a. eine Sortieranlage zusammen mit der Gruppierung der Funktionsmodule auf die Steuerungsebene als Grundlage des Beitrags. Außerdem werden im Beitrag die Notwendigkeit und die Vorteile der automatischen Erstellung standardisierter digitaler Zwillinge hervorgehoben und eine minimale Umgebung für die Umsetzung einer entsprechenden Generierungslösung

[4] `https://github.com/admin-shell-io/aasx-package-explorer`
[5] `https://momut.org/`
[6] `https://wiki.eclipse.org/BaSyx`

vorgestellt, siehe Abb. 3.1. Die Autoren beschreiben einen Generierungsprozess, der aus passiven digitalen Vertretern aktive Verwaltungsschalen in Form von Docker-Containern generiert. Als Voraussetzung für den Generierungsprozess gilt die Implementierung eines OPC UA Servers seitens der Steuerungskomponente. Die Aufgabe der Generierungslösung besteht in dem Durchqueren der Adressraumstruktur des OPC UA Servers und ihre Abbildung in einer Typ 2 Verwaltungsschale mit REST-Schnittstelle, siehe 3.2. Somit gilt als einzige Voraussetzung für den Generierungsprozess die Verfügbarkeit eines OPC UA Servers mit der entsprechenden Adressraumstruktur. Während die Kommunikation mit der I4.0 Komponente für den Anwender über HTTP stattfindet, erfolgt die Synchronisation zwischen der Steuerung und der VWS über OPC UA. Für weitere Informationen siehe [Sch21].

Darauf aufbauend wird in [Sch22b] ein Testkonzept vorgestellt, welches eine zentrale Rolle für die praktische Ausarbeitung dieser Abschlussarbeit spielt und nachfolgend erläutert wird. Der in [Sch22b] vorgeschlagene Ansatz berücksichtigt das PPR-Modell und das Konzept der fähigkeitsorientierten Produktion. Im Rahmen des Artikels wird ein Engineering-Prozess als allgemeine Voraussetzung für eine Industrie 4.0 orientierte Testlösung beschrieben, siehe Abb. 3.3. Dieser gilt außerdem als eine allgemeine Vision für die dynamische Zuordnung und Verifizierung von Fähigkeiten in wandelbaren Produktionsanlagen und wird in gleicher oder ähnlicher Form von [Glo18], [MRK16] und [Wey18] adressiert. Der in [Sch22b] beschriebene Engineering-Prozess gilt als Ausgangspunkt für eine Testdurchführung und ermöglicht den systematischen Vergleich des Ist- mit dem Soll-Zustand eines Testobjekts. Des Weiteren wird dieser als Konzept für die Realisierung wandelbarer Produktionsanlagen beschrieben. Es wird zwischen drei Phasen unterschieden.

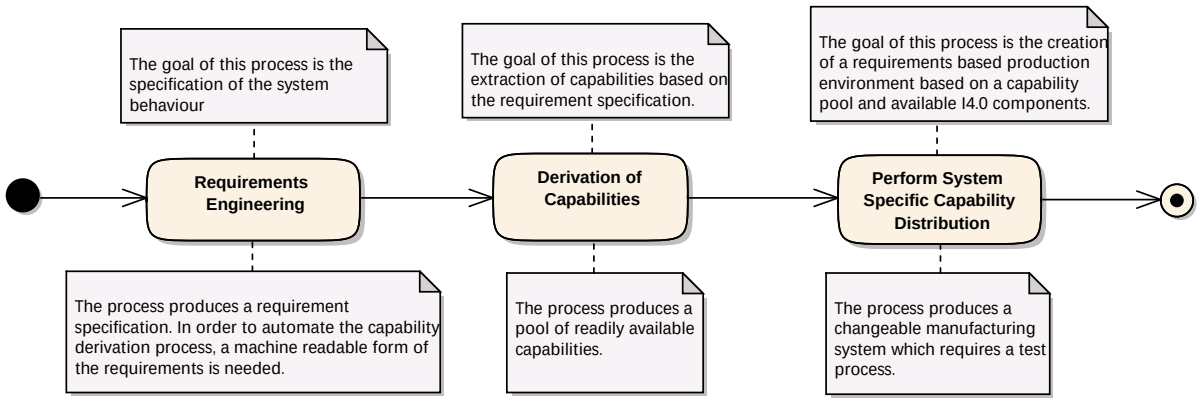


Abbildung 3.3: Engineering-Prozess als Voraussetzung für eine Testlösung nach [Sch22b]

Als ersten Schritt definieren die Autoren die Anforderungsspezifikation. Durch diese wird das Verhalten der Komponenten in Korrelation mit dem konkreten Anwendungsszenario beschrieben. Im Rahmen dieser Phase werden Vor- und Nachbedingungen sowie Invarianten, Komponentenmerkmale und ggf. Testdaten spezifiziert, die alle mit einer Aktion bzw. Fähigkeit assoziierbaren Voraussetzungen und Zuständen darstellen. Als Ergebnis der Anforderungsspezifikation dient laut [Sch22b] eine konkrete Beschreibung des erwarteten Verhaltens aller zu integrierenden Ressourcen. Diese Beschreibung gilt sowohl als Artefakt für die Durchführung eines Soll-/Ist-Abgleichs als auch als wichtige Voraussetzung für den nächsten Schritt des Engineering-Prozesses. Im Rahmen der zweiten Phase erfolgt die Ableitung der Fähigkeiten aus der Anforderungsspezifikation mithilfe eines Engineering-Werkzeugs. Die Metamodelle für Fähigkeiten und Skills entsprechen zusammen mit einer geeigneten Semantik für die Formulierung der Anforderungen einer zentralen Voraussetzung für die Implementierung des Tools. Des Weiteren können bei der Anforderungsspezifikation auch konkrete bzw. implementierte Funktionalitäten aus einer bereits existierenden Funktionsbibliothek betrachtet, ggf. implementiert und jeder Anforderung zugeordnet werden. Ein standardisiertes Pool aus Skills würde laut den Autoren zu einer Ressourceneffizienz führen, weil dieses in Zusammenhang mit dem Engineering-Tool eine direkte Ableitung und Distribution von Fähigkeiten ohne zusätzlichen Programmieraufwand garantiert. Eine Voraussetzung für den Phasenübergang ist also

die Aufbereitung der Anforderungen in einer maschineninterpretierbaren Form und die Existenz einer Software, die aus formalisierten Anforderungen verteilbare Fähigkeiten ableitet. Zum Zweck der Wiederverwendbarkeit sehen die Autoren die Persistenz der abgeleiteten Fähigkeiten in einer Bibliothek vor. Der letzte Schritt des Engineering-Prozesses beschäftigt sich mit der Verteilung der Fähigkeiten auf die Industrie 4.0 Komponenten. Wie die Realisierung einer I4.0 Komponente stattfinden kann, ist im Dokument anhand drei Szenarien vorgestellt, wobei eine zukunftssichere Lösung vom VWS-Generator angeboten wird und qualitative Testlösungen erfordert. Eine Voraussetzung für die Funktionsverteilung ist, dass die Charakteristiken der zu distribuierenden Fähigkeit mit den Merkmalen der entsprechenden Ressource übereinstimmen. Die Ausführung der Fähigkeit erfordert das Erfüllen aller spezifizierten Vorbedingungen. Laut Schäfer u. a. verlangt die fehlerfreie Interaktion der Ressourcen den Einsatz qualitativer Testlösungen [Sch22b].

Die in [Sch22b] vorgeschlagene Testumgebung besteht aus einem Test-Koordinator, mehreren aus Steuerungsmodulen und Verwaltungsschalen bestehenden Industrie 4.0 Komponenten und einer virtuellen Simulationsumgebung. Die entsprechende Zuordnung von Schnittstellen soll laut den Autoren eine dynamische Austauschbarkeit von Testobjekten bzw. ein ständiges Wechseln von Kommunikationspartnern ermöglichen. Der Testkoordinator dient im Rahmen der Testumgebung als eine Testautomatisierungslösung, die Testergebnisse protokolliert, und fungiert auf Basis bereitgestellter Testszenarien. Im Beitrag werden mehrere Testschritte vorgeschlagen. Im Rahmen des Test-Pipieline werden Strukturtests durchgeführt, ggf. die Zustandsmaschine der Steuerungskomponente verifiziert und eine virtuelle Inbetriebnahme als Methode zur Testung der Funktionalitäten vorgeschlagen. Für weitere Informationen siehe [Sch22b].

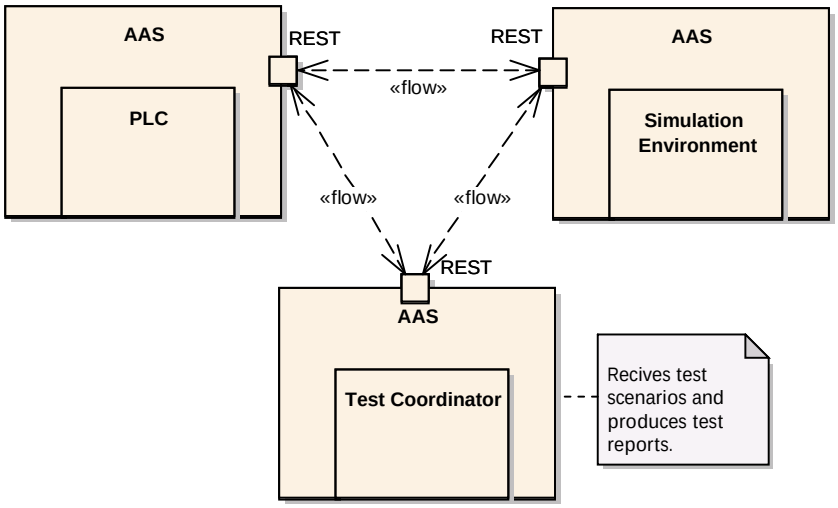


Abbildung 3.4: Testlösungsarchitektur nach [Sch22b]

In [Sch22c] diskutieren Schäfer u. a. Möglichkeiten zur Überführung von Bestandsanlagen mithilfe speicherprogrammierbarer Steuerungen. Die Autoren berücksichtigen sowohl die Verwendung eines VWS-Generators als auch die Nutzung von *OMAC PackML*[7] als Werkzeug zur Synchronisation der Prozessführung. Darauf basierend stellen Schäfer u. a. einen Ansatz zur Bereitstellung von Zustandsinformationen auf der Abstraktionsebene der Verwaltungsschalen dar. Als Voraussetzung dafür gilt die Implementierung bestimmter Strukturen in der globalen Variablentabelle des Steuerungsprogramms. Dadurch können die entsprechenden Zustandsinformationen als Knoten innerhalb des Adressraums eines OPC UA Servers gespeichert und im Rahmen des Generierungsvorgangs innerhalb der VWS entsprechend instanziiert werden. Da der Fokus von *PackML* auf sog. Betriebseinheiten (Units) liegt, wird im Beitrag die Datenstruktur `Unit` vorgeschlagen, die alle für die Abstraktion notwendigen Informationen beinhaltet. Während durch den Datentyp `aUnitInfo` Informationen zur Beschreibung des Prozesszustands bereitgestellt werden, erfolgt durch `aUnitController` die Steuerung der entsprechenden Einheit. Im Rahmen des Beitrags wird

[7] `https://www.omac.org/packml`

außerdem eine Datenstruktur zur Speicherung von ergänzenden Prozessinformationen vorgestellt. Die Autoren sehen für die Instanziierung der Daten im Kontext der VWS die Verwendung von Submodellen und Submodell-Elementen vor. Auf Basis der spezifizierten Klassen werden generische VWS-Strukturen vorgestellt, welche die Struktur des Ergebnisses jedes Generierungsvorgangs beschreiben und somit als Templates für weitere Lösungen einsetzbar sind.

Kapitel 4

Konzeption

Im Rahmen dieses Kapitels werden die Muss-, Wunsch-, Kann-, sowie die Abgrenzungskriterien für eine VWS-orientierte Testautomatisierungslösung dargelegt. Anhand der Anforderungen erfolgt die Beschreibung der grundlegenden Komponenten, Anwendungsszenarien und Softwarefunktionalitäten. Es wird die Softwarearchitektur einer Testlösung diskutiert, auf Basis geeigneter Softwarekomponenten entworfen und dokumentiert.

4.1 Einführung

4.1.1 Ziel des Systems

Wie bereits im Kapitel 2.4.1 erläutert wurde, dient die Verwaltungsschale durch die Implementierung eines Metamodells als Abstraktionsmittel bzw. "*Single Point of Information*" eines oder mehrerer Assets [Zie22]. Eine Verwaltungsschale lässt sich also als allgemeingültige Schnittstelle jeder Industrie 4.0 konformen Anlagenkomponente bezeichnen und stellt somit die digitale Basis für die Standardisierung eines vendorübergreifenden Testprozesses dar.

Wie in [Sch22a] beschrieben, verfolgt den Testprozess folgende Ziele:

- Verifikation und Validierung von Anlagenkomponenten nach veränderten Anforderungen
- Qualitätssicherung in den operativen Phasen
- Zertifizierung von Verwaltungsschalen, Submodellen und Operationen zum Zweck der Wiederverwendbarkeit
- Evaluierung von Komponenten anhand domänenspezifischer Anwendungsszenarien
- Aufzeichnung von Komponentenverhalten zum Zweck der Validierung und des Soll-/Ist-Abgleichs

Ziel des ersten Schritts in der Realisierung einer Testautomatisierungslösung soll das Design und die Implementierung einer Testautomatisierungslösung (nachfolgend *System* genannt) auf Basis des *BaSyx .NET SDK* sein. Die Vorgehensweise charakterisiert sich durch drei Phasen.

- Spezifikation der Systemanforderungen nach Analyse der bestehenden *BaSyx* Infrastruktur und unter Berücksichtigung aller Funktionalitäten und Komponenten
- Erstellung einer Architektur auf Basis der Anforderungsspezifikation
- Implementierung und Inbetriebnahme des Systems durch den Einsatz geeigneter Testobjekte

Im Rahmen des praktischen Anteils dieser Abschlussarbeit muss also ein Demonstrator synthetisiert und implementiert werden. Dieser muss eine Testung von Anlagenkomponenten ermöglichen. Der Test-Demonstrator muss Kommunikationsbeziehungen mit registrierten Verwaltungsschalen aufbauen und eine Test-Pipeline für jedes Testobjekt durchführen. Das System soll aktive Verwaltungsschalen erkennen, adressieren und ggf. parametrieren, die strukturelle Architektur der Testobjekte prüfen und in der Lage

sein, zustandsorientierte Tests durchzuführen. Die Funktionalität des zu entwickelnden Systems ist entsprechend nachzuweisen. Hierfür werden Ressourcen in Form von Verwaltungsschalen und SPS-Programmen verwendet.

4.1.2 Systemumgebung

Das *BaSys 4* Projekt wurde von der Bundesministerium für Bildung und Forschung (BMBF) zum Zweck der Realisierung eines Basissystems für Industrie 4.0 konformen Produktionsanlagen gefördert und lief bis Jahr 2019 [IES22]. Das entwickelte Basissystem gilt als technologische Antwort der Anforderungen von Industrie 4.0, wobei der Fokus auf die Wandelbarkeit der Produktionsanlagen liegt. Heutzutage wird die Softwareinfrastruktur im Rahmen der *BaSys 4.0* und *BaSys 4.2* Projekte weiterentwickelt. *Eclipse BaSyx*[1] ist die Middleware, welches im Rahmen von *BaSys 4* und den nachfolgenden Projekten als offene Plattform für die vierte industrielle Revolution vorangetrieben wird und als Instrumentierung für die digitale Anlagentransformation gilt [Fou22]. Die Softwarearchitektur[2] wird in mehreren Programmiersprachen implementiert, sodass eine Digitalisierung unabhängig von den bereits eingesetzten Technologien, v.a. Schnittstellenorientiert, stattfinden kann und die Plattform von einem breiten Spektrum von Entwicklern nutzbar ist.

Laut [Fou22] und wie auch von der I4.0 Vision vorgegeben, gelten die Verwaltungsschalen, Submodelle und Steuerungskomponenten als Kernkonzepte von *BaSys*, wobei die Verwaltungsschalen und Submodelle im Kontext der Referenzimplementierung als HTTP Server-Prozesse mit REST API distribuiert werden können. Die Softwareplattform orientiert sich nach dem in [Bad20] beschriebenen Datenmodell und bietet neben der entsprechenden API für die Realisierung passiver, reaktiver und proaktiver Verwaltungsschalen auch infrastrukturelle Softwarekomponenten in Form von Docker-Containern an, die für Persistenz von Informationen, Visualisierung von Daten, Management von Verwaltungsschalen und andere Zwecke geeignet sind. Die grundlegenden Komponenten des Basissystems sind in der Dokumentation[3] hierarchisch vorgestellt.

Mit *BaSys* als Middleware erfolgt v.a. eine Digitalisierung der Komponenten und Dienste auf der Steuerungsebene. Voraussetzung dafür ist, dass diese bestimmte Schnittstellenanforderungen erfüllen [Fou22]. Als *BaSys*-konforme Kommunikationsprotokolle gelten HTTP und OPC UA. *BaSys* unterscheidet zwischen Steuerungskomponenten, welche mit Feldgeräten interagieren, und den sog. Gruppenkomponenten. Die *BaSys*-Steuerungskomponenten (Programmable Logic Controllers, PLCs) sind über digitale und analoge IO-Schnittstellen mit Feldgeräten verbunden und werden auf Basis der IEC61131-3 und IEC61499 Paradigmen programmiert [Fou22]. Diese bieten unterschiedliche Dienste an und können als steuernde Instanzen von Feldbusnetzwerken eingesetzt werden, indem sie mit als Mastermodule von Subnetzwerken agierenden Feldbus-Slaves Daten austauschen. Als Beispiel gilt das *Cube67*-System[4] des deutschen Automatisierungspartners Murrelektronik GmbH, welches von einer mit der Steuerung kommunizierenden und im Kontext des Subnetzwerks als Master fungierenden Instanz dirigiert wird. Laut der Dokumentation dienen die Gruppenkomponenten zur Realisierung abstrakter bzw. zusammengesetzter high-level Dienste indem sie mit Steuerungs- und weiteren Gruppenkomponenten kommunizieren und ihre Dienste ausnutzen. Auf diese Weise lassen sich u.a. komplexe Fähigkeiten umsetzen. Sowohl die Steuerungs- als auch die Gruppenkomponenten werden mittels Verwaltungsschalen abstrahiert. Jede VWS kann in einer *Registry*-Komponente, der sog. *Directory*, registriert werden. Die Registry dient dann als zentrale Auskunftsstelle für AASs, die von mehreren Applikationen verwendet werden kann, und lässt sich als eine bedeutende Softwarekomponente innerhalb der *BaSys* Infrastruktur bezeichnen. Darauf basierend können übergeordnete Module komplexe Geschäftsoperationen durchführen, indem sie registrierte Verwaltungsschalen nach bestimmten Anforderungen filtern, Ressourcen buchen und Felddaten visualisieren. Hierfür definiert *BaSys* zwei high-level Services. Die sog. *Strategy*- bzw. *Optimization* Komponente ist für die Erstellung des Produktionsplans und die Buchung der entsprechenden Ressourcen verantwortlich. Die

[1] `https://wiki.eclipse.org/BaSyx`
[2] `https://github.com/eclipse-basyx`
[3] `https://wiki.eclipse.org/BaSyx_/_WhatIsBasyx`
[4] `https://www.murrelektronik.com/highlights/cube67/`

Prozessteuerungskomponente (engl. *Process control*) führt die im Produktionsplan spezifizierten Schritte durch. Da diese Dienste anlagenspezifisch sind, ist z.Zt. keine Implementierung verfügbar, anhand welcher diese weiter analysiert werden können.

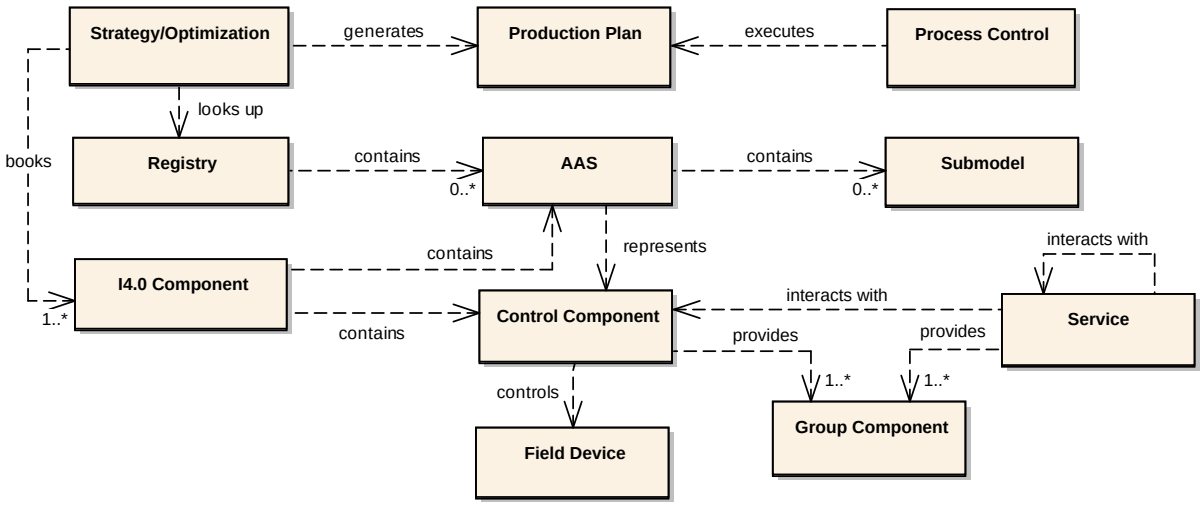


Abbildung 4.1: *BaSyx* - Kernkomponenten und ihre Interaktionen

Es ist anzumerken, dass in der Dokumentation keine Komponenten vorgestellt sind, die sich für die Durchführung eines *Feasibility Check* eignen. Die Zuordnung dieser Aufgabe zu den Verantwortungen der Prozessteuerungs- oder Strategy-Komponente wäre ein Verstoß gegen das *Single-Responsibility*-Prinzip. Der allgemeine Zusammenhang zwischen den Kernkomponenten des Basissystems ist zusammen mit der Idee von *BaSys* in Abb. 4.1 exemplarisch dargestellt. Die zu entwickelnde Testlösung muss als Teil der *BaSyx*-Softwareinfrastruktur implementiert werden und mit den bereits existierenden Komponenten des Basissystems interagieren.

4.2 Soll-Konzept

In diesem Abschnitt wird eine prototypische Testumgebung vorgeschlagen und konzeptionell beschrieben. Als Grundlage des Konzeptes gelten die von Schäfer u. a. in [Sch22b] und [Sch22a] vorveröffentlichten Ansätze. Im Rahmen des Kapitels werden diese verfeinert. Darauf basierend erfolgt die Konzeption einer Softwarearchitektur, die als Möglichkeit für die Umsetzung der vorgeschlagenen Testlösung gilt.

Während die Konzeption und Beschreibung einer vollständigen Umgebung zum Testen von Industrie 4.0 Komponenten als Ziel des Kapitels dient, ist eine vollständige Implementierung aller nachfolgend vorgeschlagenen Funktionalitäten aufgrund der begrenzten zeitlichen und zur Verfügung gestellten Softwareressourcen außerhalb des Scopes dieser Abschlussarbeit. Im Rahmen der Implementierungsphase wird ein Teil der vorgeschlagenen Softwarearchitektur zum Zweck der Konzeptprüfung umgesetzt.

4.2.1 Systemfunktionalität

In diesem Abschnitt werden die Hauptfunktionalitäten des Systems aus Sicht des Anwenders beschrieben. Die folgende Abbildung stellt die Haupt-Anwendungsfälle einer möglichen Testlösung dar. Die einzelnen Elemente der Abbildung werden in den nachfolgenden Unterabschnitten diskutiert.

Allgemeine Anwendungsszenarien

Die folgende Abbildung liefert einen groben Überblick über die Funktionalitäten des Systems. Aus dem Use-Case Diagramm geht hervor, dass die Testlösung zwei Testdurchführungsstrategien unterstützt und zum Zweck der Auswertung Testberichte generiert. Während die erste Teststrategie die Existenz der BaSyx Registry-Komponente voraussetzt, bildet die Grundlage für die zweite Verifikationsmöglichkeit die Nutzung eines Ticketverzeichnisses.

Das Modell wird in diesem Abschnitt oberflächlich erläutert, um das allgemeine Konzept zu veranschaulichen. Im folgenden Unterabschnitt erfolgt eine Erweiterung und detaillierte Beschreibung der primären Anwendungsszenarien.

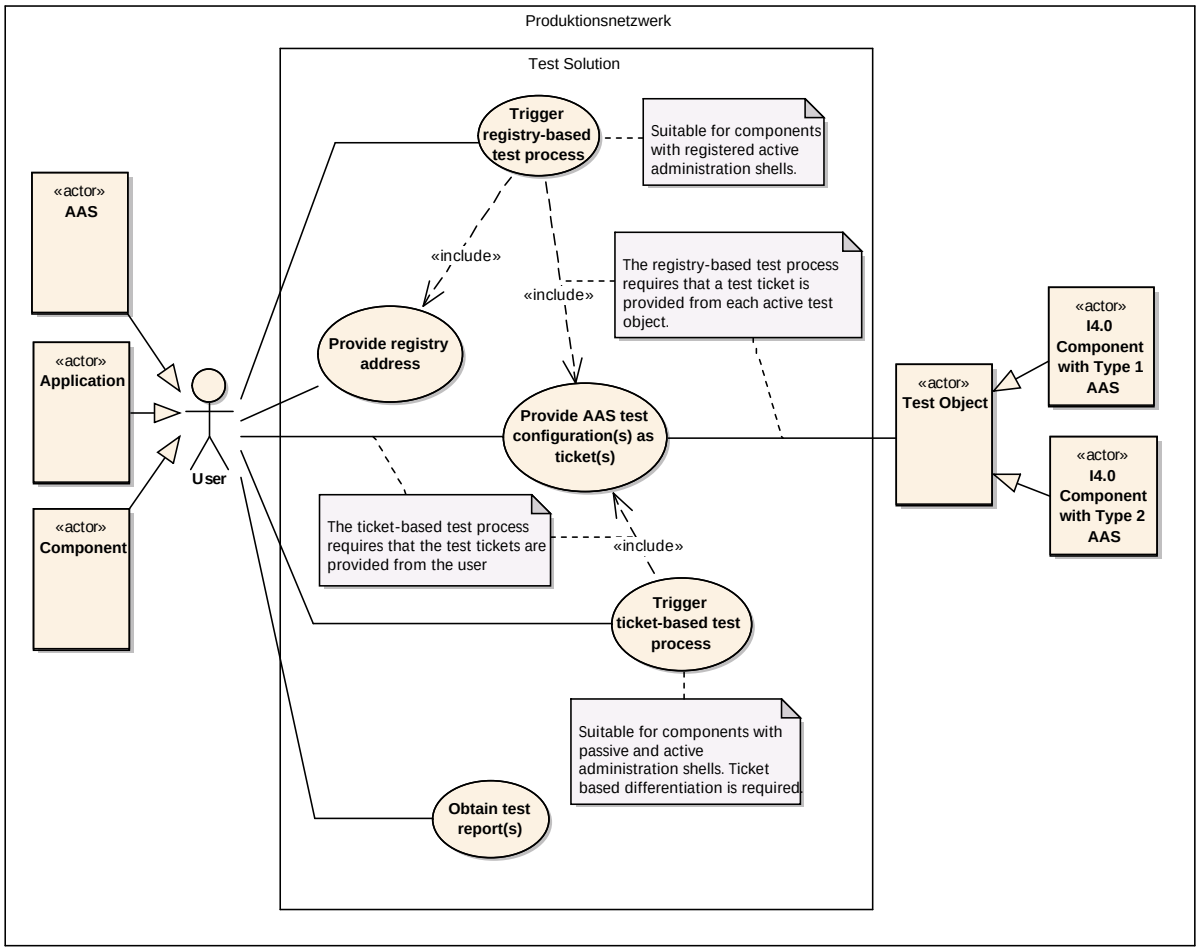


Abbildung 4.2: Allgemeine Anwendungsszenarien

4.2.1.1 User

Der Anwender des Systems kann sowohl einen Integrator als auch eine andere im gleichen Netzwerk integrierte Komponente sein. Im Rahmen des Diagramms werden Typ 2 bzw. Typ 3 Verwaltungsschalen, Softwareapplikationen und Anlagenkomponenten wie z.B. Edge-Gateways als potenzielle Nutzer der Testlösung angesehen. Jede Entität, die mit einer Verwaltungsschale über die entsprechende Schnittstelle interagieren kann, gilt als potenzieller Anwender der Testlösung. Als Schnittstelle zur Interaktion dient die BaSyx AAS HTTP REST API[5] und die BaSyx Web UI.

[5] `https://app.swaggerhub.com/apis/BaSyx/basyx_asset_administration_shell_http_rest_api/v1`

4.2.1.2 Test Object

Als Testobjekt gilt jede Industrie 4.0 Komponente, die durch eine passive oder reaktive VWS repräsentiert wird. Zum Zweck der Kommunikation mit der Testlösung müssen alle reaktiven Verwaltungsschalen die BaSyx AAS HTTP REST API implementieren.

4.2.1.3 Provide AAS test configuration(s) as ticket(s)

In Abhängigkeit von der Testdurchführungsstrategie müssen die Test-Rezepte entweder vom Anwender oder vom Testobjekt bereitgestellt werden.

4.2.1.4 Trigger registry-based test process

Die erste Testdurchführungsstrategie basiert auf die Verfügbarkeit einer Registry-Komponente und erfordert die Existenz bestimmter Test-Rezepte (Tickets) als Teil eines vordefinierten Submodells innerhalb der Verwaltungsschale des Testobjekts. Jedes Ticket dient als Beschreibung der Erwartungen an das Testobjekt und umfasst weitere Informationen, die zum Zweck der Testdurchführung notwendig sind.

4.2.1.5 Provide registry address

Da die BaSys-Registrierungskomponente durch eine aktive Verwaltungsschale repräsentiert wird, ist zum Zweck des Verbindungsaufbaus zwischen dem Orchestrator und der Registry die Angabe der entsprechenden Adresse erforderlich.

4.2.1.6 Trigger ticket-based test process

Im Rahmen des zweiten Szenarios werden die Tickets vom Anwender des Systems bereitgestellt. Darauf basierend erfolgt die Verifikation aller durch Tickets beschriebenen Testobjekte.

4.2.1.7 Obtain test report(s)

Die Testergebnisse werden in Form von Test-Reports dem Anwender bereitgestellt.

Kernfunktionalitäten

Dieser Abschnitt beschreibt die Kernfunktionalitäten des Systems aus Sicht des Anwenders und soll einen Eindruck der gewünschten Funktionen des Systems liefern. Die folgende Abbildung stellt die Hauptfunktionalitäten des Systems dar. Die einzelnen Elemente der Abbildung sind nachfolgend detailliert erläutert.

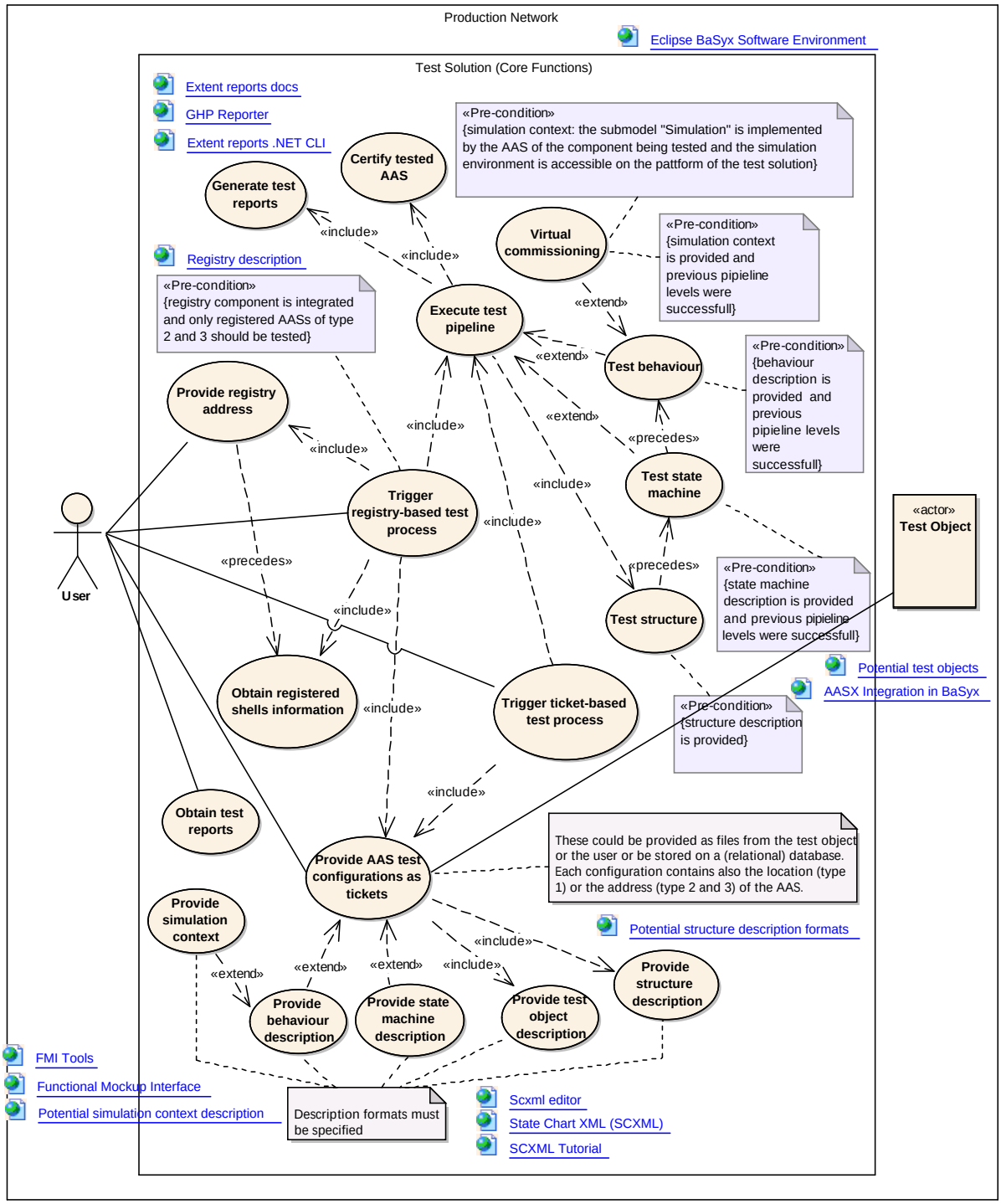


Abbildung 4.3: Kernfunktionalitäten

4.2.1.8 Trigger registry-based test process

Die Registry-Komponente dient im Rahmen der *BaSys*-Middleware als *Single Point of Information* für Typ 2 und Typ 3 Verwaltungsschalen. Aus diesem Grund informieren Schäfer u. a., dass diese als Basis für die Durchführung eines Testprozesses eingeführt werden kann. Die Voraussetzungen dafür sind, dass die Registrierungskomponente als Teil des Produktionsnetzwerks integriert und eine Kommunikationsbeziehung zwischen der Testautomatisierungslösung und dem *Directory* aufgebaut wird. Laut [Sch22a] besteht der Nachteil dieses Ansatzes darin, dass Vertreter, die als Dateien distribuiert werden und keine Kommunikationsschnittstelle besitzen, im Rahmen der Registry-basierten Nutzung nicht verifizierbar sind. Im Verlauf des Testprozesses wird für jede registrierte VWS und die entsprechende Anlagenkomponente eine vordefinierte Testsequenz durchgeführt, die im Folgenden als *Test-Pipeline* bezeichnet wird [Sch22a].

Die Bereitstellung eines Test-Rezeptes für jede registrierte VWS erfordert vom Anwender Kenntnis über die Anzahl der Registrierungen und spezifisches Wissen für jedes Testobjekt. Die Anzahl der registrierten Verwaltungsschalen ist für den Anwender nicht immer bekannt und das System muss von Anwendern ohne technisches Wissen nutzbar sein. Eine Lösung der Problematik bietet die Integration des Test-Rezeptes innerhalb der Verwaltungsschale des Testobjekts an. Das Rezept bzw. das Ticket kann vom Komponentenlieferanten bereitgestellt werden. Soll diese im Rahmen des Lebenszyklus der Komponente nicht manipuliert werden können, sind die entsprechende Schutz- bzw. Security-Mechanismen erforderlich.

4.2.1.9 Provide registry address

Als Voraussetzung für die Durchführung eines Testprozesses für alle in einer Registry angemeldeten Verwaltungsschalen, muss das System dem Anwender die Möglichkeit bieten, die URL-Adresse der Registry Komponente bereitzustellen. Die Bereitstellung kann in Form einer Datei, als Argument beim Applikationsstart oder über die UI erfolgen.

4.2.1.10 Obtain registered shells information

Im Rahmen der registry-orientierten Testdurchführung muss der Demonstrator Informationen von jeder registrierten VWS anfordern können. Zum Zweck der Verifikation kann die VWS des Testobjekts lokal im Speicher vorliegen. Ist die Bereitstellung einer kompletten VWS über die REST API nicht möglich, so können die Informationen durch die entsprechenden Anfragen im Speicher geladen werden.

4.2.1.11 Provide AAS test configurations as tickets

Das System muss dem Anwender die Möglichkeit bieten, ein Test-Rezept für jedes Testobjekt bereitzustellen. Jedes Ticket gilt als Grundlage für die Testdurchführung. Auf Basis der Bestandteile des bereitgestellten Rezeptes muss der Orchestrator die vorzunehmende Schritte der Test-Pipeline automatisiert ableiten und ausführen können.

Bei dem Registry-orientierten Testprozess sind die Test-Tickets von den Testobjekten zu erwarten. Hierfür wäre die Spezifikation eines Submodells oder einer Sammlung von Eigenschaften zum Zweck der Repräsentation des Tickets durch das Metamodell erforderlich.

4.2.1.12 Provide test object description

Die Beschreibung des Testobjekts ermöglicht eine Unterscheidung zwischen passiven und aktiven Verwaltungsschalen. Diese kann den Typ der VWS des Testobjekts und/oder seine URL-Adresse umfassen. Auf Basis dieser information findet die Konstruktion des Testobjekts im Kontext des Softwaresystems statt. Die Beschreibung soll als Teil des Test-Rezeptes bereitgestellt werden.

4.2.1.13 Provide structure description

Unabhängig davon, von welchem Typ die VWS des Testobjekts ist, muss der Anwender oder das Testobjekt der Testlösung eine Beschreibung bereitstellen, welche die Soll-Struktur der VWS in einer maschineninterpretierbaren Form spezifiziert. Anhand dieser Information können Strukturtests durchgeführt werden. Unternehmen und Organisationen, deren Verwaltungsschalen interne und strukturelle Richtlinien einhalten müssen, können durch die Spezifikation einer allgemeinen Strukturbeschreibung verifiziert werden. Die Strukturbeschreibung soll als Teil des Tickets bereitgestellt werden.

4.2.1.14 Provide state machine description

Repräsentiert das Testobjekt eine Zustandsmaschine, so ist die Verifikation ihrer Funktionsweise notwendig. Die Testung erfolgt durch die Initiierung von Transitionen und die Überprüfung, ob der Zustand nach der Transition mit den Anforderungen korrespondiert. Da von jeder VWS eine unterschiedliche Zustandsmaschine implementiert werden kann, ist auch hier Ein Test-Rezept erforderlich, welches das Verhalten der Führungskomponente beschreibt. Auf Basis der Inhalte dieser Beschreibung werden die Zustandstests durchgeführt. Aus diesem Grund muss das Rezept vertrauenswürdig sein. Diese Verifikation ist nur bei aktiven Verwaltungsschalen, die Führungskomponenten repräsentieren, durchzuführen. Die Beschreibung soll als Teil des Test-Rezeptes bereitgestellt werden.

4.2.1.15 Provide behaviour description

Handelt es sich um eine aktive VWS als Testobjekt, so kann zum Ticket eine Verhaltensspezifikation hinzugefügt werden, die als Grundlage für die Durchführung von Unit-Tests dient. Da z. Zt. die Implementierung von Fähigkeiten seitens der VWS nicht vollständig spezifiziert ist, werden im Rahmen des Dokuments als Verhalten die Operationen angesehen, die von einer VWS über die Submodelle angeboten werden. Zum Zweck der Verifikation dieser Operationen muss die Verhaltensspezifikation die Identifikationsdaten der Operation, die Testdaten und die erwarteten Ergebnisse definieren.

4.2.1.16 Provide simulation context

Eine vollständige Testlösung soll zum Zweck der Verifikation eines von Typ 2 oder Typ 3 VWS repräsentierten Testobjekts als Teil einer Produktionsumgebung die automatisierte Durchführung einer virtuellen Inbetriebnahme ermöglichen. Hierfür benötigt der Orchestrator einen Simulationskontext. Dieser kann entweder vom Anwender als Datei oder vom Testobjekt als Submodell bereitgestellt werden. Als Vorlage kann die Struktur des Submodells Simulation[6] in Betracht gezogen werden. Die Durchführung einer VIBN ist von der Simulationsumgebung und den angebotenen Schnittstellen stark abhängig. Hierfür wäre die Festlegung und Standardisierung einer (REST) API sinnvoll, welche als Abstraktionsschicht zwischen der konkreten Simulationsumgebung und ihren Nutzern dient. Diese Abstraktion würde die Unabhängigkeit jeder Testlösung von der einzusetzenden Simulationsumgebung garantieren und kann in Form einer Verwaltungsschale umgesetzt werden.

4.2.1.17 Trigger ticket-based test process

Wie in [Sch22a] beschrieben, ist der ticket-basierte Testprozess im Gegensatz zur Registry-orientierten Testung für passive, reaktive und proaktive Verwaltungsschalen geeignet. Laut den Autoren ist es eine Frage der Spezifikation, ob das Test-Rezept als Entität einer relationalen Datenbank oder als XML oder JSON-Datei der Testlösung zur Verfügung gestellt wird. Die Interaktion mit dem Ticket-System kann über die Dienste eines Submodells stattfinden, wodurch die Verwendung von spezifischen Programmiersprachen seitens des Benutzers ausgeschlossen, eine Validierung der Bestandteile des Test-Rezeptes ermöglicht und eine Schnittstelle zum Ticket-Repository für übergeordnete Orchestrierungssysteme angeboten wird.

Als Eingangsinformationen für diesen Testprozess gelten Tickets, welche sowohl das Testrezept als auch den Zugriffspunkt des Testobjekts beschreiben [Sch22a]. Zum Zweck der Verifikation passiver Vertreter muss der Inhalt der entsprechenden AAS-Datei als Argument bei der Erstellung des Tickets eingegeben werden. Als Voraussetzung für die Testung aktiver Komponenten definieren die Autoren die Eingabe der entsprechenden URL-Adresse.

4.2.1.18 Execute test pipeline

Im Rahmen des Testprozesses muss das System eine Test-Pipeline durchführen. Im Laufe der Pipeline sind die entsprechenden Teststufen auf Basis des vom Anwender bereitgestellten Test-Rezeptes vom Demonstrator zu identifizieren und durchzuführen.

4.2.1.19 Test structure

Auf Basis der vom Anwender bereitgestellten Strukturbeschreibung muss das System die Struktur der VWS der zu testenden Komponente verifizieren. Dieser Vorgang lässt sich darüber motivieren, dass die Struktur jeder VWS jederzeit manipulierbar ist. Als Strukturinformation dient sowohl die allgemeine Struktur der VWS in Hinsicht auf ihre Submodelle als auch die Anordnung der einzelnen Elemente jedes Teilmodells. Jede Informationseinheit, unabhängig davon, ob es sich um ein Submodell, Property oder

[6] `https://github.com/admin-shell-io/submodel-templates/tree/main/development/Simulation/1/0`

eine Operation handelt, gilt im Kontext der VWS als Strukturinformation, deren Existenz überprüft werden muss.

Die potenzielle Verifikation eines VWS- oder Submodell-Elements, z.B. ein Submodell oder eine Operation, erfordert das Zugreifen bzw. die Existenz der übergeordneten Informationseinheit. Ist diese nicht vorhanden, so ist ein Testen des Elements ausgeschlossen.

Der Sachverhalt wird anhand Abbildung 4.4 veranschaulicht. Auf der linken Seite ist eine vereinfachte VWS-Struktur dargelegt. Es ist eine einfache Verwaltungsschale mit einem Submodell, drei atomaren Eigenschaften und eine Sammlung von Submodell-Elementen zu sehen. Zum Zweck des Beispiels entspricht das letzte Element der Kollektion einer Operation, die im Rahmen der Test-Pipeline validiert werden soll und mit einem Verhalten zu assoziieren ist. Damit diese von einer Testlösung überhaupt zugegriffen werden kann, ist

- die Existenz des übergeordneten Submodells und
- die Existenz der Sammlung aus Submodell-Elementen

sicherzustellen. Nach der Verifikation, dass diese als Teil der VWS-Struktur enthalten sind, kann die Existenz des eigentlichen Testobjekts, in diesem Fall der Operation, überprüft werden. Erst nach diesem Vorgang kann die Operation als Gegenstand einer weiteren Stufe der Test-Pipeline angesehen werden.

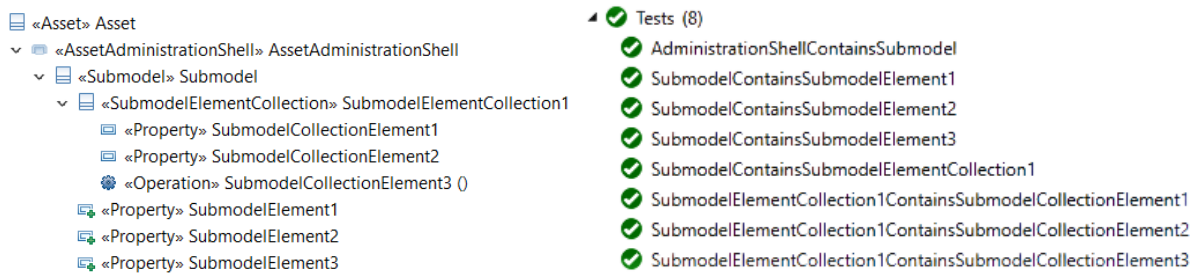


Abbildung 4.4: Strukturorientierte Testdurchführung - Notwendigkeit und Vorgehensweise

4.2.1.20 Test state machine

Ist für das Testobjekt ein Rezept bereitgestellt worden, das eine Zustandsmaschine beschreibt, so muss die richtige Funktionsweise der Führungskomponente über die Verwaltungsschale entsprechend nachgewiesen werden. Voraussetzung dafür ist, dass

- die erfolgreiche bidirektionale Kommunikation zwischen dem Asset und der VWS sichergestellt,
- die Steuerung des Automats über die Verwaltungsschale möglich und
- den aktuellen Zustand der Führungskomponente als Eigenschaft der VWS enthalten ist - also über die REST API ausgelesen werden kann.

Damit es der Testlösung bekannt ist, über welche Eigenschaften der Zustand der Führungskomponente ausgelesen bzw. gesetzt werden kann, muss das Rezept die Identifikatoren der entsprechenden VWS-Eigenschaften spezifizieren.

Im Rahmen der Testung initiiert die Testlösung Transitionen. Nach jeder Transition wird der aktuelle Zustand des Testobjekts mit dem in der Beschreibung spezifizierten erwarteten Zustand verglichen.

4.2.1.21 Test behaviour

Das über die VWS vom Asset zur Verfügung gestellte Verhalten ist vom Testbett auf Basis der entsprechenden Beschreibung zu verifizieren. Da Unterschiedliche Möglichkeiten zur Darstellung von Verhalten im Kontext der Verwaltungsschale existieren (Fähigkeiten, Services, Operationen usw.) und

- keine maschinelle Interpretation der Fähigkeiten auf Grund der unterschiedlichen Verwendung der Begrifflichkeit gegeben ist und

- keine beispielhafte Implementierung von Fähigkeiten zur Verfügung steht,

können im Rahmen der Verhaltenstests nur die von den Submodellen bereitgestellten Operationen auf Grundlage des Test-Rezeptes mittels Unit-Tests überprüft werden.

4.2.1.22 Virtual commissioning

Wird vom Anwender oder dem Testobjekt ein Simulationskontext bereitgestellt, so ist die Durchführung einer virtuellen Inbetriebnahme notwendig. Voraussetzung für diese ist die Verfügbarkeit der im Simulationskontext angegebenen Simulationsumgebung auf der Plattform der Testlösung und die Ausführung der spezifizierten Simulationsdatei.

4.2.1.23 Certify tested AAS

Jede aktive und getestete VWS kann vom Demonstrator entsprechend gekennzeichnet werden.

4.2.1.24 Generate test reports

Im Verlauf der Testdurchführung muss der Demonstrator Test-Reports generieren.

4.2.1.25 Obtain test reports

Das System muss in der Lage sein, die im Laufe der Testdurchführung generierten Test-Reports dem Anwender nach Aufforderung bereitstellen zu können.

4.2.2 Systemkontext

Dieser Abschnitt gibt einen groben Einblick des Test-Demonstrators im Kontext seiner Umgebung an und dient als Information für die Ableitung der in den folgenden Abschnitten definierten Anforderungen.

Registry-orientierter Testprozesskontext

Die folgende Abbildung stellt die Kommunikationsbeziehungen zwischen den Komponenten im Kontext eines Registry-orientierten Testprozesses dar.

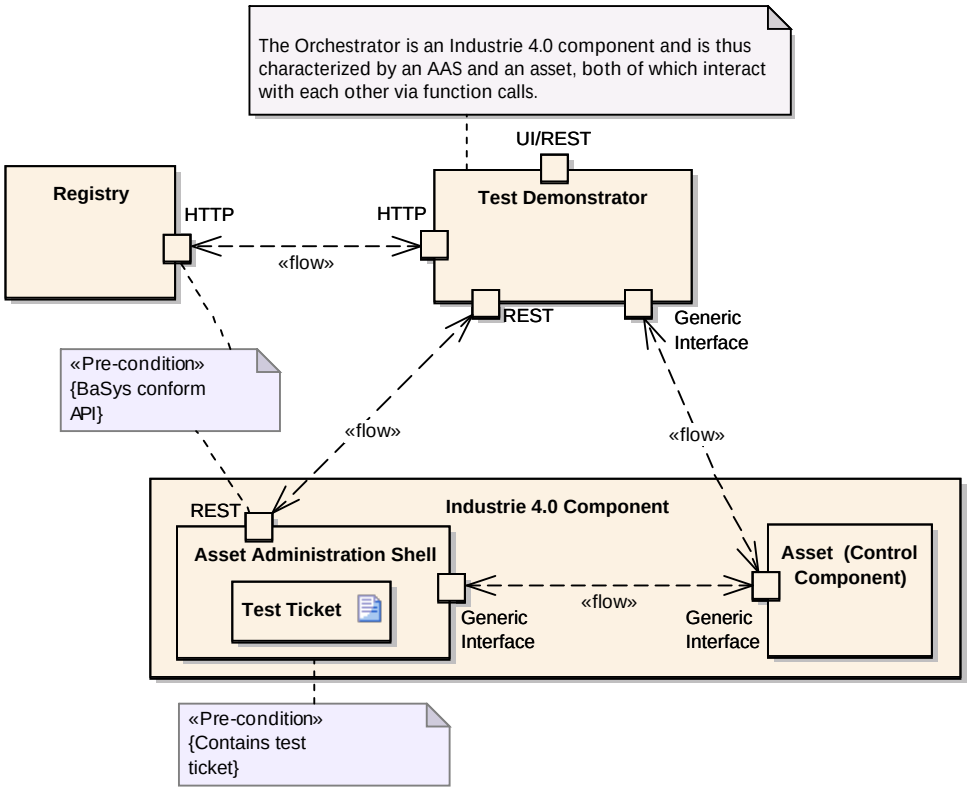


Abbildung 4.5: Registry-orientierte Testumgebung [Sch22a]

4.2.2.1 Asset (Control Component)

Die Steuerungskomponente gilt als Bestandteil der Industrie 4.0 Komponente und kommuniziert mit der VWS über eine geeignete Schnittstelle.

4.2.2.2 Asset Administration Shell

Die Testung der Anlagenkomponente erfolgt auf Basis der Verwaltungsschale. Hierfür gilt die entsprechende REST API als Basis. Das Test-Rezept muss in einer geeigneten Form in der VWS enthalten sein. Ist das nicht der Fall, so lässt sich das Testobjekt nicht verifizieren.

4.2.2.3 Industrie 4.0 Component

Als Testobjekt gilt jede Steuerungskomponente, die von einer Typ 1, Typ 2 oder Typ 3 Verwaltungsschale repräsentiert wird. Es ist für den Orchestrator ohne Bedeutung, wie die Steuerungskomponente mit der Verwaltungsschale kommuniziert bzw. wie die Daten der VWS von der Steuerungskomponente aktualisiert oder ausgelesen werden. Die Schnittstelle der Industrie 4.0 Komponente entspricht der Schnittstelle der Verwaltungsschale.

4.2.2.4 Test Demonstrator

Im Rahmen einer registry-basierten Testung baut der Demonstrator einen Kommunikationskanal zur Registry auf und fragt die Adressen der registrierten Verwaltungsschalen ab. Für jede Adresse wird eine Verbindung zum Testobjekt realisiert. Dazu gehört sowohl der Aufbau einer Kommunikationsbeziehung zur Verwaltungsschale als auch die direkte Verbindung der Testlösung zum Asset. Auf diese Weise lässt sich die richtige Synchronisation der beiden Bestandteile der Industrie 4.0 Komponente nachweisen. Während der Orchestrator mit der Registry und der VWS über bereitgestellte APIs kommuniziert, ist der Informationsaustausch mit dem Asset vom eingesetzten Kommunikationsprotokoll abhängig. Eine Beschreibung des Asset-Endpoints wäre für diese Kommunikationsbeziehung ebenfalls notwendig. Hierfür wäre die Standardisierung eines Submodells sinnvoll, das diese Informationen enthält. Als Basis könnte das von der IDTA geplante Asset Interface Description Submodell[7] dienen. Eine weitere Möglichkeit bietet die Standardisierung von OPC UA als Kommunikationsprotokoll zwischen der AAS und dem Asset.

4.2.2.5 Test Ticket

Das Ticket beinhaltet alle Daten, die zum Zweck der Durchführung der Testpipeline notwendig sind. Im Rahmen einer umfassenden und vollständigen Verifikation aktiver Verwaltungsschalen setzt sich ein Ticket aus den folgenden Deskriptoren zusammen:

- Uniform Resource Identifier (obligatorisch)
- Beschreibung der Schnittstelle zum Asset (optional)
- Strukturbeschreibung (obligatorisch)
- Beschreibung des Zustandsautomaten (optional)
- Beschreibung des Verhaltens (optional)
- Simulationskontext (optional)

Allgemeiner Ablauf

Das folgende Sequenzdiagramm beschreibt den prinzipiellen Ablauf einer Registry-orientierten Testdurchführung und veranschaulicht die allgemeine Kommunikation zwischen den Komponenten. Die Abbildung stellt keine vollständige oder optimale Implementierung, sondern einen Testablauf auf konzeptioneller Ebene dar.

Wie bereits erwähnt wurde, gilt als Voraussetzung für den Testprozess der Kommunikationsaufbau zwischen den Komponenten. Zum Zweck der dynamischen Spezifikation der Auskunftsstelle für Testobjekte, also der Registry, wird dem Anwender die Möglichkeit angeboten die Adresse der Registrierungskomponente dem System bereitzustellen und damit diesen Kommunikationsaufbau zu initiieren (1.0).

Während die erfolgreiche Verbindung zur Registry die erste Voraussetzung für diesen Testprozess ist, entspricht die Existenz des Test-Tickets als Teil der VWS des Testobjekts einem weiteren kritischen Punkt.

Nach dem Kommunikationsaufbau lässt sich eine Testsequenz für jedes registrierte Testobjekt initiieren (1.5). Die API soll außerdem Funktionalitäten zur Verifikation von konkreten Industrie 4.0 Komponenten anbieten. Es werden die Adressen der registrierten Verwaltungsschalen vom Orchestrator abgefragt (1.9). Zum Zweck der Kommunikation mit dem Testobjekt wird für jede Registrierung ein AAS HTTP Client instanziiert (1.11). Über die HTTP Verbindung erfolgt die Bereitstellung des entsprechenden Test-Tickets (1.13). Jedes Ticket wird im Kontext der Softwareapplikation als Instanz der Klasse `TestTicket` integriert (1.15). Dies gilt als Voraussetzung für den weiteren Ablauf des Testprozesses.

[7] `https://github.com/admin-shell-io/submodel-templates/tree/main/development/Asset%20Interface%20Description/1/0`

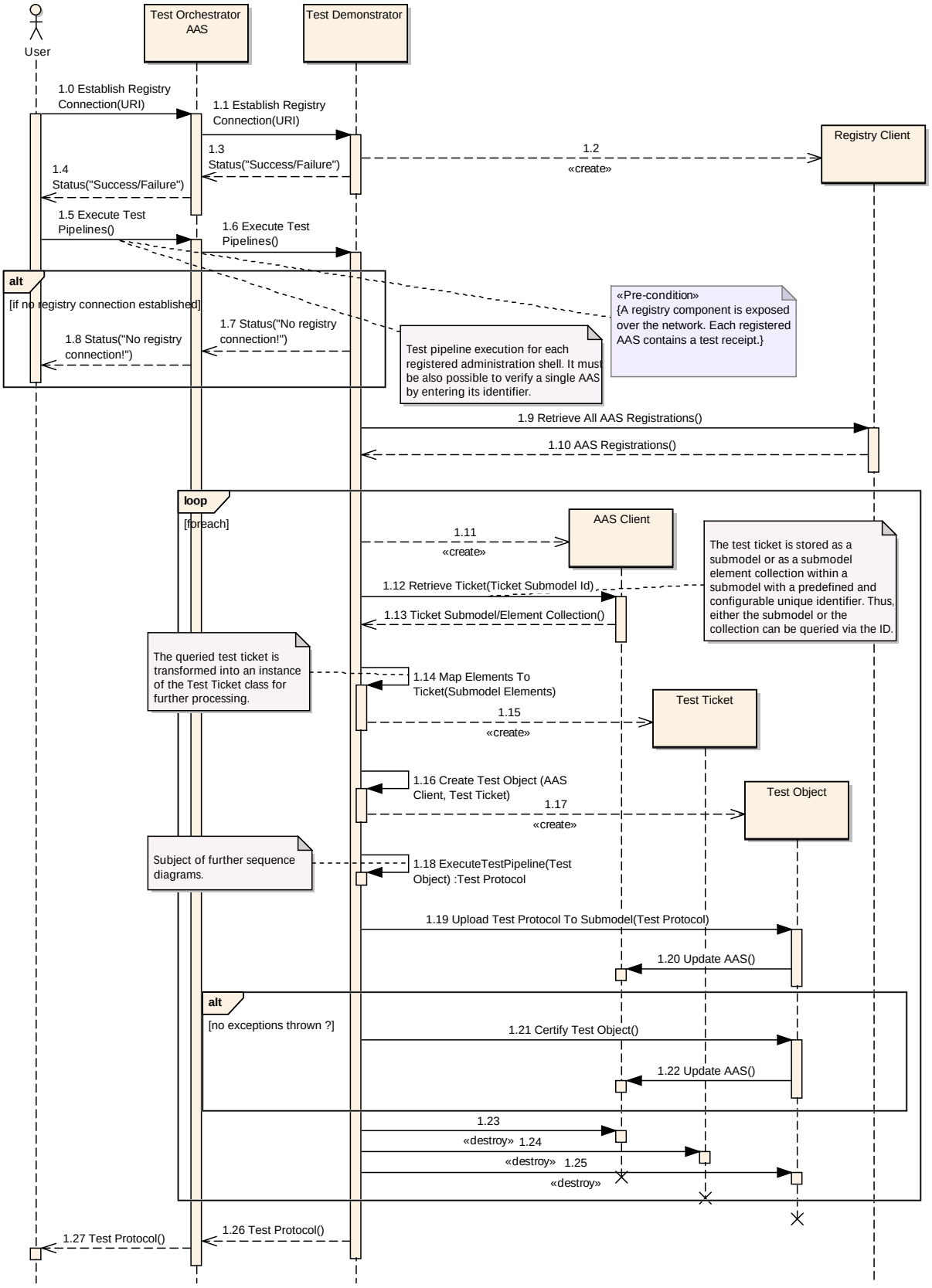


Abbildung 4.6: Registry-orientierte Testdurchführung - Ablauf

Aus dem Ticket und der AAS Client-Instanz wird vom System ein Testobjekt konstruiert (1.17). Dieses dient als Argument der Funktion zur Durchführung der Test-Pipeline (1.18). Im Rahmen der Testsequenz werden alle Tests entsprechend protokolliert. Nach der Testdurchführung erfolgt die Instanziierung des Test-Protokolls innerhalb der VWS des Testobjekts (1.19).

Sind im Rahmen der Pipeline keine Ungenauigkeiten identifiziert worden, so lässt sich eine Kennzeichnung der Verwaltungsschale des Testobjekts durchgeführen (1.21).

Nach der Verifikation aller registrierten Verwaltungsschalen wird ein allgemeines Protokoll für alle Testobjekte erstellt und dem Anwender bereitgestellt (1.27).

Ticket-orientierter Testprozesskontext

Wie bereits erwähnt wurde, besteht die Idee der Ticket-orientierten Testdurchführung in der direkten Spezifikation und Zuordnung von Test-Rezepten an bereits integrierte Industrie 4.0 Komponenten. Zum Zweck der Persistenz der Ticket-Informationen kann eine Datenbank, ein File-System oder eine Cloud-Infrastruktur zum Einsatz kommen. Unabhängig davon, wo genau die Tickets gespeichert werden, lässt sich die Komponente, welche für die Persistenz der Informationen verantwortlich ist, als Ticket-Repository bezeichnen. Die folgende Abbildung illustriert die Hauptkomponenten des Systems im Rahmen eines ticket-orientierten Testdurchführungsprozesses.

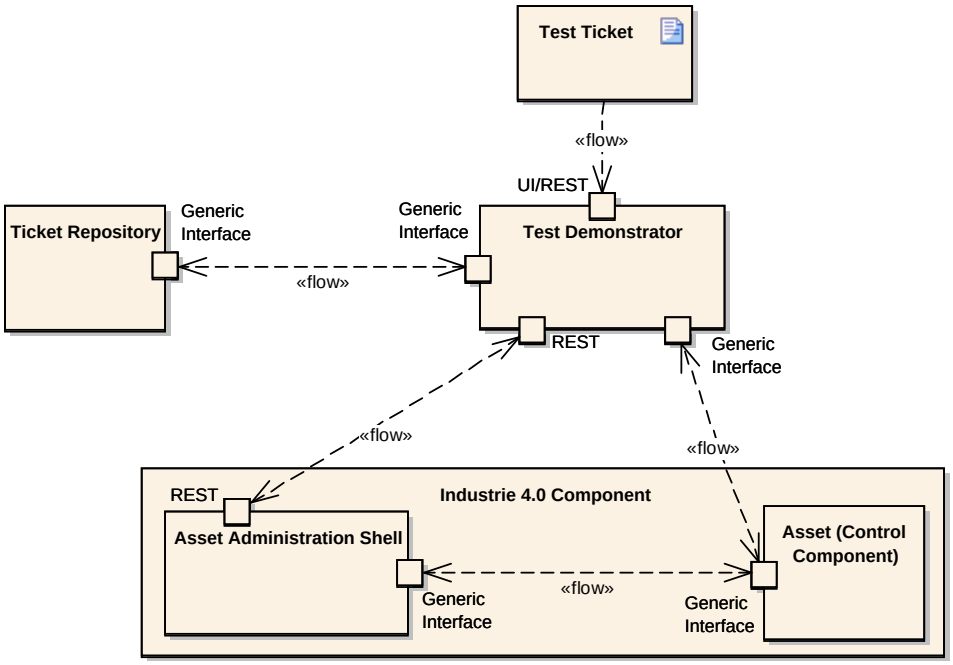


Abbildung 4.7: Ticket-orientierte Testumgebung

4.2.2.6 Test Demonstrator

Die Test-Rezepte werden dem Demonstrator über die entsprechende UI- oder REST-Schnittstelle vom Anwender übergeben und innerhalb des Ticket-Verzeichnisses persistiert. Im Rahmen der Testdurchführung extrahiert der Demonstrator das jedem Testobjekt zugeordnete Ticket und führt die Test-Pipeline durch.

4.2.2.7 Ticket Repository

Das Ticketverzeichnis dient als Auskunftsstelle für Test-Rezepte. Der Anwender hat keinen direkten

Schreibzugriff zu dieser Komponente. Die Interaktion zwischen dem Anwender und dem Tickerverzeichnis erfolgt über die Verwaltungsschale der Testlösung.

4.2.2.8 Test Ticket

Im Rahmen des Testprozesses können Komponenten mit passiven und aktiven Verwaltungsschalen getestet werden. Jedes vom Anwender des Systems bereitgestellte und gültige Ticket wird einer eindeutigen Identifikationskennzeichnung vom Orchestrator automatisch zugewiesen. Die Inhalte der Tickets werden in den folgenden Abschnitten weiter diskutiert.

Allgemeiner Ablauf

Das folgende Sequenzdiagramm beschreibt den prinzipiellen Ablauf einer Ticket-orientierten Testdurchführung und veranschaulicht die allgemeine Kommunikation zwischen den Komponenten. Die Abbildung stellt keine vollständige oder optimale Implementierung, sondern einen Testablauf auf konzeptioneller Ebene dar.

Dem Anwender wird die Möglichkeit angeboten, Test-Tickets im Ticket-Repository hochzuladen (1.0), abzufragen, zu aktualisieren und zu löschen. Im Rahmen des Testprozesses (1.1) werden die Tickets aus dem Verzeichnis ausgelesen (1.5). Für jedes Ticket erfolgt die Instanziierung der Klasse `Test Ticket` (1.6). Es wird also ein Mapping zwischen dem Ticket innerhalb des Verzeichnisses und dem Ticket im Kontext der Softwareapplikation durchgeführt. Während alle Tickets, die für die Verifikation von Industrie 4.0 Komponenten mit aktiven Verwaltungsschalen eingesetzt werden, zum Zweck des Aufbaus der Kommunikationsbezihung die URI der VWS beinhalten, wird mit den Tickets aller passiven Repräsentanten die VWS-Datei vom Anwender bereitgestellt.

Das Test-Ticket wird für die Konstruktion einer Instanz der Klasse `TestObject` verwendet (1.7). Wird dem Konstruktor ein Ticket bereitgestellt, das eine URI beinhaltet, so wird eine Kommunikationsbeziehung zur Verwaltungsschale aufgebaut (1.8). Obwohl im Fall einer passiven VWS alle für die Testdurchführung notwendigen Informationen im Ticket enthalten sind, erfordert die Schnittstelle des Moduls zum Ausführen der Test-Pipeline die Übergabe einer Instanz der Klasse `TestObject`.

Im nächsten Schritt wird also die Pipeline durchgeführt (1.9). Es erfolgt eine Zuordnung des im Rahmen der Pipeline erstellten Test-Protokolls zum Test-Ticket und eine darauf folgende Aktualisierung des Tickets im Verzeichnis (1.10).

Sind keine Abweichungen zwischen den Testergebnissen und den Ticket-Deskriptoren im Rahmen der Pipeline identifiziert worden, so erfolgt eine Kennzeichnung des Testobjekts (1.11). Nach der Durchführung der Sequenz für jedes Testobjekt wird ein allgemeines Testprotokoll für alle Testobjekte erstellt und dem Anwender bereitgestellt (1.16).

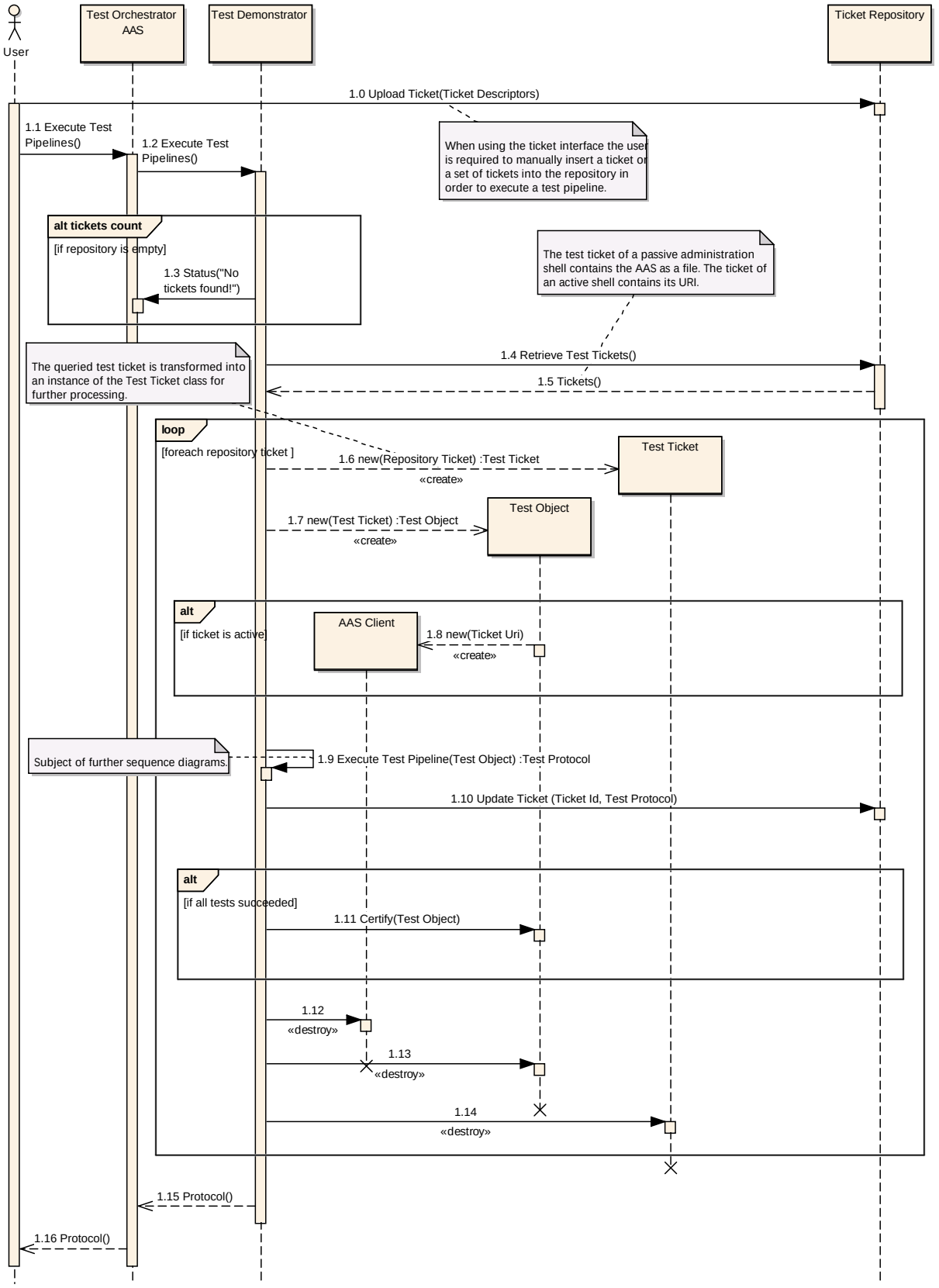


Abbildung 4.8: Ticket-orientierte Testduchführung - Ablauf

4.2.3 Anforderungen

Dieser Abschnitt beschreibt die funktionalen Systemanforderungen zusammen mit den im Rahmen der Konzipierung getroffenen Annahmen und den daraus resultierenden Abhängigkeiten. Unabhängig von den vorgeschlagenen Charakteristiken muss die Softwarekomponente jederzeit mit weiteren Funktionalitäten ausgestattet werden können.

Annahmen und Abhängigkeiten

Softwareumgebung:

1. Das System muss unter Verwendung des *Eclipse BaSyx SDK* entwickelt werden. Eine weitere Möglichkeit für die Umsetzung von Prototypen bietet jedoch auch den sog. *FAAAST*[8]-Dienst an.
2. Das System wird in C# mit der Entwicklungsumgebung *Visual Studio 2022* und *.NET 6* entwickelt.
3. Für die Umsetzung der Ticket-Repository ist eine SQL-Datenbank vorgesehen. Im Rahmen der Konzeptprüfung wird *SQL Server 2019 Developer edition*[9] zusammen mit *SQL Server Management Studio*[10] verwendet.
4. Als Hilfskomponente für das System und im Rahmen einer zukünftigen Implementierung des Registry-orientierten Testprozesses kann die *BaSys* Registry-Komponente verwendet werden.
5. Zur Kommunikation zwischen der Testlösung und den aktiven Verwaltungsschalen ist die BaSyx AAS REST API vorgesehen.
6. Als Möglichkeit zur Distribution der Softwarearchitektur ist die Containerisierungsplattform *Docker*[11] vorgesehen.

Architektur:

1. Die Erstellung der Datenbank-basierte Ticket-Repository soll zum Zweck der höheren Wartbarkeit des Gesamtsystems auf Basis der *Code-First* Methode stattfinden und von der konkreten Datenbank unabhängig sein. Die Datenbank soll durch eine Änderung des entsprechenden Treibers problemlos ausgetauscht werden können.
2. Die Kommunikation zwischen der oberen Softwareschicht, also dem sog. Business-Layer, und dem Data-Layer, also der Datenbank, soll auf Basis der *Repository-Pattern*[12] realisiert werden, um eine höhere Wartbarkeit zu gewährleisten.
3. Die Erstellung von Test-Tickets im Kontext der Softwareapplikation soll auf Basis der *Builder-Pattern*[13] stattfinden.
4. Die lose Kopplung der einzelnen Softwaremodule soll durch den Einsatz des *Dependency-Injection*[14] Prinzips garantiert werden.
5. Die Testlösung muss mit Test-Rezepten interagieren, deren Deskriptoren als JSON oder XML-Dateien zur Verfügung gestellt werden.

[8] `https://github.com/FraunhoferIOSB/FAAAST-Service`
[9] `https://www.microsoft.com/en-us/sql-server/sql-server-downloads`
[10] `https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15`
[11] `https://www.docker.com/`
[12] `https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design`
[13] `https://refactoring.guru/design-patterns/builder`
[14] `https://stackoverflow.com/questions/130794/what-is-dependency-injection`

Testobjekte:

1. Als Testobjekte für das Prototyp können sowohl die von der IDTA bereitgestellten als auch im Rahmen der Forschungsgruppe entwickelten Verwaltungsschalen verwendet werden. Eine zusätzliche Erstellung von Testobjekten weicht vom Scope der Arbeit ab. Als Voraussetzung für die Bearbeitung der Themenstellung gilt also die Existenz entsprechender Testobjekte.
2. Als geeignetes Kommunikationsprotokoll zwischen dem Asset und dem digitalen Vertreter ist im Rahmen der Konzeptionsphase OPC UA vorgesehen. Die Architektur soll unabhängig von der technischen Implementierung dieses Kommunikationskanals erstellt werden.

Abgrenzungskriterien:

1. Das System wird für Testobjekte mit nicht-verteilten Verwaltungsschalen und Submodellen entwickelt.
2. Die Implementierung der Schnittstelle zwischen dem Demonstrator und dem Asset ist kein Gegenstand des praktischen Anteils dieser Abschlussarbeit. Als Voraussetzung für die technische Implementierung dieses Kommunikationskanals im Rahmen der Softwarearchitektur des Demonstrators dient die Veröffentlichung des *Asset Interface Description*-Submodells.
3. Eine dynamische Redefinition der Stufen der Test-Pipeline seitens des Anwenders ist im Rahmen der Arbeit nicht vorgesehen.
4. Die Implementierung einer Softwareebene zur Abstraktion der Testlösung vom BaSyx .NET SDK ist im Rahmen der Arbeit nicht vorgesehen.
5. Als Schwerpunkt des praktischen Anteils der Arbeit gilt die Implementierung der Test-Infrastruktur und die Umsetzung der Strukturtests. Die dynamische Validierung von Zustandsmaschinen gilt ebenfalls als Ziel der Arbeit. Die Durchführung von Verhaltenstests ist zusammen mit der virtuellen Inbetriebnahme ein Gegenstand weiterer Arbeiten. Im Rahmen der Untersuchung werden jedoch diverse Ansätze diskutiert, welche als Grundlage für die praktische Umsetzung dieser zwei Pipeline-Stufen angesehen werden.

Applikation (Höherwertige Funktionalität)

Nachfolgend werden die höherwertigen Anforderungen aufgelistet, welche bei einer vollständigen Implementierung der vorgestellten Testinfrastrukturen zu erfüllen wären. Diese wurden bis jetzt implizit dargelegt.

4.2.3.1 Das System muss dem Anwender die Möglichkeit bieten, die Adresse der Registry-Komponente einzugeben.

Die Eingabe findet über die REST API bzw über die Web UI statt und hat den Vorteil, dass die Spezifikation der Registry zur Laufzeit der Testlösung realisiert werden kann. Eine einmalige Spezifikation der Adresse ist vor dem Starten des Orchestrators, mittels einer Konfigurationsdatei, ebenfalls realisierbar. Der Nachteil dieses Ansatzes besteht darin, dass die Durchführung eines Testprozesses für eine zweite Registry-Komponente einen Neustart und eine Rekonfiguration der Testlösung erfordert.

Abgeleitet von: 4.2.1.9

4.2.3.2 Falls eine gültige Registry-Adresse vom Anwender bereitgestellt worden ist, dann muss die Testlösung dem Anwender die Möglichkeit bieten, einen Registry-orientierten Testprozess zu initiieren.

Hierfür ist die Definition eines Submodells mit dem entsprechenden Verhalten erforderlich. Die Initiierung des Testprozesses ohne Eingabe oder nach Eingabe einer üngültigen Adresse soll zur entsprechenden Benachrichtigung führen.

Abgeleitet von: 4.2.1.8

4.2.3.3 Falls eine Registry-orientierte Testdurchführung vom Anwender initiiert wurde und die VWS des Testobjekts ein Test-Ticket enthält, muss das System fähig sein, dieses aus der VWS auslesen zu können.

Hierfür ist die Spezifikation eines Submodells oder einer Sammlung von Submodell-Elementen erforderlich, welche das entsprechende Ticket repräsentiert.

Abgeleitet von: 4.2.1.8 und 4.2.1.11

4.2.3.4 Nach einer gültigen Konfiguration muss die Testlösung fähig sein, mit Datenbanksystemen zum Zweck der Ticket-orientierten Testdurchführung zu interagieren.

Hierfür ist die Definition eines Submodells erforderlich, das die entsprechenden Operationen zur Interaktion mit dem Tickerverzeichnis bereitstellt. Die Adresse des Datenbanksystems soll als Teil der Konfigurationsdatei des Orchestrators figurieren.

Abgeleitet von: 4.2.1.17, 4.2.2.6, 4.2.2.7

4.2.3.5 Das System muss dem Anwender die Möglichkeit bieten, Test-Tickets hochzuladen.

Abgeleitet von: 4.2.1.17, 4.2.2.6, 4.2.2.7

4.2.3.6 Das System muss fähig sein, jedem hochgeladenen und gültigen Ticket einen global eindeutigen Identifikator zuzuweisen.

4.2.3.7 Das System muss dem Anwender die Möglichkeit bieten, Test-Tickets basierend auf der entsprechenden Identifikationskennzeichnung zu löschen.

Abgeleitet von: 4.2.1.17

4.2.3.8 Das System muss dem Anwender die Möglichkeit bieten, Test-Tickets basierend auf der entsprechenden Identifikationskennzeichnung zu aktualisieren.

Abgeleitet von: 4.2.1.17

4.2.3.9 Das System muss dem Anwender die Möglichkeit bieten, Test-Tickets basierend auf der entsprechenden Identifikationskennzeichnung auszulesen.

Abgeleitet von: 4.2.1.17

4.2.3.10 Das System muss dem Anwender die Möglichkeit bieten, einen Ticket-orientierten Testprozess zu initiieren.

Abgeleitet von: 4.2.1.17

4.2.3.11 Das System muss dem Anwender die Möglichkeit bieten für jedes Testobjekt mit einem validen Test-Rezept eine Testsequenz (Test-Pipeline) durchzuführen.

Die Reihenfolge der im Rahmen der Sequenz durchzuführenden Teststufen kann vom Anwender nicht beeinflusst werden.

Abgeleitet von: 4.2.1.18, 4.2.2.6

4.2.3.12 Das System muss fähig sein, die durchzuführenden Stufen der Test-Pipeline auf Basis des Rezeptes automatisch abzuleiten.

Abgeleitet von: 4.2.1.18

4.2.3.13 Falls das Test-Ticket einen Strukturdeskriptor enthält, dann muss das System fähig sein, Strukturtests auf Basis der Beschreibung durchzuführen.

Hierfür ist die Spezifikation eines geeigneten Formats zur Beschreibung des Soll-Stands der VWS-Struktur erforderlich. Dieser Testvorgang ist nicht optional und für jedes Testobjekt relevant.

Abgeleitet von: 4.2.1.19

4.2.3.14 Falls vom Testobjekt eine Zustandsmaschine repräsentiert wird, muss das System fähig sein, diese auf Basis einer Beschreibung des Verhaltens des Zustandsautomats durch die Initiierung von Transitionen und Auswertung von Zuständen zu verifizieren.

Abgeleitet von: 4.2.1.20

4.2.3.15 Falls dem Testobjekt eine maschinenlesbare Verhaltensbeschreibung zugeordnet ist, welche die zu verfizierende Operationen, ihre Argumente, Testdaten und erwartete Ergebnisse beschreibt, muss das System fähig sein, die beschriebene Operationen mittels Unit- bzw. Integrationstests zu verifizieren.

Im Kontext der Verwaltungsschale werden die Operationen als die gängigste Form zur Implementierung von Verhalten angesehen. Diese sind mit den Funktionen bzw. Methoden in den prozeduralen und objektorientierten Sprachen vergleichbar und charakterisieren sich durch Eingags- und Ausgangsvariablen. Ihre Verifikation kann also auf Basis von Unit- bzw. Integrationstests stattfinden.

Abgeleitet von: 4.2.1.21

4.2.3.16 Falls dem Testobjekt ein Simulationskontext zugeordnet ist, muss das System fähig sein, eine virtuelle Inbetriebnahme durchzuführen.

Abgeleitet von: 4.2.1.22

4.2.3.17 Im Laufe der Test-Pipeline muss die Lösung Test-Reports generieren.

Die Reports können in Form von JSON-, XML- oder HTML-Dateien erstellt werden.

Abgeleitet von: 4.2.1.25

4.2.3.18 Falls vom Anwender eine Ticket-orientierte Testdurchführung initiiert wurde, dann kann das Test-Report für das entsprechende Test-Objekt als Teil des zugeordneten Tickets in der Repository gespeichert werden.

Abgeleitet von: Abb. 4.7

4.2.3.19 Falls vom Anwender eine Registry-orientierte Testdurchführung initiiert wurde, dann kann das Test-Report für das entsprechende Test-Objekt als Teil der VWS hochgeladen werden.

Abgeleitet von: Abb. 4.5

4.2.3.20 Das System kann jede erfolgreich getestete VWS mit einem Tag markieren.

Abgeleitet von: 4.2.1.23

4.2.3.21 Falls mehrere aktive Test-Objekte registriert wurden bzw. Test-Tickets im Verzeichnis vorhanden sind und der Anwender einen Testprozess für alle Objekte initiiert hat, muss das System fähig sein, eine Kommunikationsbeziehung mit dem ersten Testobjekt aufzubauen, die Testsequenz durchzuführen, die Verbindung zu schließen und diesen Vorgang für jedes Testobjekt durchzuführen.

Abgeleitet von: 4.2.1.8

4.2.3.22 Falls Test-Reports im Rahmen einer Testdurchführung erzeugt wurden, muss das System fähig sein, diese in einer geeigneten Form dem Anwender bereitstellen zu können.

Werkzeuge zur Erstellung von Test-reports müssen als Teil der Softwarearchitektur des Systems berücksichtigt werden. Idealerweise kann der Orchestrator Test-Reports als HTML-Files dem Anwender mit Visualsierung der Testfällen bereitstellen. Es besteht das Risiko, dass eine Generierung von HTML-Reports zu einer zeitlichen Ineffizienz führt. Als Alternative gilt die Verwendung von XML oder JSON.

Abgeleitet von: 4.2.1.25

4.2.4 Primäre APIs

Die Testplattform setzt sich aus einer Softwarekomponente als Asset und einer Verwaltungsschale zusammen. Der folgende Abschnitt ermöglicht einen groben Überblick über die synthetisierten Softwaremodule der Testautomatisierungslösung. Es werden ihre Zwecke, primäre Funktionalitäten und Charakteristiken diskutiert. Als Kernmodule gelten das Ticketverzeichnis und die Testausführungskomponente (nachfolgend auch als Test-Orchestrator bzw. Test-Koordinator bezeichnet). Ein weiteres wichtiges Modul dient zur Überführung von Test-Tickets oder registrierten Verwaltungsschalen in Testobjekten. Des Weiteren wird im Rahmen des Abschnittes eine Komponente vorgestellt, durch welche die eigentliche Durchführung der Test-Pipeline ermöglicht wird.

Als Nutzer der APIs können unterschiedliche Module dienen. Der Anwender des Ticket-Verzeichnisses und der Orchestrierungskomponente sind vom Design des Gesamtsystems abhängig. Es wird zwischen drei Szenarien unterschieden. Die Hauptkomponenten können

- durch die API der Test-Orchestrierungsklasse,
- direkt von der VWS der Testlösung oder
- von spezialisierten Service-Module, derer Funktionalitäten die Operationen des entsprechenden Submodells abbilden,

zugegriffen werden.

Im Rahmen des ersten Szenario werden alle primären Klasseninstanzen als Argumente bei der Instanziierung einer Hauptklasse übergeben. Als Hauptklasse könnte das Modul `TestOrchestrator` dienen, siehe Abschnitt 4.2.5.32. In diesem Szenario kann beispielsweise eine konkrete Instanz der `TicketRepository` dem Orchestrator nach dem *Dependency Inversion* Prinzip, also als Konstruktor-Argument, zur Verfügung gestellt werden. Dadurch erfolgt der Zugriff zum Verzeichnis auf Basis der Funktionalitäten der Hauptklasse. In diesem Sinne wird die gesamte Systemfunktionalität von diesem Kernmodul gekapselt, siehe Abb. 4.9. Seine Instanziierung als Attribut der VWS-Instanz würde einen Zugriff zu allen von den untergeordneten Komponenten bereitgestellten Funktionalitäten bei der Implementierung der Submodelle ermöglichen.

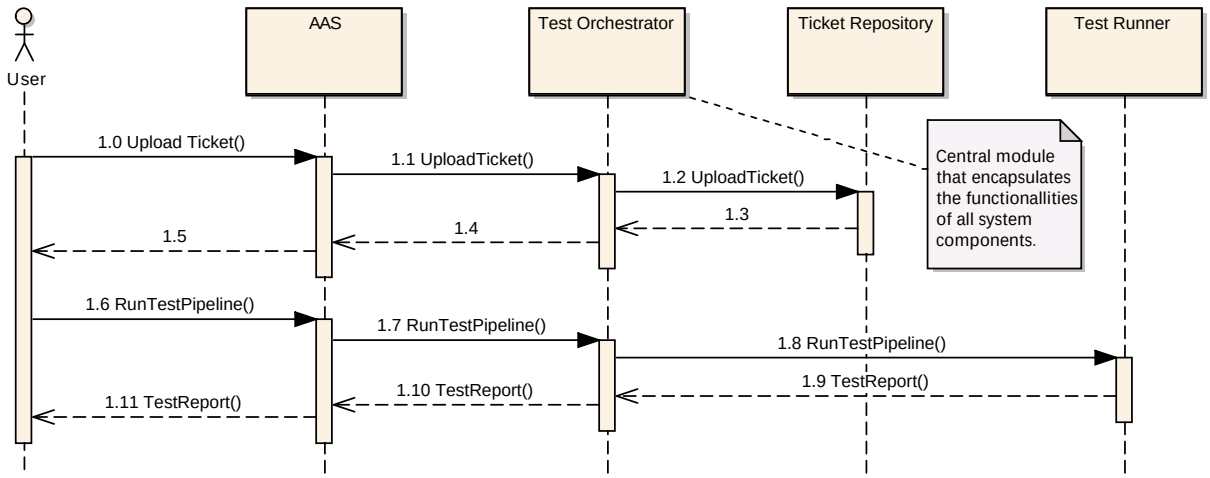


Abbildung 4.9: Szenario 1: Kapselung der Funktionalitäten durch eine Hauptklasse

Diese Variante dient der Komplexitätsreduktion auf die höchste Ebene des Gesamtsystems, da die Verwaltungsschale der Testlösung nur eine einzige Klasseninstanz, also diese der Klasse `TestOrchestrator`, verwalten muss. Auf der anderen Seite ist somit die Hauptklasse für die Verwaltung mehrerer Instanzen mit unterschiedlichen Zwecken verantwortlich, was als Verstoß gegen das *Single Responsibility* Prinzip gilt.

Als Anwender der Funktionalitäten des Ticket-Verzeichnisses lässt sich auch die Verwaltungsschale der Testlösung betrachten. Im Rahmen des zweiten Szenario werden alle Kernkomponenten als Attribute der VWS instanziiert. Im Rahmen des Beispiels entspricht die erste Instanz der Klasse `TestOrchestrator` und charakterisiert sich durch einen Lesezugriff zum Ticketverzeichnis. Die nächste Instanz entspricht der Klasse `TicketRepository`. Über das entsprechende Submodell bekommt den Nutzer einen Schreibzugriff zum Hochladen, Aktualisieren und Löschen von Test-Rezepten.

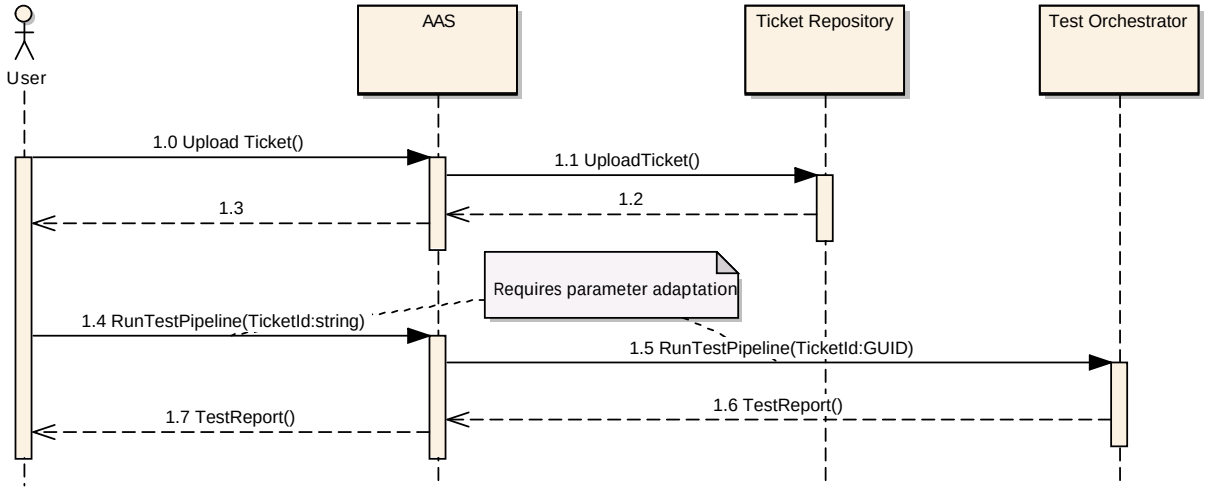


Abbildung 4.10: Szenario 2: Direkte Instanziierung der Kernkomponenten

Dieser Ansatz führt zu einer Steigung der Komplexität der obersten Softwareebene der Architektur, da die Parameter der Submodell-Operationen an die Argumente der Instanzmethoden adaptiert werden müssen.

Das dritte Szenario charakterisiert sich durch die Definition von Service-Klassen. Jede Service-Klasse gilt als Anwender einer primären, also für den Nutzer relevanten, Instanz. Die primären Instanzen werden nach dem *Dependency Inversion* Prinzip der Service-Klassen bereitgestellt. Die Funktionalitäten des entsprechenden Dienstes rufen die entsprechenden Instanzmethoden nach einer Adaptierung der Argumente auf. Die Instanz der Verwaltungsschale bekommt als Attribute die spezifizierten Dienste und dient als Anwender ihrer Funktionalitäten.

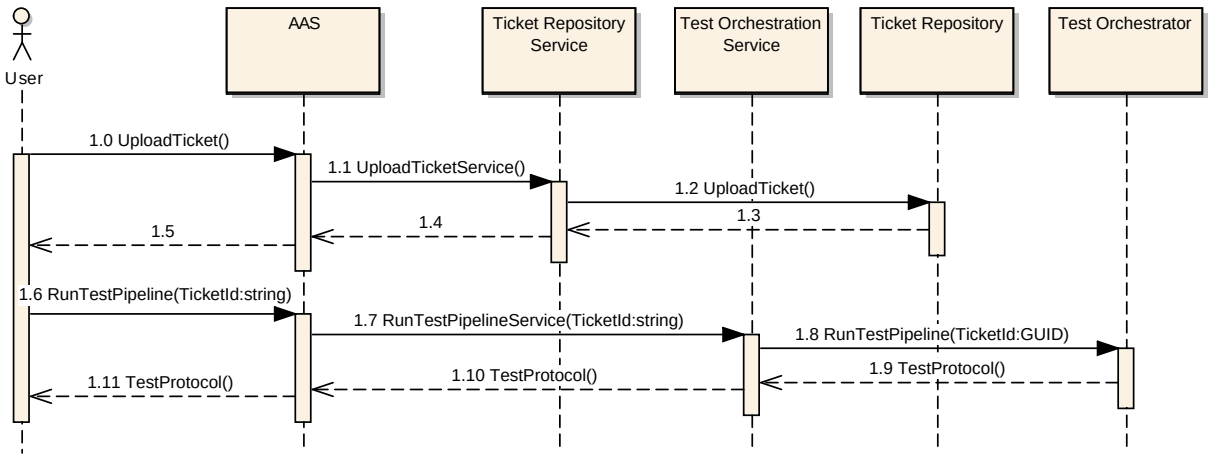


Abbildung 4.11: Szenario 3: Bereitstellung der Dienste durch Service-Klassen

Aufgrund der klaren Zuordnung der Verantwortlichkeiten und der daraus resultierenden gleichmäßigen Verteilung der Komplexität der Komponenten der obersten Softwareschichten, lässt sich dieser Ansatz als die beste Alternative bezeichnen.

Ticket Repository

Wie bereits erläutert wurde, gilt das Ticket-Verzeichnis als allgemeine Auskunftsstelle für Test-Rezepten und bietet die Schnittstelle für eine Ticket-Verwaltung an. Die konkrete Implementierung des Ticket-Verzeichnisses kann in Abhängigkeit vom Anwendungsfall und den Kundenpräferenzen variieren. Durch die GitHub REST API[15] bietet sich beispielsweise die Definition dieses Versionverwaltungssystems als konkrete Instanz einer Ticket-Repository. Wie im Rahmen des Grobkonzeptes erwähnt, lässt sich für die konkrete Implementierung der Komponente weiterhin auch eine Datenbank verwenden. Das folgende Anwendungsfalldiagramm illustriert die Dienste, welche mit dem Ticketverzeichnis assoziiert werden.

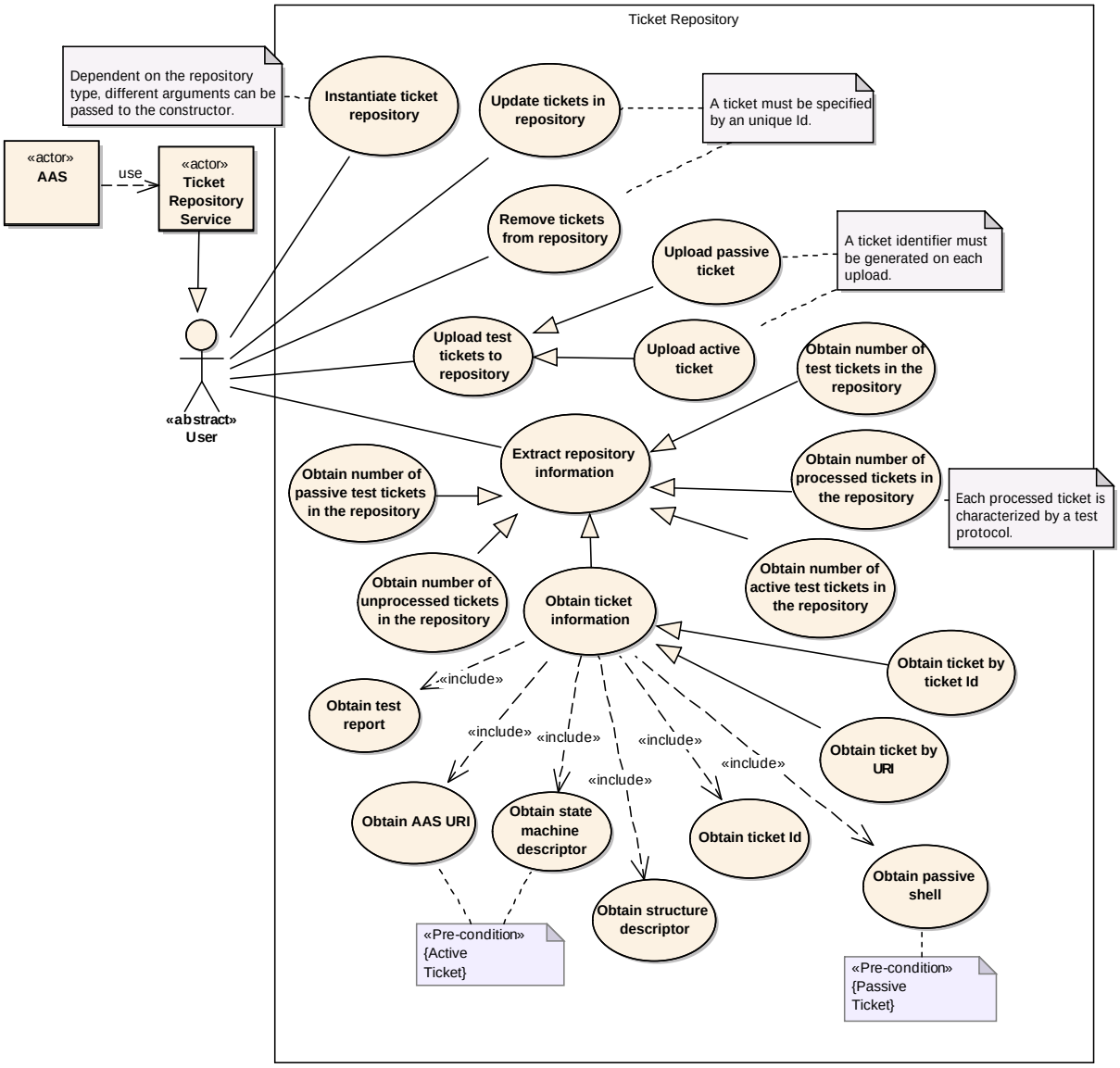


Abbildung 4.12: Ticket-Repository Dienste

[15] `https://docs.github.com/en/rest`

4.2.4.1 AAS
Der Zugriff zum Ticketverzeichnis erfolgt durch die Verwaltungsschale der Testlösung.

4.2.4.2 User
Der Anwender der *Ticket-Repository API* kann jede übergeordnete Softwarekomponente sein, welche die zur Verfügung stehenden Funktionalitäten über eine geeignete Schnittstelle der VWS-Komponente bereitstellt, siehe 4.2.4.3 und 4.11. Anhand des Diagramms ist als Anwender des Verzeichnisses ein *Repository-Service* vorgesehen.

4.2.4.3 Ticket-Repository Service
Die Dienste des Moduls *Ticket-Repository* werden durch den Einsatz der entsprechenden Service-Klasse als Teil eines VWS-Submodells über das Web dem Anwender zur Verfügung gestellt. Die API könnte zum Zweck der Erweiterung des Orchestrators um weitere Funktionalitäten und der Entwicklung weiterer Lösungen von Personen mit Fachkenntneissen verwendet werden.

4.2.4.4 Upload test tickets to repository
Die Instanz bietet dem Anwender die Möglichkeit Test-Tickets zum Ticketverzeichnis hinzuzufügen. Es wird zwischen aktiven und passiven Tickets unterschieden. Das Hochladen eines aktiven Testrezeptes unterscheidet sich durch die Angabe weiterer Deskriptoren.

4.2.4.5 Upload active ticket
Die Instanz bietet dem Anwender die Möglichkeit, Test-Tickets für aktive Verwaltungsschalen zum Ticketverzeichnis hinzuzufügen. Das Hochladen erfolgt durch Eingabe der notwendigen Deskriptoren.

4.2.4.6 Upload passive ticket
Die Instanz bietet dem Anwender die Möglichkeit, Test-Tickets für passive Verwaltungsschalen zum Ticketverzeichnis hinzuzufügen. Im Rahmen des Uploads ist die AAS-Datei, also das Testobjekt, vom Anwender bereitzustellen.

4.2.4.7 Extract repository information
Das System bietet dem Anwender die Möglichkeit, mit dem Ticketverzeichnis assoziierbare Informationen abzufragen.

4.2.4.8 Obtain number of active test tickets in the repository
Die Instanz kann die Anzahl der Tickets abfragen, welche einer aktiven Verwaltungsschale zugewiesen sind.

4.2.4.9 Obtain number of passive test tickets in the repository
Die Instanz kann die Anzahl der Tickets abfragen, welche einer passiven Verwaltungsschale zugewiesen sind.

4.2.4.10 Obtain number of test tickets in the repository
Die Instanz kann die Anzahl aller im Verzeichnis registrierten Tickets abfragen.

4.2.4.11 Obtain ticket information
Die Instanz bietet dem Anwender die Möglichkeit Ticket-Informationen abzufragen. Als Ticket-Informationen gelten Deskriptoren, Typ des Testrezeptes, Testberichte u.a.

4.2.4.12 Remove tickets from repository
Die Instanz kann Tickets aus dem Verzeichnis löschen.

4.2.4.13 Update tickets in repository
Die Instanz kann Tickets im Verzeichnis aktualisieren. Als Eingangsinformation hier gilt die Identifikationskennzeichnung des Tickets.

Test Orchestrator API

Das folgende Use-Case Diagramm definiert die Dienste des Moduls `TestOrchestrator` und veranschaulicht seine Rolle im Kontext der Softwarearchitektur.

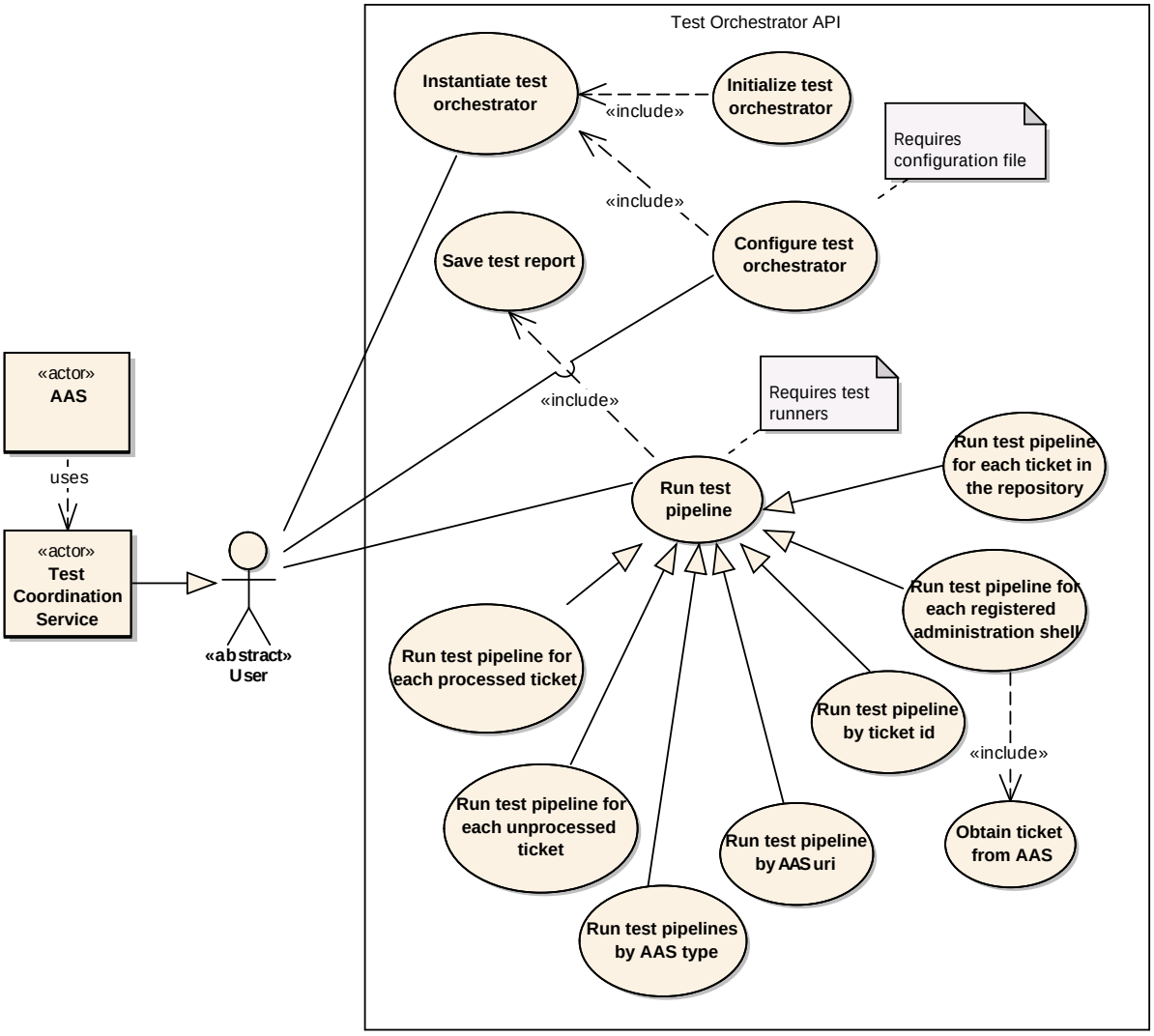


Abbildung 4.13: Test-Orchestrator Dienste

4.2.4.1 AAS

Der Zugriff zu den Diensten der Testplattform erfolgt durch die Verwaltungsschale der Testlösung.

4.2.4.2 User

Der Anwender der *Test Orchestrator API* kann jede übergeordnete Softwarekomponente sein, welche die zur Verfügung stehende Funktionalitäten über eine geeignete Schnittstelle der VWS-Komponente bereitstellt, siehe 4.2.4.3 und 4.11. Anhand des Diagramms ist als Anwender der Orchestrierungsklasse ein Koordinierungsdienst vorgesehen.

4.2.4.3 Test Coordination Service

Die Verwaltungsschale dient als Anwender der Funktionalitäten der Klasse `TestCoordinationService`. Wie bereits erwähnt wurde, besteht die Aufgabe des Dienstes in der Adaptierung der vom Submodell bereitgestellten Operationen an die internen Funktionalitäten des Softwaresystems.

4.2.4.4 Instantiate test orchestrator
Die API muss dem Anwender die Möglichkeit bieten, die Klasse `TestOrchestrator` zu instanziieren. Von der Klasse ist eine Interface[16] zu implementieren, welches die Funktionalitäten jeder konkreten Instanz des Moduls spezifiziert und als einheitliche Schnittstelle für Erweiterungen gilt.

4.2.4.5 Configure test orchestrator
Zum Zweck der Konfiguration des Orchestrators kann eine Schnittstelle implementiert werden, welche bestimmte Daten aus einer Konfigurationsdatei ausliest. Die Konfigurationsdatei kann entweder als JSON oder XML gespeichert werden, muss von der Softwarekomponente deserialisiert werden können und kann beim Anlegen der Instanz dem Konstruktor übergeben werden.

4.2.4.6 Run test pipeline
Die Instanz muss dem Anwender die Möglichkeit bieten, eine oder mehrere Test-Pipelines durchzuführen. Da die Tests durch den Einsatz unterschiedlicher .NET Test-Frameworks wie z.B. *NUnit* oder *XUnit* implementiert werden können, ist die entsprechende Realisierung durch Funktionsargumente vom API-Client zu spezifizieren. Somit kann die Abhängigkeit des Orchestrators vom konkreten Test-Framework eliminiert werden.

4.2.4.7 Run test pipeline by AAS URI
Die Instanz kann eine Test-Pipeline auf Basis der URI-Adresse des entsprechenden Testobjekts durchführen.

4.2.4.8 Run test pipeline by ticket ID
Die Instanz kann eine Test-Pipeline auf Basis der Identifikationskennzeichnung des Tickets durchführen. Die Kennzeichnung wird automatisch generiert und nach jedem erfolgreichen Ticket-Upload dem Anwender bereitgestellt.

4.2.4.9 Run test pipeline for each registered administration shell
Das System kann eine Test-Pipeline für jede registrierte I4.0 Komponente durchführen. Diese Funktionalität setzt die Verfügbarkeit der Registry-Komponente voraus.

4.2.4.10 Obtain ticket from AAS
Das System kann das Test-Rezept aus der VWS des Testobjekts auslesen können.

4.2.4.11 Run test pipeline for each ticket in the repository
Die Instanz kann eine Test-Pipeline für jedes Testobjekt durchführen, welches einem Ticket zugeordnet wird. Diese Funktionalität setzt einen Zugriff zum Ticketverzeichnis voraus.

4.2.4.12 Run test pipelines by AAS type
Die Instanz kann die Tickets im Ticketverzeichnis nach Typ der Verwaltungsschale filtern und für jede Klasse eine Test-Pipeline durchführen.

4.2.4.13 Run test pipeline for each unprocessed ticket
Die Instanz kann eine Testsequenz für alle nicht-verifizierten Testobjekte durchführen, die durch ein Ticket im Ticketverzeichnis repräsentiert werden.

4.2.4.14 Run test pipeline for each processed ticket
Die Instanz kann eine Testsequenz für alle verifizierten Testobjekte durchführen, die durch ein Ticket im Ticketverzeichnis repräsentiert werden.

4.2.4.15 Save test report
Die Instanz kann Testberichte erstellen und speichern.

[16] der Begriff wird in Zusammenhang mit der objektorientierten Programmierung verwendet

Test-Object Provider API

In Zusammenhang mit dem Ticket-orientierten Testprozess besteht die Aufgabe dieser Komponente in der Konstruktion von Testobjekten[17] basierend auf Ticket-Informationen. Aus diesem Grund wird beim Instanziieren einer konkreten `TestObjectProvider`-Klasse eine Instanz der `TicketRepository` als Konstruktor-Argument übergeben. Dadurch wird den lesenden Zugriff zum Ticketverzeichnis umgesetzt. Im Kontext der Registry-basierten Testung erstellt das Modul ein Testobjekt aus jeder registrierten VWS-Instanz. Die API ermöglicht also die Konstruktion von Testobjekten und wird zum Zweck der beiden Teststrategien vom für die Testdurchführung verantwortlichen Modul, also den `TestOrchestrator`, benutzt.

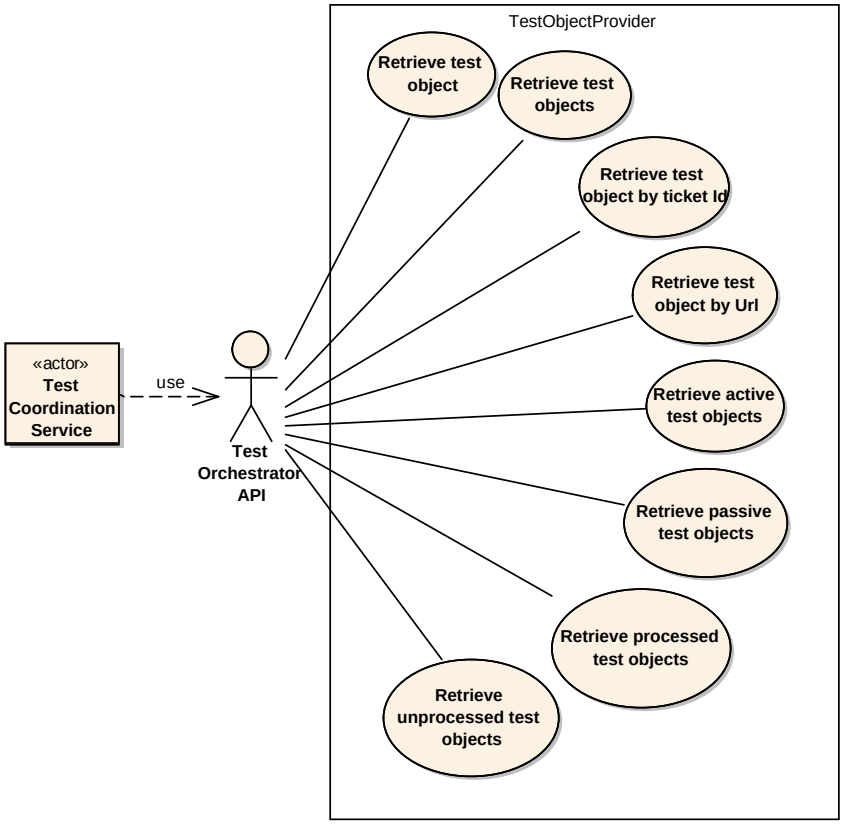


Abbildung 4.14: Test-Object Provider Dienste

4.2.4.1 Test Coordination Service

Der Dienst für die Testdurchführung basiert auf die API der Klasse `TestOrchestrator`. Für die Funktionsweise des Orchestrators ist eine konkrete Instanz des Providers erforderlich.

4.2.4.2 Test Orchestrator API

Zum Zweck der Durchführung einer Testsequenz ruft die *Test Orchestrator API* die Methoden des Providers auf.

4.2.4.3 Retrieve test object

Die Methode dient der Konstruktion eines Testobjekts.

4.2.4.4 Retrieve test object by URI

Die Funktion liefert ein aktives Testobjekt auf Basis einer URI-Adresse.

[17] Instanzen der Klasse `TestObject`, siehe Abschnitt 4.2.5.19

4.2.4.5 Retrieve test object by ticket Id
Die Funktion liefert ein Testobjekt auf Basis der eingegebenen Ticket Id.

4.2.4.6 Retrieve active test objects
Die Funktion liefert eine Liste aus aktiven Testobjekten.

4.2.4.7 Retrieve passive test objects
Die Funktion liefert eine Liste aus passiven Testobjekten.

4.2.4.8 Retrieve test objects
Die Funktion liefert eine Liste aus allen registrierten bzw. durch ein Ticket repräsentierten Testobjekten.

4.2.4.9 Retrieve processed test objects
Die Funktion liefert eine Liste aus allen Testobjekten, für die eine Testsequenz ausgeführt worden ist.

4.2.4.10 Retrieve unprocessed test objects
Die Funktion liefert eine Liste aus allen Testobjekten, für die keine Testsequenz ausgeführt worden ist.

Test-Runner API

Die Test Runner API definiert die Methoden zur Ausführung der vorimplementierten Test-Klassen.

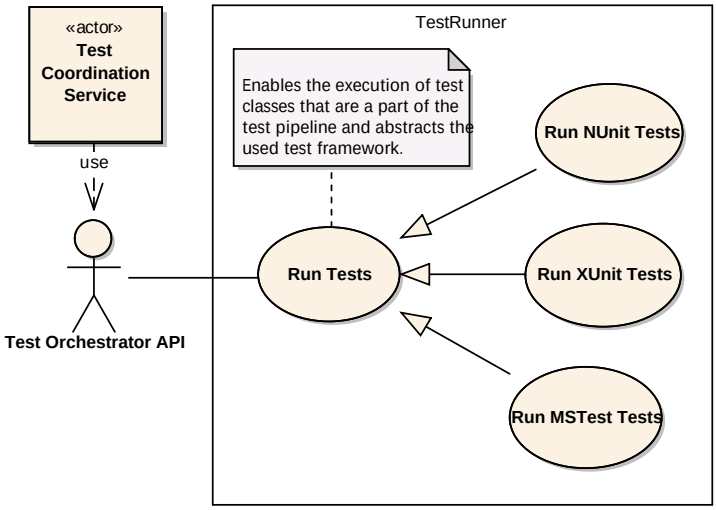


Abbildung 4.15: Test-Runner Dienste

4.2.4.1 Test Orchestrator API
Die Orchestrator API muss zum Zweck der Initiierung einer Test-Pipeline eine Komponente verwenden, die die Ausführung der entsprechenden Test-Klassen startet.

4.2.4.2 Run Tests
Die Funktion ermöglicht die Ausführung ausgewählter und vorimplementierter Testklassen. Diese wird angesichts der Ausführung der Test-Pipeline von der Orchestrator API benutzt. Zum Zweck der Kompatibilität des Gesamtsystems mit den gängigen .NET Test-Frameworks kann die Methode für jede Testumgebung entsprechend implementiert und im Kontext der Orchestrator API durch eine Schnittstelle abstrahiert werden. Dies würde die Durchführung der Test-Pipeline unabhängig von der im Rahmen der Erstellung der Testklassen eingesetzten Technologie garantieren.

4.2.5 Komponenten

In diesem Abschnitt werden die Module abgeleitet, die eine Möglichkeit zur softwaretechnischen Implementierung der Testlösung im Kontext eines MVP darstellen. Diese werden anhand der bereits beschriebenen Systemcharakteristiken und Spezifikationen nach dem *Bottom-Up* Prinzip synthetisiert.

Aus dem Inhalt des Kapitels lassen sich folgende Module ableiten:

- Test-Ticket Descriptors
- Test-Ticket
- Test-Ticket Repository
- Test-Object
- Test-Object Provider
- Test-Runner
- Test Classes (vorimplementierte Pipeline-Stufen)
- Test Orchestrator/Koordinator
- Ticket Repository Service
- Test Orchestration/Coordination Service
- Test Management Shell (AAS)

In den folgenden Abschnitten wird jedes Modul detailliert beschrieben.

Ticket Deskriptoren

Wie bereits erwähnt wurde, stellt ein Ticket im Kontext der Testlösung die Eingangsinformationen zur Durchführung der gesamten Test-Pipeline dar. Laut Definition setzt sich diese aus den folgenden Teststufen zusammen.

- Kommunikationstests
- Strukturtests
- zustandsbezogene Tests
- Verhaltenstests sowie
- VIBN

Die Durchführung von Kommunikationstests als Bestandteil der Pipeline kann implizit, also als Teil der *Arrange*-Phase einer anderen Test-Stufe, erfolgen. Jeder Stufe des Testprozesses kann ein Deskriptor zugewiesen werden, der eine Erwartungshaltung spezifiziert und als Teil des Tickets verfügbar ist. Wie bereits spezifiziert wurde, muss das System dem Anwender die Möglichkeit bieten Test-Tickets dem Orchestrator bereitzustellen. Der Nutzer muss also die entsprechenden Deskriptoren in einer geeigneten Form der Testlösung zur Verfügung stellen können. Hierfür wurden im Rahmen der Konzepterstellung zwei Möglichkeiten identifiziert.

(a) Eine REST-Schnittstelle zum Überschreiben oder Ersetzen bereits existierender Dateien einer VWS ist als Teil des allgemeinen Submodell-Controllers[18] im Rahmen des BaSyx .NET SDK implementiert worden. Die BaSyx AAS REST API bietet jedoch keine Funktionalitäten zur dynamischen Speicherung von Dateien als Elemente einer Verwaltungsschale. Unter dynamische Speicherung wird das Hochladen einer bei der Erstellung der VWS unspezifizierten Anzahl von Dateien in Laufzeit verstanden. Aufgrund der begrenzten Funktionalitäten der BaSyx AAS REST API, stellt die erste Möglichkeit die Implementierung einer weiteren REST-Funktionalität dar, welche einen direkten Datei-Upload aus beliebigen Endpunkten ermöglicht.

[18] `https://github.com/eclipse-basyx/basyx-dotnet-components/blob/main/BaSyx.API.Http.Controllers/AssetAdministrationShell/SubmodelController.cs`

Als Vorteil dieses Ansatzes gilt die nutzerfreundliche Interaktion des menschlichen Nutzers mit der Testlösung. Bei einem Ticket-Upload können die Deskriptoren entweder durch die Spezifikation des lokalen Pfads oder mittels einer Upload-Taste ausgewählt werden.

Ein bedeutender Nachteil dieser Variante ist, dass eine solche Erweiterung einer Abweichung von der standardisierten Kommunikationsschnittstelle entspricht. Aus diesem Grund gilt die zweite Alternative als eine bessere Variante.

(b) Der Inhalt der entsprechenden XML- oder JSON Deskriptor-Datei wird einer geeigneten Orchestrator-Operation als Zeichenkette übergeben.

Der Nachteil dieses Ansatzes besteht in der aufwändigen Interaktion des menschlichen Nutzers mit der Testlösung. Der Aufwand wird durch das Kopieren und Einfügen von langen XML Bäumen und JSON Objekten dargestellt. Weiterhin muss die Testlösung die entsprechenden Deskriptor-orientierten Validierungsmechanismen auf Basis der entsprechenden Schemas implementieren.

Als Vorteil gilt die Beibehaltung der standardisierten REST API.

Im Folgenden wird eine Möglichkeit zur softwaretechnischen Umsetzung der Ticket-Deskriptoren beschrieben. Zum Zweck der Erweiterbarkeit kann jeder Deskriptor durch das entsprechende Interface abstrahiert werden. Auf diese Weise kann jede Klasse, welche einer nachfolgend dargestellten Schnittstelle gerecht wird, als gültige Ersatzmöglichkeit des entsprechenden Deskriptors im Kontext der Applikation betrachtet bzw. ohne Änderungen im System integriert werden.

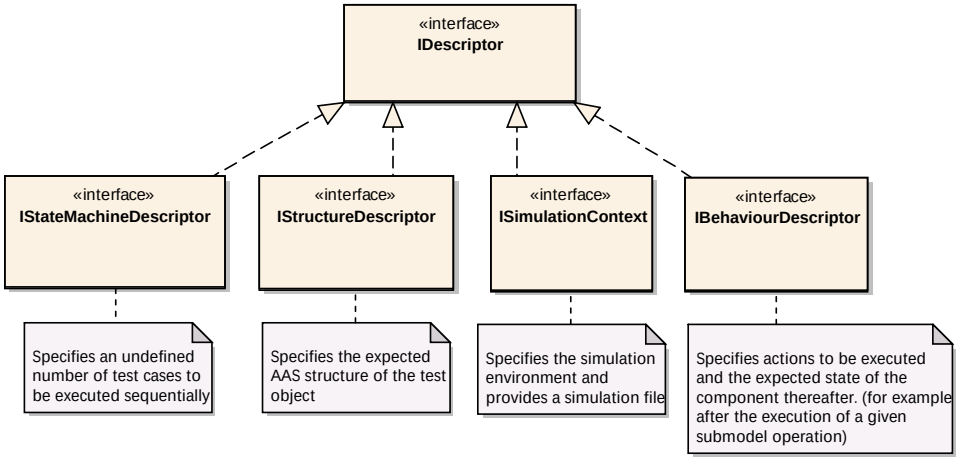


Abbildung 4.16: Klassendiagramm Ticket-Deskriptoren

4.2.5.1 IDescriptor

Dieses Interface ist zum Zweck der Erweiterbarkeit aller Ticket-Deskriptoren entstanden und dient als gemeinsame Programmierstelle zur Spezifikation ihrer gemeinsamen Charakteristiken. Im Rahmen der prototypischen Realisierung findet die Schnittstelle keine Anwendung. Sie gilt als Super-Klasse aller Deskriptor-Interfaces und wurde zum Zweck der Implementierung eines Deskriptor-Verwaltungssystems als Gegenstand zukünftiger Arbeiten implementiert.

4.2.5.2 IAssetAdministrationShell

Das *Eclipse BaSyx* Interface `IAssetAdministrationShell` gehört zum BaSyx .NET SDK und repräsentiert die Eigenschaften, durch welche jede Verwaltungsschale einer Industrie 4.0 Komponente charakterisiert wird. Von besonderer Bedeutung für die Verifikation der VWS-Struktur ist das zusammengesetzte Instanzattribut `Submodels`, welches Informationen über alle Submodelle und ihre Elemente enthält.

4.2.5.3 IStructureDescriptor

Die Strukturbeschreibung beinhaltet alle Informationen zur Struktur einer Verwaltungsschale und kann somit als eine Sammlung von statischen Daten angesehen werden. Der Deskriptor muss die Identifikationsdaten aller Submodelle, Submodell-Elemente und Kollektionen in einer hierarchischen Form, also zusammen mit deren Beziehungen zueinander, als Grundlage für die Strukturtests bereitstellen können.

Als geeignete Formate zur Spezifikation der erwarteten Struktur werden im Kontext des Prototyps XML und JSON angesehen. Für die eigentliche Semantik der Strukturbeschreibung eignen sich die von der IDTA bereitgestellten Schemas[19]. Die Kernelementen der Schemen sind in Listing 4.1 und 4.2 aufgelistet. Da diese zur Erstellung von passiven Verwaltungsschalen eingesetzt werden, sind die Schemas für Strukturspezifikationen sehr gut geeignet.

Für die Strukturtests kann entweder das ganze Schema oder nur den Anteil übernommen werden, welcher die Strukturinformationen beschreibt. Es wird zwischen zwei Möglichkeiten unterschieden.

a) Durch die Festlegung des gesamten Metamodells als Grundlage für die Durchführung von Strukturtests entsteht ein Erweiterungspotenzial für die gesamte Testplattform. Der Einsatz des gesamten Schemas bietet der Testlösung die Möglichkeit, nicht nur strukturorientierte Tests durchzuführen sondern alle statischen Daten einer VWS zu verifizieren. Dazu gehören weiterhin Datentypen, aktuelle Werte u.a. Des Weiteren lassen sich durch diesen Ansatz Strukturbeschreibungen direkt mithilfe des *AASX Package Explorer* erstellen. Der Inhalt der erstellten VWS-Datei kann dann dem Orchestrator als Teil des Test-Rezeptes zur Verfügung gestellt werden.

b) Eine weitere Möglichkeit entspricht der Identifikation und Ableitung der notwendigen Strukturinformationen aus der Vorlage. Im Rahmen dieses Ansatzes wird vom Anwender die Strukturspezifikation manuell und in textueller Form, als JSON bzw. XML-Datei und auf Basis der erwähnten Schemas spezifiziert und dem Testbett bereitgestellt. Hier operiert die Testlösung also nicht mit der gesamten Beschreibung einer passiven VWS, sondern mit dem Teil davon, welcher die Submodelle und ihre Strukturen beschreibt.

Es besteht die Frage, wie die Integration der Strukturbeschreibung, also der XML bzw. JSON-Datei, in der .NET Softwareumgebung stattfindet. Hierfür eignet sich eine konkrete Betrachtung der VWS-Beschreibung. Listings 4.1[20] und 4.2[21] beschreiben die Struktur einer XML- bzw. JSON-Datei, durch welche eine passive VWS beschrieben werden kann.

```
1 {
2   "assetAdministrationShells": [],
3   "assets": [],
4   "submodels": [],
5   "conceptDescriptions": []
6 }
```

Listing 4.1: JSON-Objekt einer Typ 1 VWS

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <aas:aasenv ...>
3   <aas:assetAdministrationShells></aas:assetAdministrationShells>
4   <aas:assets></aas:assets>
5   <aas:conceptDescriptions></aas:conceptDescriptions>
6   <aas:submodels></aas:submodels>
7 </aas:aasenv>
```

Listing 4.2: XML-Struktur einer Typ 1 VWS

[19] https://github.com/admin-shell-io/aas-specs
[20] https://github.com/admin-shell-io/aas-specs/tree/master/schemas/json
[21] https://github.com/admin-shell-io/aas-specs/tree/master/schemas/xml

Aus den beiden Listings geht hervor, dass die Strukturinformationen als Array aus JSON bzw. XML-Elementen vom Objekt `submodels` gekapselt werden. Nach einer Ableitung der Klassen aus der Schema-Definition lassen sich die Strukturdaten mittels geeigneter Bibliotheken deserialisieren. Die Ableitung kann entweder manuell oder durch Generatoren wie z.B. das *XML Schema Definition Tool*[22] realisiert werden. Eine bessere Alternative bietet die Nutzung der BaSyx API an. Zum Zweck des Umgangs mit externen Verwaltungsschalen definiert *Eclipse BaSyx* Funktionalitäten[23] zur Serialisierung und Deserialisierung von Verwaltungsschalen.

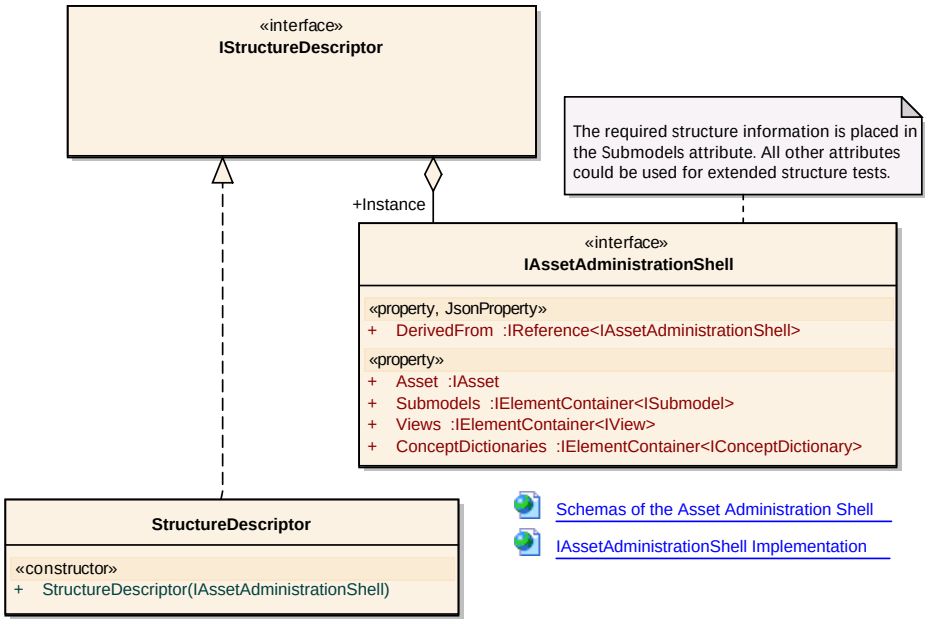


Abbildung 4.17: Klassendiagramm des Strukturdeskriptors

Zum Zweck der Erweiterbarkeit des strukturorientierten Testvorgangs können die Informationen nach der Deserialisierung des Struktur-Deskriptors durch das BaSyx-Interface `IAssetAdministrationShell` repräsentiert werden. Auf diese Weise operiert die Testlösung mit zwei Instanzen dieses Interface. Während die erste Instanz die VWS des eigentlichen Testobjekts darstellt, spezifiziert die Zweite seine Soll-Struktur.

4.2.5.4 StructureDescriptor

Die Klasse implementiert das `IStructureDescriptor`-Interface und enthält die Soll-Struktur des Testobjekts als Instanz des Datentyps `IAssetAdministrationShell`.

4.2.5.5 IStateMachineDescriptor

Zum Zweck der universellen Beschreibung einer Zustandsmaschine kann die *State Chart XML*-Spezifikation[24] in Betracht gezogen werden. Aus der Spezifikation können die Grundkonstrukte eines traditionellen Zustandsautomaten, nämlich die Attribute zur Spezifikation eines Zustands zusammen mit den zugehörigen Transitionen und Zielzustände, übernommen und für die Definition von Testfällen im Kontext der Testplattform eingesetzt werden.

Wie bereits in Abschnitt 2.1.5 diskutiert, besteht die softwareorientierte Verifikationsstrategie eines Zustandsautomaten in der Initiierung von Zustandswechseln und der darauffolgenden Überprüfung, ob der neue Zustand mit der Spezifikation korrespondiert. Der Deskriptor der Führungskomponente kann eine Sequenz von Pfaden abbilden, welche im Rahmen der Test-Pipeline, nach einer potenziellen Durchführung von Zwischentests, sequentiell verifiziert werden und als Testszenarien gelten.

[22] `https://docs.microsoft.com/en-us/dotnet/standard/serialization/xml-schema-definition-tool-xsd-exe`
[23] `https://github.com/eclipse-basyx/basyx-dotnet-sdk/tree/main/BaSyx.Models.Export`
[24] `https://www.w3.org/TR/2015/REC-scxml-20150901/`

Diverse Vorbedingungen für diese Stufe der Test-Pipeline wurden in Abshnitt 4.2.1.20 diskutiert. Um sicherzustellen, dass bei der Definition der Pfaden kein Schreibfehler stattgefunden ist, lassen sich alle möglichen Transitionen und Zustände am Anfang des Deskriptors deklarieren und im Rahmen eines Zwischentests validieren. Die Gültigkeit der spezifizierten VWS-Eigenschaften kann sowohl im Rahmen der Strukturtestung als auch als Teil eines weiteren sekundären Test bei der Validierung des Zustandsautomaten überprüft werden.

Das folgende Listing illustriert eine Möglichkeit zur Spezifikation von zwei Testfällen im Rahmen eines XML-Deskriptors.

```
1  <!-- aborted => stopped -->
2  <path>
3    <properties submodel="SM_Unit1_(SuperiorStateMachine)"
         setter="SMC_Statemachine/Prop_ControlCommand" getter="SMC_Statemachine/Prop_State"/>
4    <state id="Aborted">
5      <transition event="Clear" target="Stopped"/>
6    </state>
7  </path>
8
9  <!-- stopped => aborted -->
10 <path>
11   <properties submodel="SM_Unit1_(SuperiorStateMachine)"
         setter="SMC_Statemachine/Prop_ControlCommand" getter="SMC_Statemachine/Prop_State"/>
12   <state id="Aborted">
13     <transition event="Clear" target="Stopped"/>
14   </state>
15   <state id="Stopped">
16     <transition event="Abort" target="Aborted"/>
17   </state>
18 </path>
```

Listing 4.3: Beispiel für die Spezifikation zustandsorientierter Testfälle

Als Voraussetzung für diesen Ansatz dient die Spezifikation eines Initialzustandes. Auf diese Weise kann jeder Pfad unabhängig von dem vorherigen bzw. nachfolgenden Testfall validiert werden. Dadurch, dass jeder Testpfad die Identifikationskennzeichnung des Submodells und der entsprechenden Properties spezifiziert, lassen sich mehrere und/oder voneinander abhängige Zustandsautomaten verifizieren. Im Rahmen der Konzeptprüfung erfolgt eine Ergänzung des vorgestellten Metamodells. Die konkrete Definition der Testszenarien gehört zu den Aufgaben des Nutzers.

4.2.5.6 StateMachineDescriptor

Die Klasse entspricht der konkreten Instanz der `IStructureDescriptor`-Schnittstelle und repräsentiert die Inhalt des Test-Rezeptes im Kontext der Softwareapplikation. Die Inhalt der Datei, welche die zustandsorientierte Testszenarien beschreibt, wird also durch diese Klasse deserialisiert.

4.2.5.7 ISimulationContext

Wie bereits erwähnt wurde, lässt sich als Deskriptor für die Durchführung einer VIBN das Submodell *Simulation*[25] verwenden. Dem Demonstrator kann also die entsprechende XML bzw. JSON-Datei bereitgestellt werden. Zum Zeitpunkt dieser Arbeit befindet sich das Submodell in Entwicklung. Eine konzeptionelle Beschreibung wie und durch welche Methoden und Werkzeuge, eine virtuelle Inbetriebnahme im Rahmen zukünftiger Arbeiten durchgeführt werden könnte, befindet sich in Abschnitt 6.3.

4.2.5.8 SimulationContext

Die Klasse entspricht der konkreten Instanz der `ISimulationContext`-Schnittstelle und würde als Hilfsmittel für die Deserialisierung des Simulationsrezeptes dienen.

[25] `https://industrialdigitaltwin.org/content-hub/teilmodelle`

4.2.5.9 IBehaviorDescriptor

Die Beschreibung des Verhaltens kann auf Grundlage einer XML- bzw. JSON-Datei erfolgen, welche eine *one-time setup*, eine *one-time teardown* Sektion und mehrere Testszenarien umfasst. Da jedes Szenario durch unterschiedliche Vor- bzw. Nachbedingungen charakterisiert werden kann, was auch als Basis für die Durchführung eines *Feasibility Check* dient, eignet sich die Spezifikation einer *setup* und *teardown* Sektion als Teil seines Metamodells. Die Beschreibung der Testschritte kann durch ein weiteres XML-Element bzw. JSON-Objekt stattfinden. Als Testschritte lassen sich Aktionen und *Assertions* definieren. Während durch die Aktionen aufzurufende VWS-Operationen spezifiziert werden, erfolgt die Definition bestimmter Eigenschaften und ihrer erwarteten Werte nach der Durchführung der Operation mittels der sog. Behauptungen (engl. *Assertions*). Die vorgeschlagene Beschreibung ist nicht endgültig und stellt eine Möglichkeit für die Durchführung von Verhaltenstests im Kontext der VWS dar. Des Weiteren bietet diese eine erste Basis für die Implementierung eines *Feasibility Check* an. Die Spezifikation der Invarianten als Teil des Deskriptors bleibt jedoch als Gegenstand weiterer Arbeiten. Als Beispiel für eine grundlegende Verhaltensbeschreibung dient Listing A.2.

4.2.5.10 BehaviorDescriptor

Die Klasse entspricht der konkreten Instanz der `IBehaviorDescriptor`-Schnittstelle und kann als Hilfsmittel für die Deserialisierung der Verhaltensbeschreibung eingesetzt werden.

Descriptor Factory

Das folgende Klassendiagramm illustriert das Modul, welches für die Instanziierung der Ticket-Deskriptoren verantwortlich ist und somit als Basis für die softwaretechnische Erstellung eines Tickets gilt.

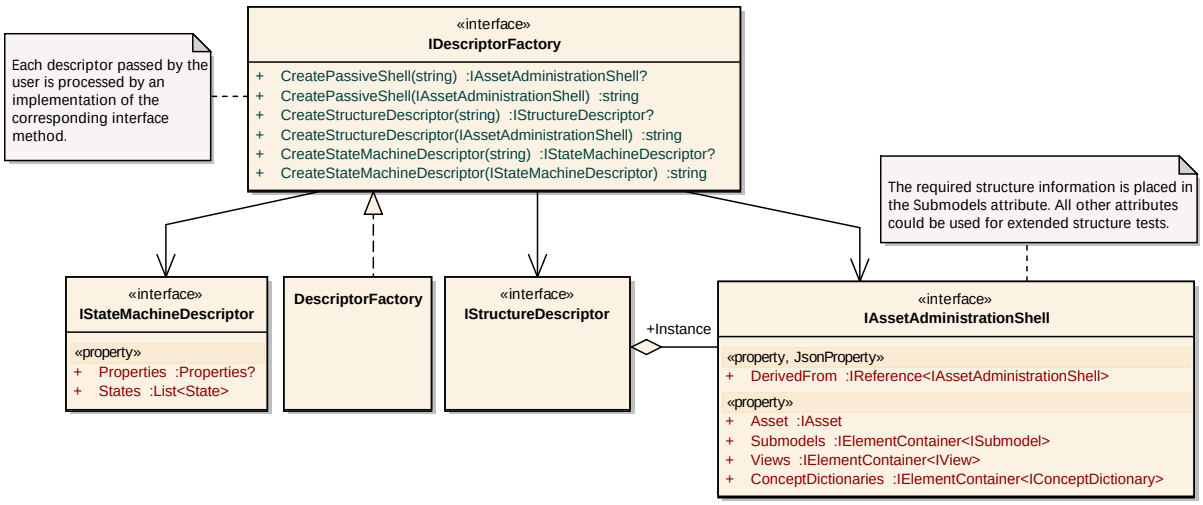


Abbildung 4.18: Klassendiagramm Descriptor Factory

4.2.5.11 IDescriptorFactory

Diese Schnittstelle definiert Methoden zur Serialisierung und Deserialisierung bzw. Instanziierung von Ticket-Deskriptoren. Alle Methoden zur Serialisierung bekommen als Eingangsinformation eine Instanz der zu serialisierenden Klasse und geben eine XML- oder JSON-Zeichenkette zurück. Die Funktionalitäten zur Deserialisierung ermöglichen die Transformation von Zeichenketten in C# Klassen. Da die Interaktion des Anwenders mit dem System die Übergabe von JSON und XML-Inhalten voraussetzt, werden im Rahmen der Deserialisierungsfunktionalitäten die bereitgestellten Inhalte nach dem entsprechenden Schema validiert. Nach der Validierung erfolgt eine Instanziierung des Deskriptors. Die Deskriptor-Instanzen werden bei der Konstruktion der Klasse `TestTicket` als Argumente der Funktionalitäten einer `TicketBuilder`-Instanz übergeben.

4.2.5.12 DescriptorFactory

Konkrete Implementierung der Schnittstelle `IDescriptorFactory`.

Test Ticket

Abbildung 4.19 stellt die Struktur eines Test-Tickets dar. Das Test-Rezept setzt sich aus einem GUID-Identifikator und mehreren Deskriptoren zusammen. Der Typ des Tickets wird durch eine Enumeration angegeben.

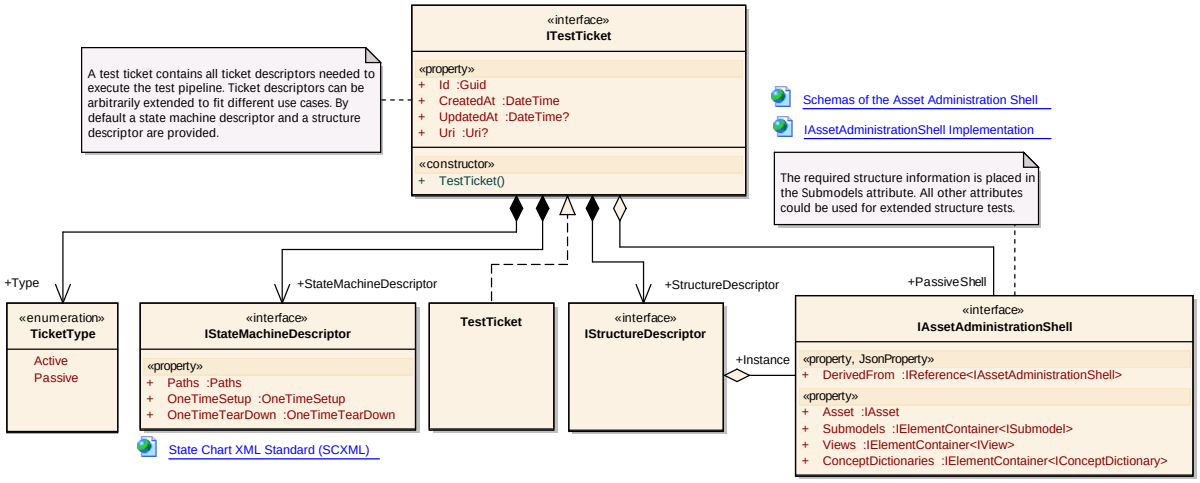


Abbildung 4.19: Klassendiagramm Test-Ticket

Obwohl das Klassendiagramm nur die zwei Deskriptoren enthält, welche im Rahmen des praktischen Anteils bereitgestellt werden, kann die Anzahl der Test-Beschreibungen je nach Anwendungsfall und Pipeline-Sequenz erweitert werden.

Ticket Builder

Da es zwischen mehreren Ticket-Typen differenziert wird, eignet sich für die Instanziierung eines Tickets die Verwendung des sog. *Builder-Pattern*. Dieses wird durch das dargestellte Klassendiagramm abgebildet. Die `ITicketBuilder`-Schnittstelle wird für die Instanziierung von Test-Rezepten für aktive und passive Verwaltungsschalen eingesetzt.

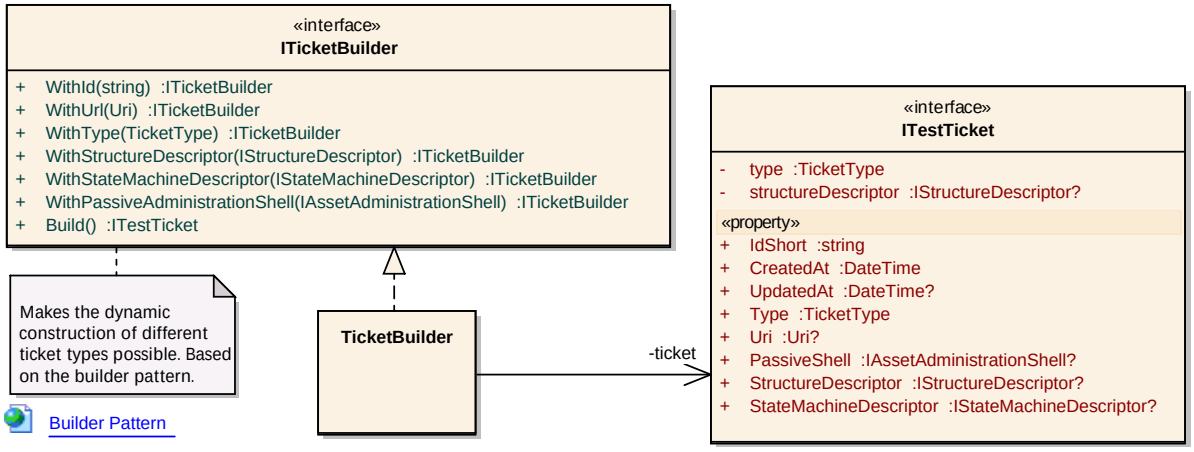


Abbildung 4.20: Klassendiagramm Ticket-Builder

4.2.5.13 ITicketBuilder

Es wird zwischen passiven, aktiven und proaktiven Verwaltungsschalen differenziert. Im Fokus des Konzepts liegt die Verifikation von Testobjekten mit aktiven und passiven Verwaltungsschalen. In Abhängigkeit vom Typ der VWS unterscheidet sich der Inhalt eines Tickets. Während ein passives Ticket die VWS-Datei des Testobjekts enthält, charakterisiert sich beispielsweise ein aktives Ticket durch die Existenz der Adresse des VWS-Endpunktes. Ein Testrezept, das sowohl eine VWS-Datei als auch eine Adresse enthält, ist in Zusammenhang mit dem Testbett nicht definiert, also ungültig, und soll dementsprechend als Ausnahmefall behandelt werden.

Das `TicketBuilder`-Interface ermöglicht die dynamische Konstruktion und Validierung von Tickets passiven und aktiven Tickets. Ein Ticket wird im Kontext der Softwareapplikation nur durch den Einsatz dieser Schnittstelle erstellt, weil dadurch die Gültigkeit der Ticket-Instanz garantiert wird.

Der Einsatz dieses Entwurfsmusters führt zu einer deutlichen Komplexitätsreduktion. Als Argument dafür spricht die Möglichkeit zur Konstruktion von mehreren Ticket-Typen durch den Einsatz einer einzigen Klasse.

4.2.5.14 TicketBuilder

Diese Klasse repräsentiert eine konkrete Implementierung der `TicketBuilder`-Schnittstelle.

Ticket Repository

Diese Komponente wurde auf Basis des *Repository*-Entwurfsmusters abgeleitet und dient der softwaretechnischen Umsetzung eines Ticketverzeichnisses.

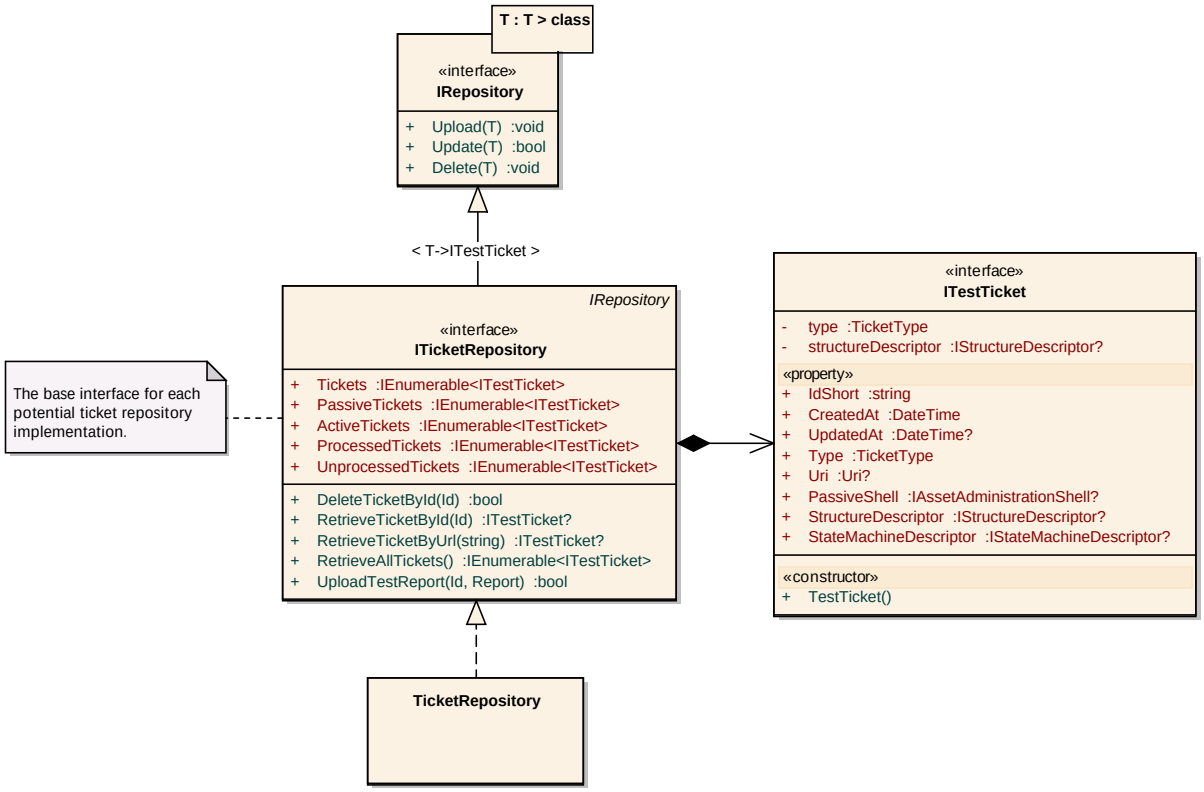


Abbildung 4.21: Klassendiagramm Ticket-Repository

4.2.5.15 IRepository

Das `IRepository` Interface definiert die grundlegenden Operationen eines Ticket Verzeichnisses.

4.2.5.16 ITicketRepository

Das `ITicketRepository`-Interface spezifiziert Funktionalitäten und Eigenschaften, welche jede konkrete Implementierung eines Ticket-Verzeichnisses umsetzen muss. Dadurch wird die Unabhängigkeit der Softwarelösung von der konkreten Implementierung des Ticket-Verzeichnisses garantiert.

Die spezifizierten Dienste wurden bereits im Rahmen der Anwendungsfalldiagramme erwähnt.

Ticket Repository Service

Das folgende Klassendiagramm veranschaulicht die Rollen der beschriebenen Komponenten im Kontext des primären Softwaremoduls `TicketRepositoryService`, siehe 4.2.4.3.

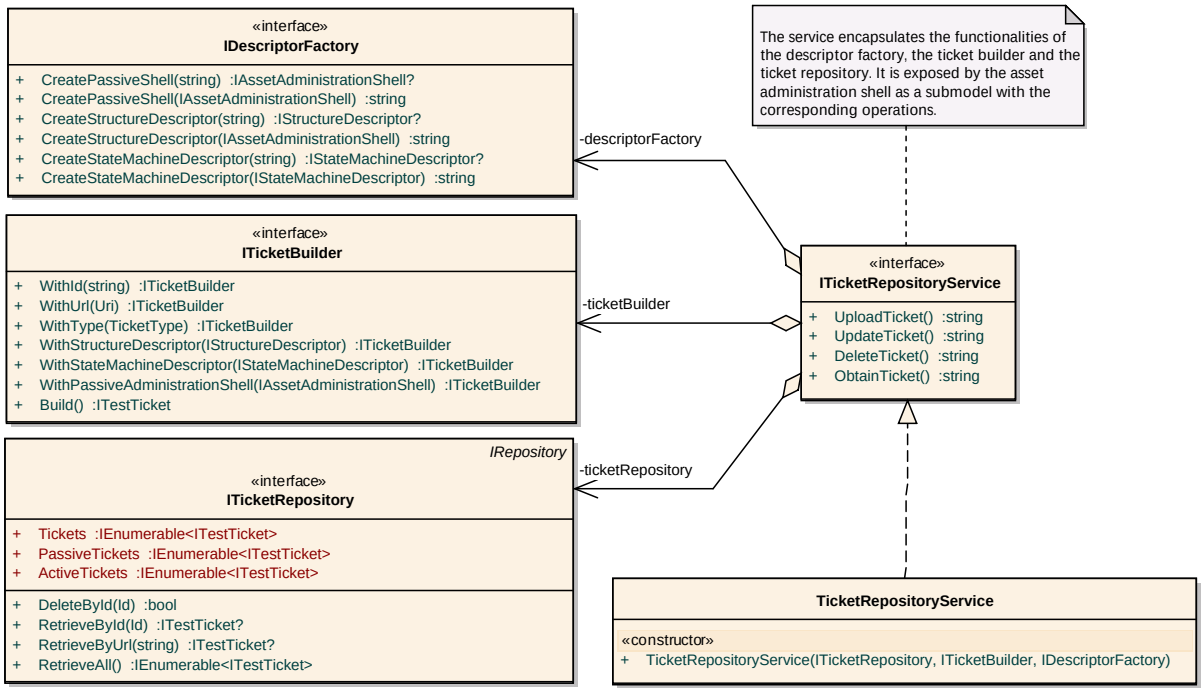


Abbildung 4.22: Klassendiagramm Ticket-Repository Service

4.2.5.17 TicketRepositoryService

Aus dem Klassendiagramm geht hervor, dass die konkrete Instanzen der Schnittstellen `IDescriptorFactory`, `ITicketBuilder` und `ITicketRepository` nach dem *Dependency Inversion* Prinzip, also als Konstruktor-Argumente, der Klasse `TicketRepositoryService` übergeben werden. Dadurch findet eine Kapselung von Funktionalitäten statt, durch welche die Implementierung der vom Interface spezifizierten Methoden ermöglicht wird.

4.2.5.18 ITicketRepositoryService

Das `ITicketRepositoryService` Interface definiert die Funktionalitäten des Repository-Dienstes. Diese werden entsprechend Abb. 4.11 von den Operationen eines im Rahmen eines nachfolgenden Abschnitts definierten Submodells aufgerufen.

4.2.5.19 Test Object

Abb. 4.23 illustriert die Bestandteile eines Testobjekts im Kontext der Softwareapplikation.

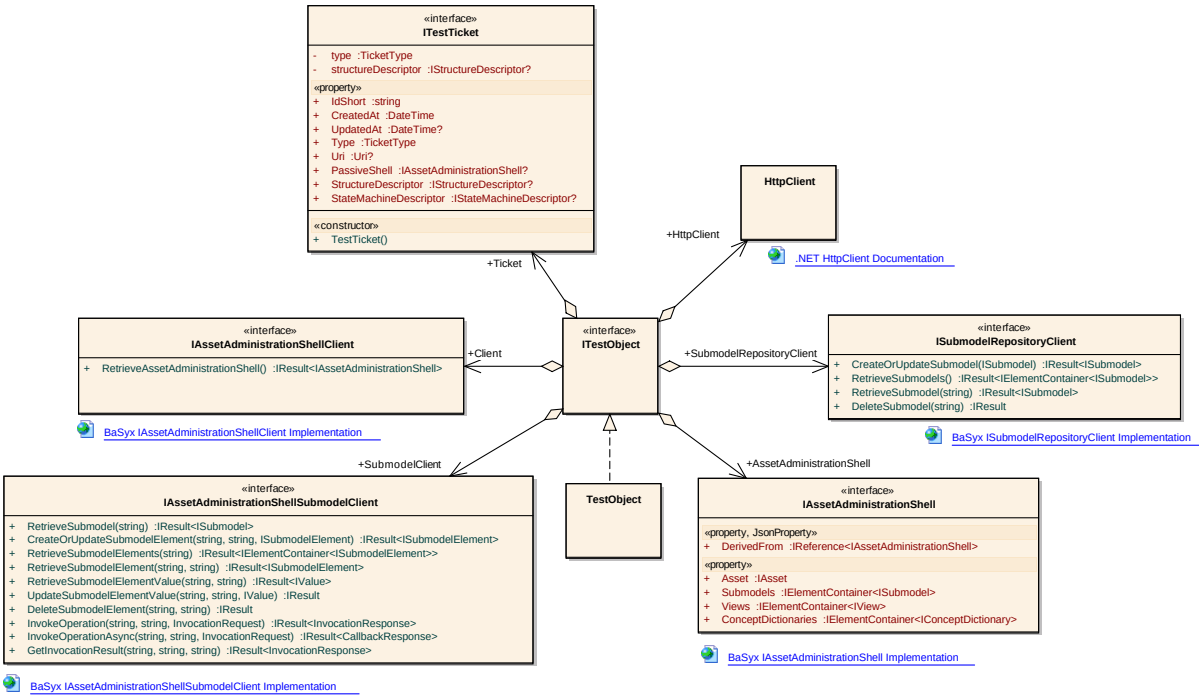


Abbildung 4.23: Klassendiagramm Test Object

4.2.5.20 ITestObject

Das `ITestObject`-Interface definiert die Eigenschaften, welche als Basis für die Testdurchführung gelten. Zu diesen zählt an erster Stelle das Test-Rezept. Beschreibt das Ticket ein aktives Testobjekt, so werden die dargestellten Client-Schnittstellen durch die entsprechenden BaSyx-Klassen instanziiert. Durch diese können Informationen vom Testobjekt im Rahmen der Test-Pipeline abgefragt werden. Ist ein passives Testobjekt identifiziert worden, so erfolgt seine Deserialisierung und darauffolgende Instanziierung durch das Interface `IAssetAdministrationShell`.

Während im Rahmen eines Ticket-orientierten Testprozesses die `TestObject`-Instanz auf Basis der Ticket-Informationen kreiert wird, erfolgt in Zusammenhang mit der Registry-basierten Testung zuerst die Verbindung zum Testobjekt und dann das Extrahieren des entsprechenden Tickets.

4.2.5.21 IAssetAdministrationShell

Die Schnittstelle repräsentiert die Verwaltungsschale des Testobjekts. Während im Rahmen der Verifikation eines aktiven Testobjekts die Instanz dieser Schnittstelle nur grundlegende Informationen zur VWS enthält, repräsentiert diese bei einem passiven Gegenstand die VWS des Testobjekts ganzheitlich.

4.2.5.22 IAssetAdministrationShellClient

Das `IAssetAdministrationShellClient`-Interface stellt dem Anwender die grundlegenden Informationen einer VWS zur Verfügung. Dazu gehören die Identifikationsdaten der Verwaltungsschale und ihrer Submodelle zusammen mit ihrer Anzahl.

4.2.5.23 IAssetAdministrationShellSubmodelClient

Das `IAssetAdministrationShellSubmodelClient`-Interface dient als Zugriffspunkt zu allen Eigenschaften eines Submodells. Die Schnittstelle definiert Methoden zum Aufrufen von Operationen, Setzen und Auslesen von Eigenschaften u.a.

4.2.5.24 IAssetAdministrationShellSubmodelRepositoryClient

Diese Schnittstelle ermöglicht das Instanzieren und Löschen und Submodellen in Systemlaufzeit.

4.2.5.25 HttpClient

Die `HttpClient`-Client ist nur für die Registry-orientierte Testdurchführung relevant. Diese Klasse wird aufgrund der fehlenden BaSyx-Operationen zum programmiertechnischen Hoch- bzw. Herunterladen von Dateien in Laufzeit einer VWS-Instanz eingesetzt.

Test Object Provider

Das nächste Klassendiagramm gibt einen Überblick über die Komponente, die zum Zweck der Konstruktion von Testobjekten eingesetzt und als *Test Object Provider* bezeichnet wird. Des Weiteren illustriert die Abbildung die Abhängigkeiten zwischen dem Provider, der Repository- und der Ticket-Komponente.

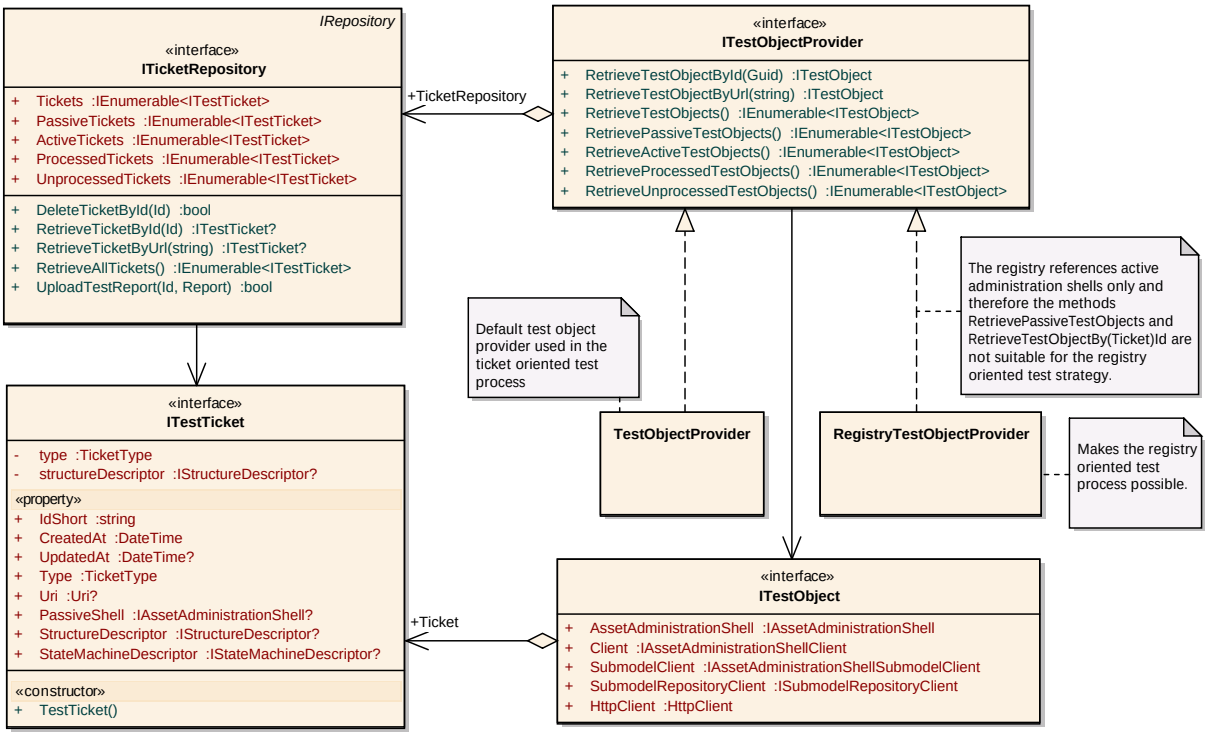


Abbildung 4.24: Klassendiagramm Test Object Provider

4.2.5.26 ITestObjectProvider

Das `ITestObjectProvider`-Interface definiert Funktionalitäten zur Bereitstellung von Instanzen der Klasse `TestObject`. Als Voraussetzung für die Konstruktion der Objekte in Zusammenhang mit der Ticket-orientierten Testdurchführung gilt der Zugriff zum Ticketverzeichnis. Aus diesem Grund bekommt die Schnittstelle eine Instanz, die das `ITicketRepository`-Interface implementiert, als Attribut. Dadurch kann die Testlösung für jedes Ticket im Verzeichnis eine `TestObject`-Instanz konstruieren, was als Voraussetzung für die Initiierung der Test-Pipeline gilt.

4.2.5.27 TestObjectProvider

Die Klasse `TestObjectProvider` implementiert die Schnittstelle `ITestObjectProvider` vollständig und dient somit als zentrale Komponente für die Realisierung einer Ticket-orientierten Test-Pipeline.

4.2.5.28 RegistryTestObjectProvider

Wie bereits mehrmals erwähnt wurde, ist die Registry-basierte Testdurchführung nur für reaktive bzw.

proaktive Testobjekte geeignet. Das `ITestObjectProvider`-Interface enthält jedoch Methoden, welche die Existenz von passiven Testobjekten voraussetzen. Aus diesem Grund implementiert die Klasse `RegistryTestObjectProvider` nur ausgewählte Methoden dieser Schnittstelle. Funktionalitäten wie `RetrieveTestObjectByTicketId()` und `RetrievePassiveTestObjects()` sind in Zusammenhang mit dieser Klasse ungültig und werden als Ausnahmefälle behandelt. Das Modul konstruiert eine `TestObject`-Instanz für jede registrierte VWS.

Test Runner

Abb. 4.25 illustriert die Komponente `TestRunner`. Die Klasse ist für die Testdurchführung verantwortlich und bildet die Schnittstelle zwischen der Softwareapplikation und dem ausgewählten .NET Test-Framework.

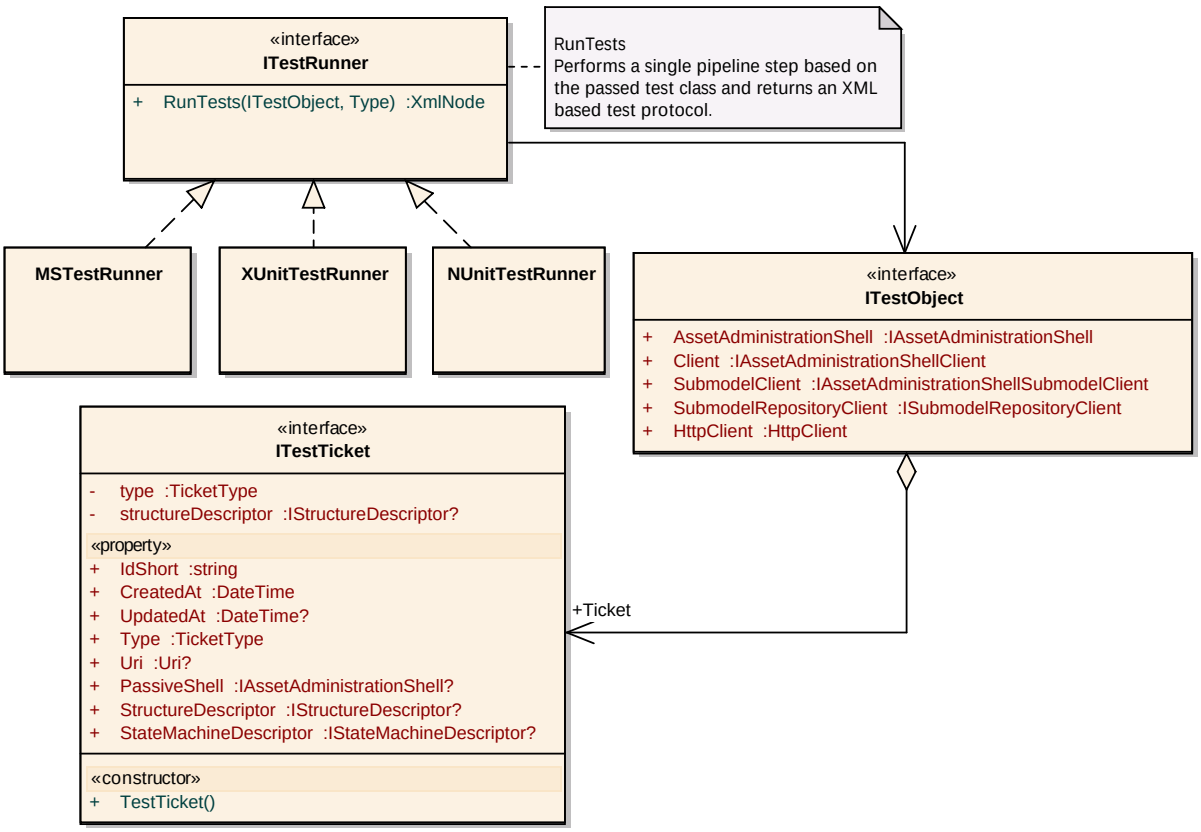


Abbildung 4.25: Klassendiagramm Test Runner

4.2.5.29 ITestRunner

Das `ITestRunner`-Interface definiert die Schnittstelle der Methode zur Durchführung einer einzigen Stufe der Test-Pipeline. Diese bekommt eine Instanz übergeben, die das `ITestObject`-Interface implementiert. Dadurch wird das Testobjekt spezifiziert. Als zweites Argument gilt eine Instanz der abstrakten .NET Klasse `Type`. Als Parameter beim Aufrufen der Methode ist somit die Übergabe einer Klasse erforderlich. Im Kontext der Testlösung muss diese Klasse die durchzuführenden Tests implementieren. Als Rückgabewert der Methode dient ein Testprotokoll in XML-Format.

Durch die Implementierung der nachfolgend aufgelisteten Klassen können die gängigsten .NET Test-Frameworks abgedeckt werden.

4.2.5.30 NUnitTestRunner

Die Klasse realisiert die `ITestRunner`-Schnittstelle und ist für die Durchführung von Test-Stufen geeignet, die auf dem *NUnit* Test-Framework basieren.

4.2.5.31 XUnitTestRunner

Die Klasse realisiert die `ITestRunner`-Schnittstelle und ist für die Durchführung von Test-Stufen geeignet, die auf dem *XUnit* Test-Framework basieren.

4.2.5.32 MSTestRunner

Die Klasse realisiert die `ITestRunner`-Schnittstelle und ist für die Durchführung von Test-Stufen geeignet, die auf dem *MSTest* Framework basieren.

Test Orchestrator

Auf Basis der bereits beschriebenen Komponenten lässt sich die zweite Kernkomponente, nämlich die Klasse `TestOrchestrator` bzw. `TestCoordinator`, ableiten. Dieses Modul stellt die Funktionalitäten für die Durchführung und die implizite Protokollierung der gesamten Test-Pipeline zur Verfügung.

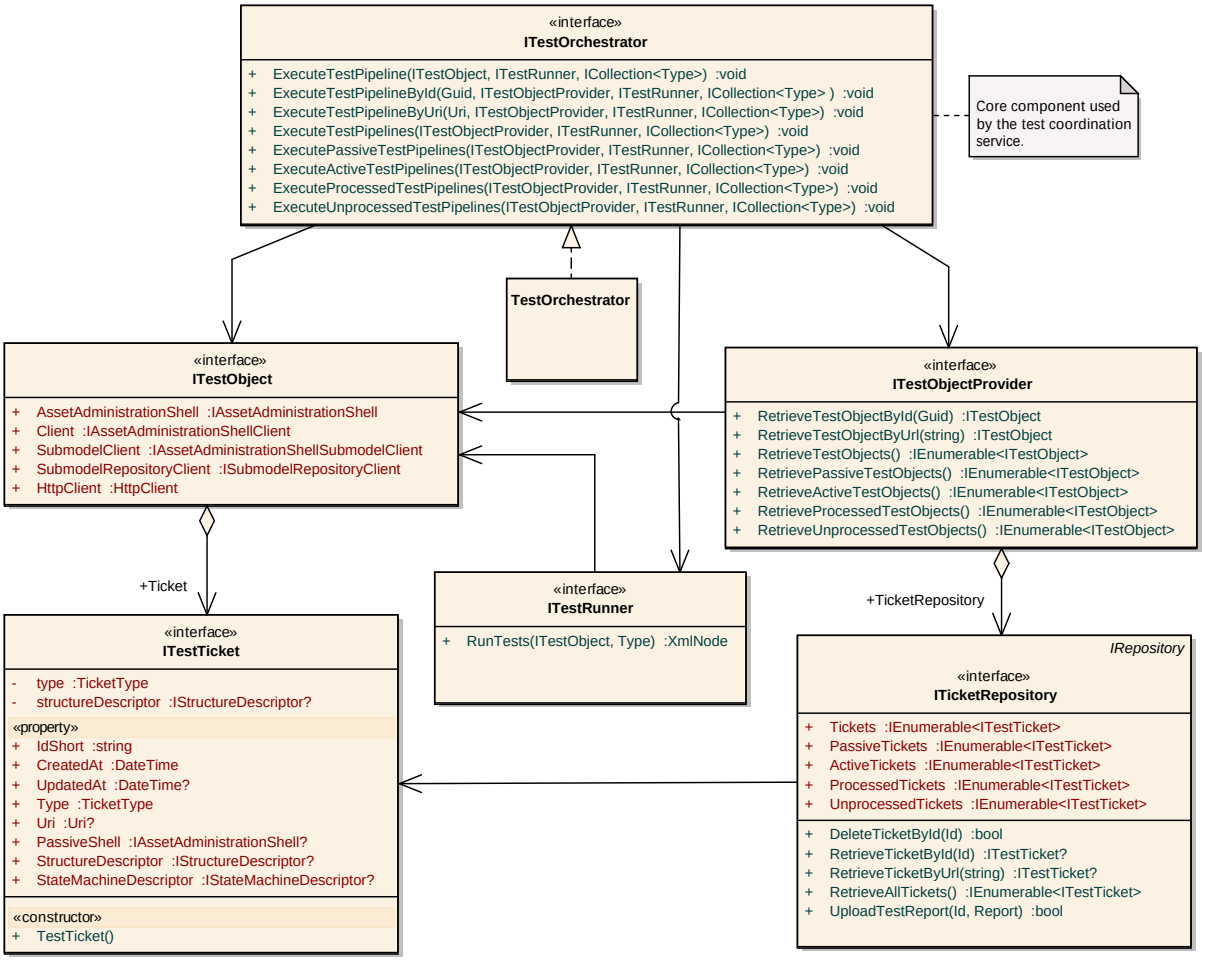


Abbildung 4.26: Klassendiagramm Test Orchestrator

4.2.5.33 ITestOrchestrator

Die `ITestOrchestrator`-Schnittstelle definiert Methoden, welche vom übergeordneten Service-Modul bei einer Initiierung der entsprechenden Testdurchführungsoperation aufgerufen werden. Diese wurden

in Abschnitt 4.2.4 spezifiziert. Mithilfe des Funktionsarguments `ITestObjectProvider` lassen sich die Testobjekte instanziieren, siehe 4.2.5.27. Die Spezifikation der Pipeline-Stufen erfolgt durch die Übergabe einer Liste von Test-Klassen. Ihre Reihenfolge definiert die Sequenz der Pipeline. Durch den Einsatz des Moduls `TestRunner` erfolgt die Durchführung der entsprechenden Testsequenzen für jedes Testobjekt.

Test Orchestration Service

Abb. 4.27 stellt die Rolle der beschriebenen Komponenten im Kontext des Orchestrierungsdienstes dar.

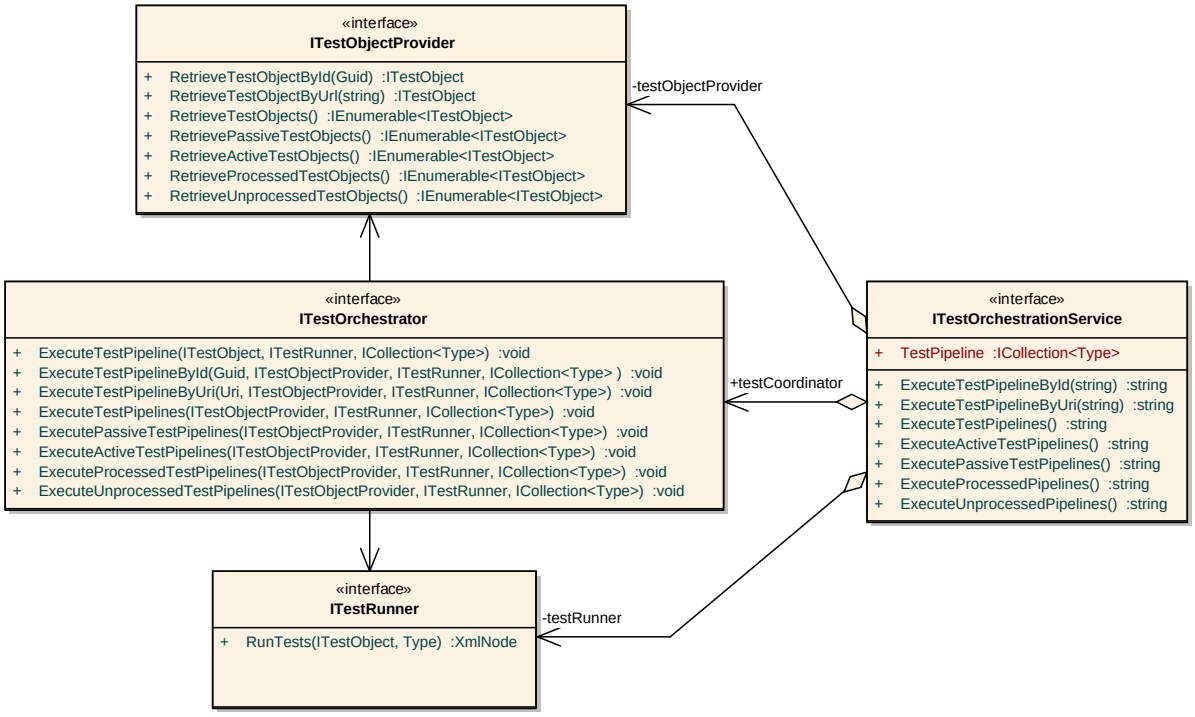


Abbildung 4.27: Klassendiagramm Test Coordination Service

4.2.5.34 ITestOrchestrationService

Der Orchestrierungsdienst dient als Adapter zwischen der Ebene der VWS und dem Test-Orchestrator. Seine Funktionalitäten werden direkt von den Operationen des entsprechenden Submodells aufgerufen. Das Modul kapselt die Dienste der untergeordneten Komponenten *Test Object Provider*, *Test Runner* und *Test Orchestrator* und implementiert eine übergeordnete Schnittstelle zur Testdurchführung.

Test Management Shell

Das nächste Klassendiagramm veranschaulicht die Kernkomponenten der obersten Softwareebene der Testautomatisierungslösung. Bei der Darstellung handelt es sich um einen konzeptionellen Vorschlag, der für die Erfüllung der im Rahmen des Kapitels bereits spezifizierten Anforderungen geeignet ist.

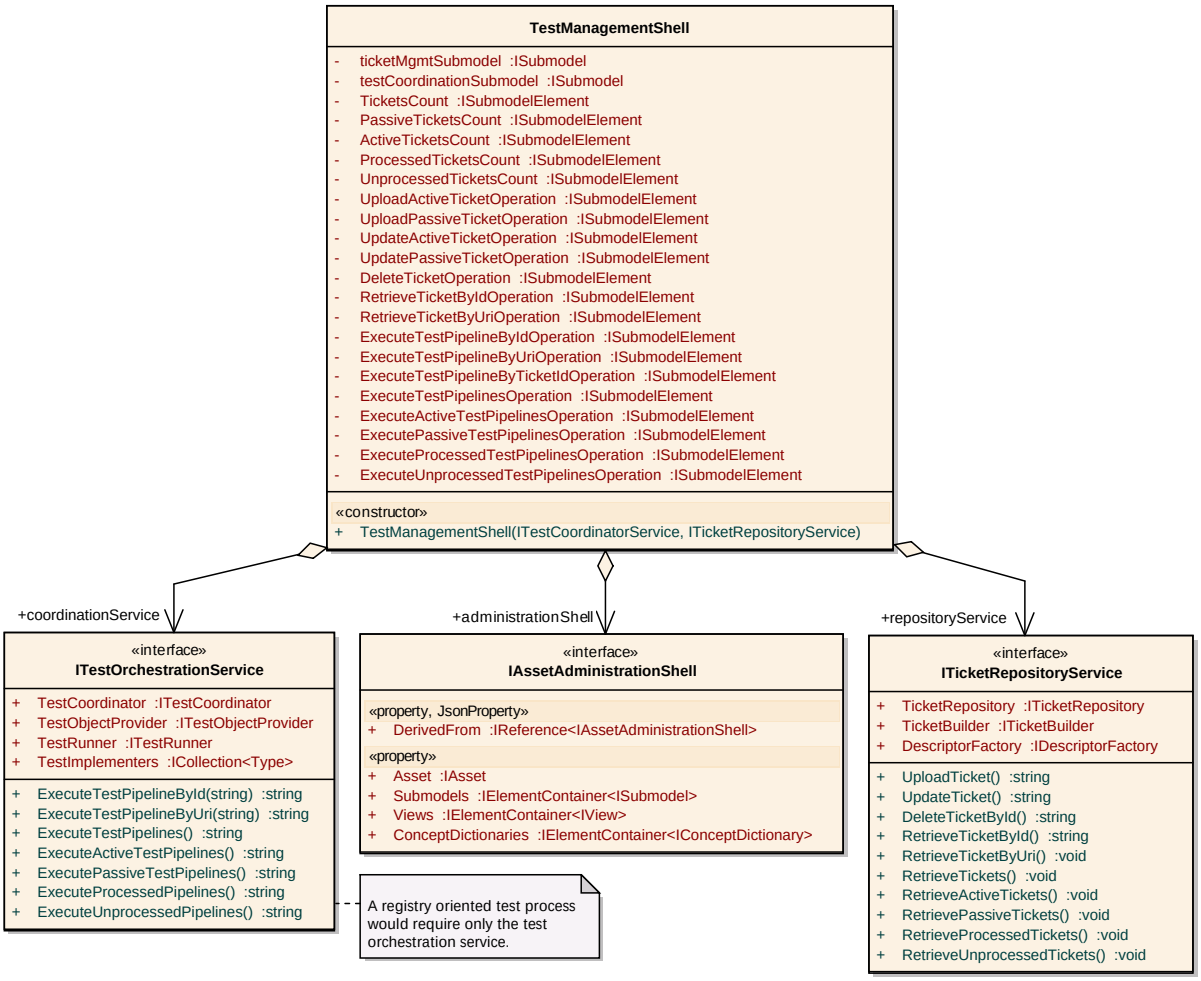


Abbildung 4.28: Klassendiagramm Test Management Shell

4.2.5.35 TestManagementShell

Die Klasse `TestManagementShell` spezifiziert die Submodelle und Eigenschaften, welche als minimale Voraussetzung für die Realisierung des Ticket-orientierten VWS-Testbetts gelten. Zu diesem Zweck werden die Services nach dem Prinzip *Constructor Dependency Injection* der Komponente zur Verfügung gestellt. Darauf basierend erfolgt im Rahmen des Konstruktors die Initialisierung der VWS der Testlösung. Es findet eine Zuordnung der dargestellten Klassenattribute zu den entsprechenden Submodellen statt. Die Kernfunktionalitäten werden durch zwei Submodelle von der Verwaltungsschale bereitgestellt, siehe Abb. 4.29.

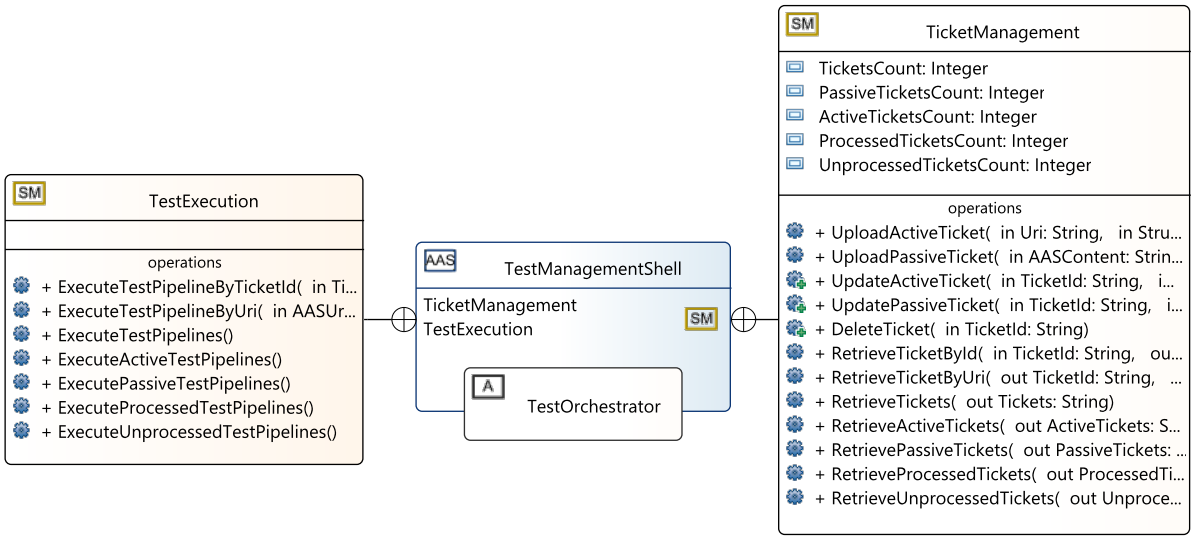


Abbildung 4.29: Test Management Shell - Struktur

Das erste Submodell repräsentiert das Ticket-System und gibt Auskünfte darüber wie viele Tickets im Verzeichnis vorhanden sind, wie viele von diesen aktive bzw. passive Verwaltungsschalen repräsentieren sowie die Anzahl der Test-Rezepten, für die eine bzw. keine Test-Pipeline durchgeführt wurde. Des Weiteren stellt das Teilmodell Operationen zur Interaktion des Nutzers mit dem Ticket-System zur Verfügung. Wie im Rahmen der Anforderungsanalyse festgelegt wurde, bietet das Modell dem Anwender die Möglichkeit, Test-Rezepte hinzuzufügen, zu löschen und zu aktualisieren. Über die entsprechenden Operationen kann der Nutzer Tickets aus dem Verzeichnis lesen.

Das zweite Submodell bietet die Operationen zur Durchführung der Test-Pipeline an. Dem Anwender wird die Möglichkeit angeboten, eine Test-Pipeline für jedes Ticket im System, für alle Tickets, die eine passive bzw. aktive VWS repräsentieren, sowie für alle prozessierten bzw. nicht-prozessierten Tickets durchzuführen.

4.3 Zusammenfassung

Im Rahmen des Kapitels wurde eine generische und erweiterbare Softwarearchitektur konzipiert, welche die automatisierte Verifikation von Industrie 4.0 Komponenten über die Verwaltungsschale ermöglicht. Die Architektur wurde nach dem *Bottom-Up*-Prinzip entworfen, wobei als Grundlage für die Ableitung der einzelnen Softwaremodule die Definition der Use-Cases zusammen mit der darauffolgenden Anforderungsanalyse diente. Durch Anwendungsfalldiagramm 4.2 wurden zwei Möglichkeiten für die Umsetzung eines VWS-basierten Testbetts vorgestellt und die entsprechenden Akteure spezifiziert. Die Kernfunktionalitäten einer vollständigen Testlösung wurden zusammen mit den entsprechenden Vorbedingungen und Postkonditionen auf Basis von Abb. 4.3 diskutiert. Im Rahmen der Beschreibung wurde die Notwendigkeit jeder Test-Stufe argumentiert und die Vorgehensweise bei einer potenziellen Implementierung erläutert. Die Beschreibung der notwendigen technischen Infrastruktur für die Realisierung der Ticket-orientierten und der Registry-basierten Testdurchführung fand in Abschnitt 4.2.2 statt. Für jede Testdurchführungsstrategie wurde weiterhin ein prinzipieller Ablauf dargelegt. Darauffolgend wurden aus dem Systemscope und den beschriebenen Anwendungsszenarien die entsprechenden Anforderungen formuliert. Im nächsten Schritt sind die Softwaremodule vorgestellt worden, welche als eine Möglichkeit für die softwaretechnische Umsetzung der Testlösung gelten.

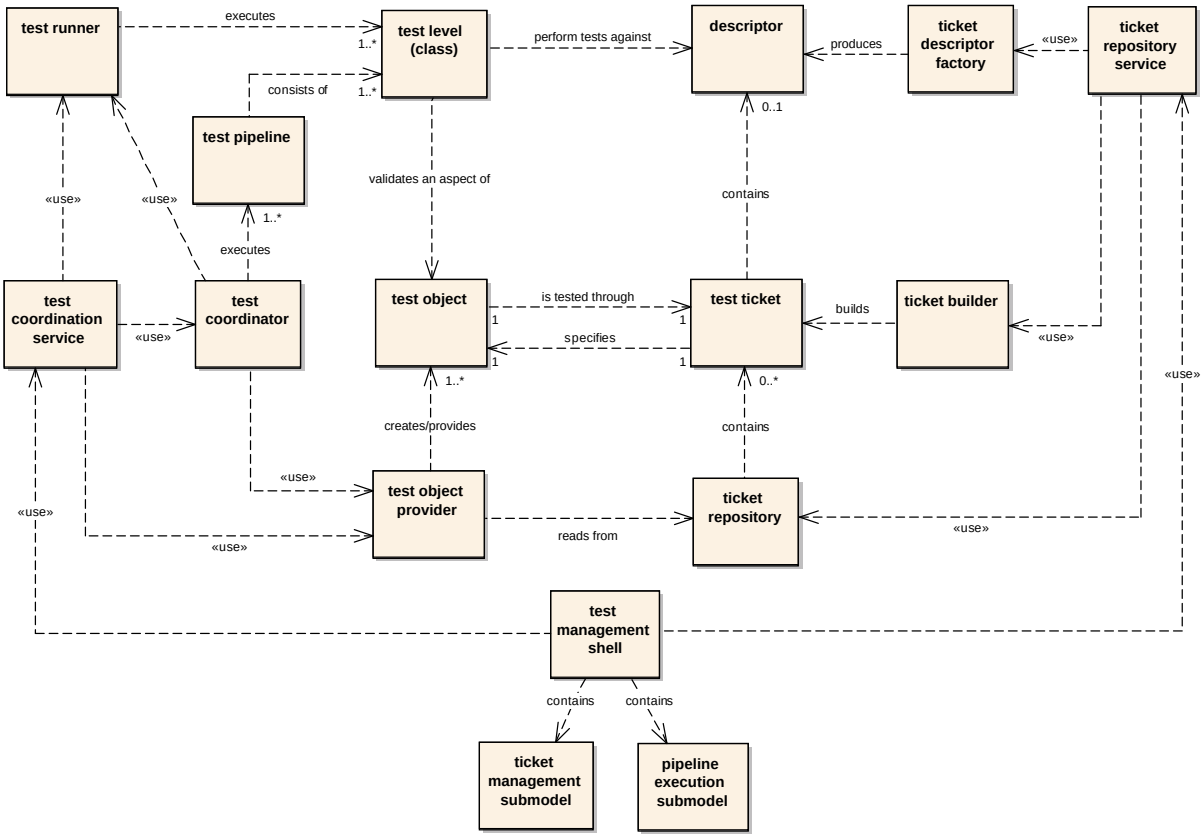


Abbildung 4.30: Überblick

Abb. 4.30 veranschaulicht die synthetisierten Komponenten und ihre Aufgaben im Kontext der Softwarearchitektur der Testautomatisierungslösung. Die Darstellung fasst die Ergebnisse des Kapitels zusammen.

Kapitel 5

Konzeptprüfung

In diesem Kapitel wird der praktische Anteil der Arbeit beschrieben. Als Schwerpunkt des Kapitels gilt die nachvollziehbare Beschreibung eines testgetriebenen Entwicklungsprozesses, der zur technischen Umsetzung einer Portion der Ticket-orientierten Softwareinfrastruktur führt. Als Ziel der Entwicklung dient die Validierung der Umsetzbarkeit des Konzeptes und die Realisierung eines Demonstrators als Nachweis für seine Realisierbarkeit. Im folgenden Abschnitt wird der ausgewählte Entwicklungsansatz behandelt. Dieser wurde auf Basis der im vorherigen Kapitel erläuterten Spezifikationen als die beste Umsetzungsmöglichkeit identifiziert. In den folgenden Abschnitten wird die testgetriebene Entwicklung jeder Komponente auf Basis einer konkreten und tabellarischen Beschreibung seiner Funktionalitäten und Testszenarien erläutert. Das Kapitel endet mit einer Zusammenfassung.

5.1 Entwicklungsansatz

Zum Zweck der Konzeptprüfung wurde das Verfahren der testgetriebenen Softwareentwicklung (engl. *Test-Driven Development*, TDD) ausgewählt. Dieser Entwicklungsprozess unterscheidet sich von der klassischen Softwareentwicklung dadurch, dass im Rahmen einer TDD das Testobjekt erst nach der Definition der entsprechenden Testmethode implementiert wird. Aus diesem Grund wird dieses Verfahren von vielen Entwicklern als Äquivalent der *Test First*-Methode angesehen.

In [Gre11] fasst Grenning den klassischen Softwareentwicklungsansatz durch den Begriff *Debug Later Programming* zusammen und gibt als Nachteil dieser Methode den undefinierten Zeitrahmen zwischen dem Entstehen eines Softwarebugs und seiner Lokalisierung und Behebung an. Des Weiteren informiert der Autor, dass die verspätete Identifikation der Fehlerursache weitere Probleme, beispielsweise durch inkorrekte Annahmen, falsch implementierte Softwarekomponenten, mit sich bringen kann.

Im Gegensatz dazu besteht die Idee der testgetriebenen Entwicklung in der Durchführung eines Aktionszyklus, der gleichzeitig eine Implementierung und Verifikation umfasst. Auf diese Weise lassen sich Softwarefehler sofort lokalisieren und beheben, was zur Steigung der Softwarequalität führt. Laut Grenning besteht ein TDD-Makrozyklus aus den durch Abb. 5.1 visualisierten Aktionen.

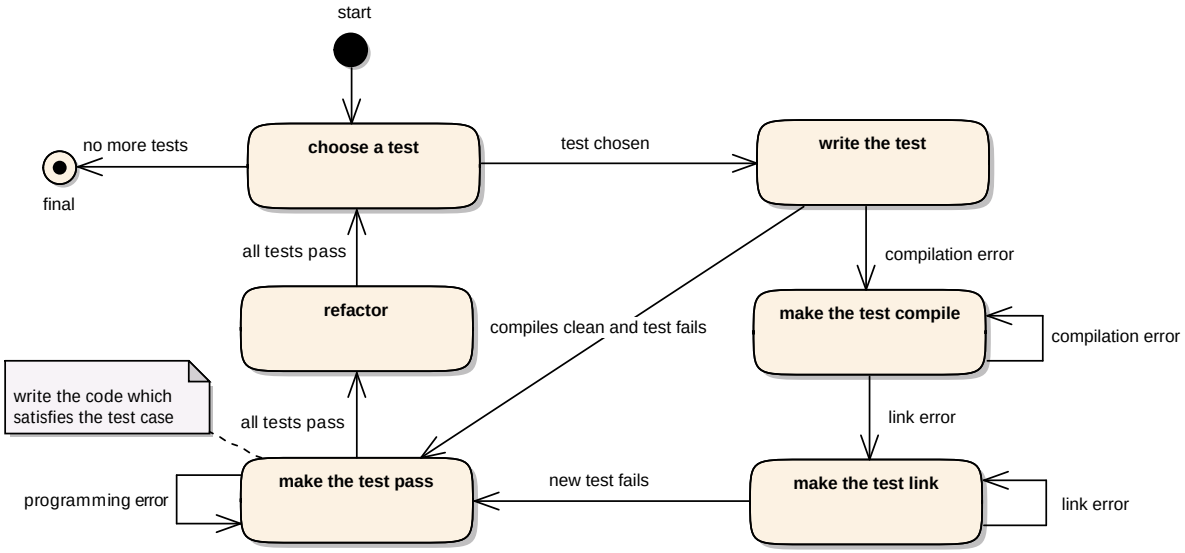


Abbildung 5.1: TDD-Makrozyklus [Gre11]

Jede im Rahmen des Designs definierte Softwarefunktionalität bekommt also ein oder mehrere Testszenarien zugeordnet. Dadurch entsteht eine Liste aus zu implementierenden Testfällen. Im Rahmen des Entwicklungsprozesses werden diese sequenziell abgearbeitet. Es handelt sich also um einen iterativen Entwicklungsprozess statt. Nach der softwaretechnischen Umsetzung des definierten Testfalls findet die Implementierung der Funktionalität statt, die durch diesen Verfiziert werden soll. Das Ziel der Implementierung besteht in dem Erfüllen der durch den Testfall gestellten Anforderungen.

Laut Karac und Turhan gilt als Voraussetzung für die Erhöhung der Softwarequalität durch diesen Entwicklungsprozess die daraus resultierenden verkürzten Entwicklungszyklen [KT18]. Des Weiteren kommentieren die Autoren, dass nicht die Reihenfolge der Aktionen, sondern ihre allgemeine Systematik, zeitliche Effizienz und disziplinierte bzw. zielgerichtete Durchführung zur Steigung der Produktivität führen. Für weitere Informationen zum Thema siehe [Gre11] und [Bec02].

5.2 Vorgehensweise

Die Herleitung der Testszenarien findet nach dem *Bottom Up*-Prinzip statt. Es werden also zuerst Testfälle für diese Module definiert, die mit keinen anderen Komponenten interagieren. Aus Abb. 5.2 geht hervor, dass die Softwarekomponenten *Test Ticket* und *Test Runner* zusammen mit den Ticket-Deskriptoren als untergeordnete Architekturelemente gelten. Im Rahmen einer testgetriebenen Entwicklung entspricht also ihre Implementierung und Verifikation der höchsten Priorität. Da die Deskriptor-Klassen kein Verhalten aufweisen, findet die Verifikation ihrer korrekten Serialisierung und Deserialisierung auf Basis des Moduls *Descriptor Factory* statt. Im nächsten Schritt erfolgt die Spezifikation der Testszenarien für diese Module, die mit den bereits getesteten Komponenten interagieren. Als solche gelten *Ticket Builder*, *Test Object*, *Ticket Repository* und *Test Object Provider*. Erst nach der Validierung des Providers lässt sich das Modul *Test Coordinator* entsprechend validieren und implementieren. Nach der testgetriebenen Umsetzung der Kernebene, findet die Realisierung der oberen Softwareschichten statt.

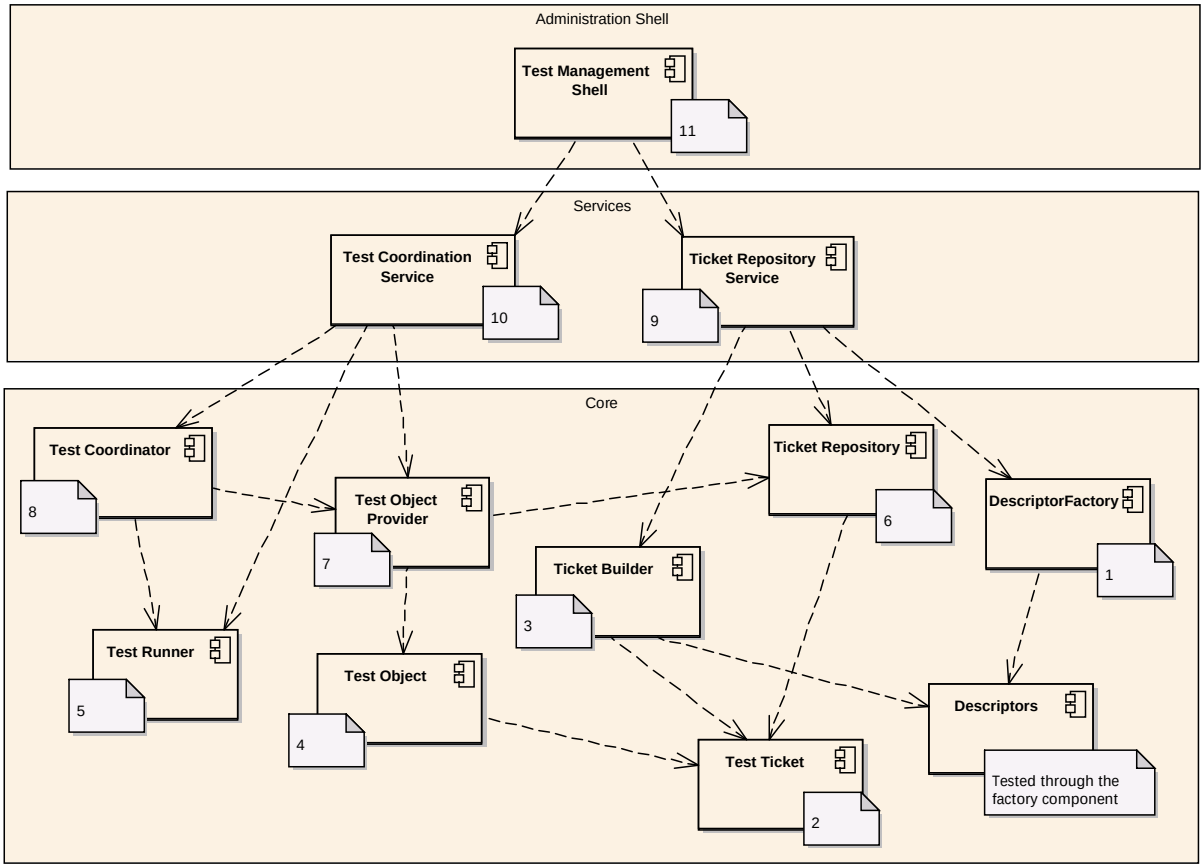


Abbildung 5.2: Softwarekomponenten und Abhängigkeiten

5.3 Kernkomponenten

Im Rahmen dieses Abschnitts wird die Implementierung und Testung aller Komponenten der Kernebene beschrieben.

5.3.1 Descriptor Factory

Wie bereits im Rahmen der Konzeption erwähnt wurde, ist die Komponente *Descriptor Factory* für die Serialisierung bzw. Deserialisierung von Ticket-Deskriptoren verantwortlich. Die vom Modul angebotenen Funktionalitäten wurden im Laufe der Konzepterstellung, anhand Abb. 4.16, grob spezifiziert. Zur detaillierten Beschreibung der Methoden und ihrer Testszenarien dienen die folgenden Beschreibungen.

Funktionalität

Komponente	Descriptor Factory
Signatur	Passive VWS instanziieren (Deserialisierung) `IAssetAdministrationShell CreatePassiveShell(string content)`
Input	Die Inhalt einer VWS-Datei, die dem XML-Schema entspricht
Return	Polymorph der Schnittstelle `IAssetAdministrationShell`
Beschreibung	Die Methode ist für die Deserialisierung passiver Verwaltungsschalen verantwortlich. Durch die Funktion lassen sich Testobjekte kontextualisieren, die durch passive Vertreter repräsentiert werden. Die Inhalt der XML-Datei wird zu einer Instanz transformiert, welche die BaSyx-Schnittstelle `IAssetAdministrationShell` implementiert.
Testszenarien	TC.DF01, TC.DF02, TC.DF03 (siehe Tabellen 5.7, 5.8, 5.9)

Tabelle 5.1: Funktionsbeschreibung: *Descriptor Factory* - `CreatePassiveShell`

Komponente	Descriptor Factory
Signatur	Passive VWS erstellen (Serialisierung) `string CreatePassiveShell(IAssetAdministrationShell descriptor)`
Input	Polymorph der Schnittstelle `IAssetAdministrationShell`
Return	Die Inhalt einer VWS-Datei, die dem XML-Schema entspricht
Beschreibung	Die Methode ist für die Serialisierung passiver Verwaltungsschalen verantwortlich. Die Instanz der Klasse `IAssetAdministrationShell` wird zu einer XML-Zeichenkette transformiert, die dem XML-Schema entspricht.
Testszenarien	TC.DF04, TC.DF05 (siehe Tabellen 5.10, 5.11)

Tabelle 5.2: Funktionsbeschreibung: *Descriptor Factory* - `CreatePassiveShell` (überladen)

Komponente	Descriptor Factory
Signatur	Strukturdeskriptor instanziieren (Deserialisierung)
Input	Die Inhalt einer VWS-Datei, die dem XML-Schema entspricht `IStructureDescriptor CreateStructureDescriptor(string content)`
Return	Polymorph der Schnittstelle `IStructureDescriptor`
Beschreibung	Die Methode dient der Deserialisierung einer XML-Datei, welche die Soll-Struktur einer VWS beschreibt und dem XML-Schema entspricht. Auf Grundlage der Deserialisierung erfolgt die Instanziierung eines Strukturdeskriptors.
Testszenarien	TC.DF06, TC.DF07, TC.DF08 (siehe Tabellen 5.12, 5.13, 5.14)

Tabelle 5.3: Funktionsbeschreibung: *Descriptor Factory* - `CreateStructureDescriptor`

Komponente	Descriptor Factory
Signatur	Strukturdeskriptor erstellen (Serialisierung) `string CreateStructureDescriptor(IAssetAdministrationShell descriptor)`
Input	Polymorph der Schnittstelle `IAssetAdministrationShell`
Return	Die Inhalt einer VWS-Datei, die dem XML-Schema entspricht
Beschreibung	Die Methode ist für die Serialisierung von Strukturdeskriptoren verantwortlich.
Testszenarien	TC.DF09, TC.DF10 (siehe Tabellen 5.15, 5.16)

Tabelle 5.4: Funktionsbeschreibung: *Descriptor Factory* - `CreateStructureDescriptor` (überladen)

Komponente	Descriptor Factory
Signatur	Deskriptor eines Zustandsautomaten instanziieren (Deserialisierung) `IStateMachineDescriptor CreateStateMachineDescriptor(string content))`
Input	Die Inhalt einer XML-Datei, welche den Zustandsautomaten zusammen mit den durchzuführenden Tests beschreibt
Return	Polymorph der Schnittstelle `IStateMachineDescriptor`
Beschreibung	Die Methode ermöglicht die Deserialisierung einer XML-Datei, welche als Basis für die Verifikation der Führungskomponente dient.
Testszenarien	TC.DF11, TC.DF12, TC.DF13 (siehe Tabellen 5.17 5.18 5.19)

Tabelle 5.5: Funktionsbeschreibung: *Descriptor Factory* - `CreateStateMachineDescriptor`

Komponente	Descriptor Factory
Signatur	Deskriptor eines Zustandsautomaten erstellen (Serialisierung) `string CreateStateMachineDescriptor(IStateMachineDescriptor descriptor)`
Input	Polymorph der Schnittstelle `IStateMachineDescriptor`
Return	Die Inhalt einer XML-Datei, welche den Zustandsautomaten zusammen mit den durchzuführenden Tests beschreibt
Beschreibung	Die Methode ermöglicht die Serialisierung einer XML-Datei, welche als Basis für die Verifikation der Führungskomponente dient.
Testszenarien	TC.DF14, TC.DF15 (siehe Tabellen 5.20 5.21)

Tabelle 5.6: Funktionsbeschreibung: *Descriptor Factory* - `CreateStateMachineDescriptor` (überladen)

Testszenarien

Identifikation	TC.DF01
Testszenario	Instanziierung (Deserialisierung) einer passiven VWS
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IAssetAdministrationShell CreatePassiveShell(string content)`
Funktionsbeschreibung	siehe Tabelle 5.1
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IAssetAdministrationShell` nach Eingabe der Inhalt einer VWS-Datei, welche eine Typ 1 VWS repräsentiert und dem XML-Schema entspricht.
Vorgehen	Vorbereitung von mehreren passiven Verwaltungsschalen. Darauffolgende Überprüfung, ob diese dem XML-Schema entsprechen. Alle Vertreter, für die keine Abweichungen vom Schema identifiziert werden, sollen als Eingangsdaten für die Funktionalität verwendet werden. Die Methode muss ein Polymorph der Schnittstelle `IAssetAdministrationShell` zurückgeben, dessen Wert nicht `null` entspricht.
Ressourcen	Als Ressourcen für diesen Testfall werden die von der IDTA bereitgestellten Verwaltungsschalen zusammen mit internen Verwaltungsschalen der HTW Berlin verwendet. Zusätzlich wird das in [Sch22c] vorgeschlagene VWS-Struktur als passive VWS abgebildet und bei der Verifikation eingesetzt. Die Gültigkeit der XML-Dateien wird durch den Einsatz des *Eclipse BaSyx Python SDK* und des zugehörigen *Compliance-Tool* sichergestellt.
Erwartung	Rückgabe eines gültigen Polymorph der `IAssetAdministrationShell`-Schnittstelle.

Tabelle 5.7: Teststrategie: *Descriptor Factory* - TC.DF01

Identifikation	TC.DF02
Testszenario	Instanziierung (Deserialisierung) einer passiven VWS
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IAssetAdministrationShell CreatePassiveShell(string content)`
Funktionsbeschreibung	siehe Tabelle 5.1
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IAssetAdministrationShell` nach Eingabe der Inhalt einer VWS-Datei, welche eine Typ 1 VWS repräsentiert und das XML-Schema entspricht. Dieses Szenario baut auf dem durch Tabelle 5.7 beschriebenen Testfall auf.
Vorgehen	Der Rückgabewert der Funktion soll nach der Instanziierung serialisiert und mit dem Eingangsargument verglichen werden (Rücktransformation).
Ressourcen	siehe Tabelle 5.7
Erwartung	Das serialisierte Objekt entspricht dem Wert des Eingangsarguments.

Tabelle 5.8: Teststrategie: *Descriptor Factory* - TC.DF02

Identifikation	TC.DF03
Testszenario	Erfolglose Instanziierung (Deserialisierung) einer passiven VWS
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IAssetAdministrationShell CreatePassiveShell(string content)`
Funktionsbeschreibung	siehe Tabelle 5.2
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IAssetAdministrationShell` nach Eingabe der Inhalt einer VWS-Datei, welche eine Typ 1 VWS repräsentiert und vom offiziellen XML-Schema abweicht.
Vorgehen	Funktionsaufruf mit ungültiger, also vom XML-Schema abweichender, VWS-Beschreibung.
Ressourcen	Interne und externe Verwaltungsschalen, die im Rahmen der Vorbereitung für Testfall TC.DF01 als ungültig identifiziert wurden, siehe Tabelle 5.7
Erwartung	Die Methode wirft Ausnahme vom Typ `NullReferenceException`.

Tabelle 5.9: Teststrategie: *Descriptor Factory* - TC.DF03

Identifikation	TC.DF04
Testszenario	Erstellung (Serialisierung) einer passiven VWS
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`string CreatePassiveShell(IAssetAdministrationShell descriptor)`
Funktionsbeschreibung	siehe Tabelle 5.2
Beschreibung	Darstellung eines Polymorph der `IAssetAdministrationShell`-Schnittstelle als VWS-Datei, die dem offiziellen XML-Schema entspricht.
Vorgehen	Deserialisierung einer bereits verifizierten VWS-Datei. Die entstandene Instanz als Eingangsargument der Methode übergeben. Softwaretechnischer Vergleich zwischen der VWS-Datei und dem Ergebnis der Methode durchführen.
Ressourcen	Interne und externe Verwaltungsschalen, die im Rahmen der Vorbereitung für Testfall TC.DF01 als gültig identifiziert wurden, siehe Tabelle 5.7.
Erwartung	Das Ergebnis der Methode stimmt mit der VWS-Datei vollständig überein.

Tabelle 5.10: Teststrategie: *Descriptor Factory* - TC.DF04

Identifikation	TC.DF05
Testszenario	Erfolglose Erstellung (Serialisierung) einer passiven VWS
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`string CreatePassiveShell(IAssetAdministrationShell descriptor)`
Funktionsbeschreibung	siehe Tabelle 5.2
Beschreibung	Darstellung eines ungültigen, also nicht serialisierbaren Polymorph der `IAssetAdministrationShell`-Schnittstelle als VWS-Datei, die dem offiziellen XML-Schema entspricht.
Vorgehen	Funktionsaufruf mit nicht serialisierbaren Eingangsargumente, z.B. `AssetAdministrationShell`-Instanzen, die dem Wert `null` entsprechen oder nicht identifiziert werden können.
Erwartung	Der Funktionsaufruf führt zu einer Ausnahme vom Typ `NullReferenceException`.

Tabelle 5.11: Teststrategie: *Descriptor Factory* - TC.DF05

Identifikation	TC.DF06
Testszenario	Instanziierung (Deserialisierung) eines Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IStructureDescriptor CreateStructureDescriptor(string content)`
Funktionsbeschreibung	siehe Tabelle 5.3
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IStructureDescriptor` nach Eingabe der Inhalt einer XML-Datei, welche eine Typ 1 VWS repräsentiert, dem offiziellen VWS-Schema entspricht und als Strukturbeschreibung dient.
Vorgehen	Funktionsaufruf mit gültiger VWS-Beschreibung
Ressourcen	Als Ressourcen für diesen Testfall gelten Verwaltungsschalen, die erfolgreich gegen das offizielle XML-Schema mittels des *Eclipse BaSyx Python SDK* und des zugehörigen *Compliance-Tool* überprüft worden sind.
Erwartung	Rückgabe eines gültigen Polymorph der `IStructureDescriptor`-Schnittstelle, dessen Wert nicht `null` entspricht.

Tabelle 5.12: Teststrategie: *Descriptor Factory* - TC.DF06

Identifikation	TC.DF07
Testszenario	Instanziierung (Deserialisierung) eines Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IStructureDescriptor CreateStructureDescriptor(string content)`
Funktionsbeschreibung	siehe Tabelle 5.3
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IStructureDescriptor` nach Eingabe der Inhalt einer VWS-Datei, welche eine Typ 1 VWS repräsentiert, dem XML-Schema entspricht und als Strukturbeschreibung dient. Dieses Szenario baut auf dem durch Tabelle 5.12 beschriebenen Testfall auf (TC.DF06).
Vorgehen	Der Rückgabewert der Funktion soll nach der Instanziierung serialisiert und mit dem Eingangsargument verglichen werden (Rücktransformation).
Ressourcen	siehe Tabelle 5.12
Erwartung	Das serialisierte Objekt entspricht dem Wert des Eingangsarguments.

Tabelle 5.13: Teststrategie: *Descriptor Factory* - TC.DF07

Identifikation	TC.DF08
Testszenario	Erfolglose Instanziierung (Deserialisierung) eines Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IStructureDescriptor CreateStructureDescriptor(string content)`
Funktionsbeschreibung	siehe Tabelle 5.3
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IStructureDescriptor` nach Eingabe der Inhalt einer VWS-Datei, welche eine Typ 1 VWS zum Zweck der Strukturbeschreibung repräsentiert und vom offiziellen XML-Schema abweicht.
Vorgehen	Funktionsaufruf mit ungültiger, also vom XML-Schema abweichender, Strukturbeschreibung.
Ressourcen	Interne und externe Verwaltungsschalen, die im Rahmen der Vorbereitung für Testfall TC.DF01 als ungültig identifiziert wurden, siehe Tabelle 5.7.
Erwartung	Die Methode wirft Ausnahme vom Typ `NullReferenceException`.

Tabelle 5.14: Teststrategie: *Descriptor Factory* - TC.DF08

Identifikation	TC.DF09
Testszenario	Erstellung (Serialisierung) eines Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`string CreateStructureDescriptor(IAssetAdministrationShell descriptor)`
Funktionsbeschreibung	siehe Tabelle 5.4
Beschreibung	Darstellung eines Polymorph der `IAssetAdministrationShell`-Schnittstelle als VWS-Datei, die dem offiziellen XML-Schema entspricht und als Strukturdeskriptor dient.
Vorgehen	Deserialisierung eines bereits verifizierten Strukturdeskriptors. Die entstandene Instanz als Eingangsargument der Methode übergeben. Softwaretechnischer Vergleich zwischen der serialisierten Beschreibung und dem Ergebnis der Methode durchführen.
Ressourcen	Interne und externe Verwaltungsschalen, die im Rahmen der Vorbereitung für Testfall TC.DF01 als gültig identifiziert wurden, siehe Tabelle 5.7.
Erwartung	Das Ergebnis der Methode stimmt mit der seriellen Strukturbeschreibung vollständig überein.

Tabelle 5.15: Teststrategie: *Descriptor Factory* - TC.DF09

Identifikation	TC.DF10
Testszenario	Erfolglose Erstellung (Serialisierung) eines Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`string CreatePassiveShell(IAssetAdministrationShell descriptor)`
Funktionsbeschreibung	siehe Tabelle 5.4
Beschreibung	Darstellung eines ungültigen, also nicht serialisierbaren Polymorph der `IAssetAdministrationShell`-Schnittstelle als VWS-Datei.
Vorgehen	Funktionsaufruf mit nicht serialisierbaren Eingangsargumenten, z.B. `AssetAdministrationShell`-Instanzen, die dem Wert `null` entsprechen oder nicht identifiziert werden können.
Erwartung	Der Funktionsaufruf führt zu einer Ausnahme vom Typ `NullReferenceException`.

Tabelle 5.16: Teststrategie: *Descriptor Factory* - TC.DF10

Identifikation	TC.DF11
Testszenario	Instanziierung (Deserialisierung) eines Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IStateMachineDescriptor CreateStateMachineDescriptor(string content)`
Funktionsbeschreibung	siehe Tabelle 5.5
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IStateMachineDescriptor` nach Eingabe der Inhalt einer Datei, welche die entsprechenden Testszenarien beschreibt und ein zwecks der Validierung zu spezifizierendem XML-Schema entspricht.
Vorgehen	Spezifikation eines XML-Schema, welches das Format des Deskriptors auf Metaebene beschreibt. Darauffolgende Definition eines Automatendeskriptors, der dem spezifizierten Schema entspricht. Es soll eine Übergabe des Dateiinhalts als Funktionsargument stattfinden.
Ressourcen	Für das Testszenario wird die Erstellung eines XML-Schema zum Zweck der Spezifikation des Deskriptors auf Metaebene vorausgesetzt. Angesichts der Testdurchführung sind XML-Dateien erforderlich, die dem Schema entsprechen.
Erwartung	Rückgabe eines gültigen Polymorph der `IStateMachineDescriptor`-Schnittstelle, dessen wert nicht `null` ist.

Tabelle 5.17: Teststrategie: *Descriptor Factory* - TC.DF11

Identifikation	TC.DF12
Testszenario	Instanziierung (Deserialisierung) eines Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IStateMachineDescriptor CreateStateMachineDescriptor(string content)`
Funktionsbeschreibung	siehe Tabelle 5.5
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IStateMachineDescriptor` nach Eingabe der Inhalt einer XML-Datei, welche die Testszenarien beschreibt und ein vordefiniertes XML-Schema entspricht.
Vorgehen	Der Rückgabewert der Funktion soll nach der Instanziierung serialisiert und mit dem Eingangsargument verglichen werden (Rücktransformation).
Ressourcen	siehe Tabelle 5.17
Erwartung	Das serialisierte Objekt entspricht dem Wert des Eingangsarguments.

Tabelle 5.18: Teststrategie: *Descriptor Factory* - TC.DF12

Identifikation	TC.DF13
Testszenario	Erfolglose Instanziierung (Deserialisierung) eines Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`IStateMachineDescriptor CreateStateMachineDescriptor(string content)`
Funktionsbeschreibung	siehe Tabelle 5.5
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `IStateMachineDescriptor` nach Eingabe der Inhalt eines Zustandsautomatendeskriptors, der vom im Rahmen der TC.DF11 (siehe Tabelle 5.17) spezifizierten XML-Schema abweicht.
Vorgehen	Funktionsaufruf mit ungültigem, also vom XML-Schema abweichendem, Deskriptors.
Ressourcen	siehe Tabelle 5.17
Erwartung	Die Methode führt zu einer Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.19: Teststrategie: *Descriptor Factory* - TC.DF13

Identifikation	TC.DF14
Testszenario	Erstellung (Serialisierung) eines Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`string CreateStateMachineDescriptor(IStateMachineDescriptor descriptor)`
Funktionsbeschreibung	siehe Tabelle 5.6
Beschreibung	Darstellung eines Polymorph der `IStateMachineDescriptor`-Schnittstelle als XML-Datei, die dem spezifizierten XML-Schema entspricht.
Vorgehen	Deserialisierung einer bereits verifizierten XML-Deskriptordatei. Die entstandene Instanz als Eingangsargument der Methode übergeben. Softwaretechnischer Vergleich zwischen der Deskriptordatei und dem Ergebnis der Methode durchführen.
Ressourcen	Definierte Strukturdeskriptoren, die dem Schema entsprechen.
Erwartung	Das Ergebnis der Methode stimmt mit dem Inhalt der Deskriptordatei vollständig überein.

Tabelle 5.20: Teststrategie: *Descriptor Factory* - TC.DF14

Identifikation	TC.DF15
Testszenario	Erfolglose Erstellung (Serialisierung) eines Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.DescriptorFactoryTests`
Funktionalität	`string CreateStateMachineDescriptor(IStateMachineDescriptor descriptor)`
Funktionsbeschreibung	siehe Tabelle 5.6
Beschreibung	Darstellung eines ungültigen, also nicht serialisierbaren Polymorph der `IStateMachineDescriptor`-Schnittstelle als XML-Deskriptordatei, die dem spezifizieren XML-Schema entspricht.
Vorgehen	Funktionsaufruf mit nicht serialisierbaren Eingangsargumente, z.B. Instanzen der Klasse `StateMachineDescriptor`, die dem Wert `null` entsprechen oder den Schema-Anforderungen nicht erfüllen.
Erwartung	Der Funktionsaufruf führt zu einer Ausnahme vom Typ `NullReferenceException`.

Tabelle 5.21: Teststrategie: *Descriptor Factory* - TC.DF15

Implementierung

Als Voraussetzung für die Implementierung der Testfälle dient die Spezifikation der `IDescriptorFactory`-Schnittstelle. Da die Verifikation nur auf Basis eines konkreten Polymorph dieses Interface realisiert werden kann, erfolgt die Implementierung der Schnittstelle durch die konkrete Klasse `DescriptorFactory`. Die Methoden dieser konkreten Implementierung werden also als Gegenstand einer testgetriebenen Entwicklung angesehen.

Die Umestzung der tabellarisch vorgestellten Testszenarien findet durch den Einsatz des .NET Testframeworks *NUnit* statt. Die Klasse, die die Testszenarien umfasst, also die *Test-Suite*, charakterisiert sich durch die Bezeichnung `DescriptorFactoryTests` und eine `SetUp`-Methode, wo die Konstruktion einer `DescriptorFactory`-Instanz erfolgt. Des Weiteren enthält das Modul die Definition der Daten, die als Eingangsinformationen für die Testfälle verwendet werden sollen, und die Methoden, durch welche die Testszenarien implementiert werden.

Nach der Spezifikation der Funktionsprototypen findet die Definition der Testdaten für alle Testfälle statt. Wie bereits im Rahmen der Testszenarien angegeben wurde, ist hierfür die Nutzung interner und

externer Verwaltungsschalen und die Spezifikation des Metamodells des Zustandsautomatendeskriptors erforderlich.

Die von der IDTA bereitgestellten AASX-Dateien werden durch den Einsatz des *AASX Package Explorer* zusammen mit den internen VWS-Instanzen der HTW Berlin in XML-Dateien konvertiert. Die Umwandlung der in [Sch22c] vorgeschlagene Struktur in eine passive VWS findet durch den Einsatz des in [Sch21] vorgestellten Generators und der mehrmals erwähnten `AssetAdministrationShellEnvironment_V2_0`[1]-Klasse statt. Die entstandene passive VWS `01_Demo_AAS.xml` wird im Rahmen der Testphasen als primäres Testobjekt für positive Testszenarien vorgesehen. Alle XML-Dateien werden als Teil eines Projektordners namens `PassiveShells` im Test-Projekt angelegt. Im nächsten Schritt findet ihre formelle Verifikation auf Basis der Klassen des Namensraums `BaSyx.Models.Export` statt, wobei im Fokus wieder die Klasse `AssetAdministrationShellEnvironment_V2_0` steht. Eine Analyse ihrer statischen Konstruktors führt zur Schlüssfolgerung, dass die Klasse bei jeder Deserialisierung eine *Callback*-basierte Validierung gegen das Schema durchführt, für eine Überprüfung der Lesbarkeit trotz erkannter Inkompatibilitäten geeignet ist und Logging-Funktionalitäten anbietet. Die Nutzung der Klasse innerhalb der Funktionalitäten der Komponente `DescriptorFactory` würde also die automatische Validierung von Strukturdeskriptoren und passiven Verwaltungsschalen ermöglichen. Digitale Vertreter und Strukturbeschreibungen, die vom offiziellen XML-Schema abweichen und dennoch erfolgreich Deserialisiert werden können, sollen vom System akzeptiert werden. Abb. A.9 illustriert die Ergebnisse der Kompatibilitätsprüfung. Die einzige problemlos deserialisierte XML-Datei ist diese, die mittels der BaSyx .NET API serialisiert wurde und sich durch die Kennzeichnung `01_Demo_AAS.xml` charakterisiert. Für alle vom Schema abweichenden und trotzdem erfolgreich deserialisierten Dateien werden die entsprechenden Fehler- bzw. Warnmeldungen ausgegeben. Als zentraler Fehler der IDTA-Samples gilt die Engabe des Wertes '1' für XML-Attribute, die dem Typ `boolean` entsprechen, derer Wert also entweder `true` oder `false` sein kann, zusammen mit der Existenz wertloser Attribute.

Nach der Klassifikation der Eingangsdaten für die (De-)Serialisierung passiver Verwaltungsschalen und Strukturbeschreibungen findet die Ableitung des Schema für die Zustandsautomatendeskriptoren aus Listing 4.3 statt. In jedem Deskriptor muss laut dem Schema genau ein Root-Element mit der Bezeichnung `paths` figurieren, das die zu verifizierenden Testpfaden spezifiziert. Die minimale Anzahl von Testpfaden entspricht dem Wert 1. Jeder Pfad muss genau ein Element mit der Bezeichnung `properties` beinhalten, das die Attribute `submodel`, `setter` und `getter` umfasst und die Schnittstelle der Verwaltungsschale zum Zustandsautomaten beschreibt. Des Weiteren charakterisiert sich jeden Pfad durch ein oder mehrere `state`-Elemente, welche die im Rahmen der Verifikation durchzulaufende Zustände und durchzuführende Transitionen beschreiben. Jede Transition setzt sich aus den obligatorischen Attributen `event` und `target` zusammen. Die durch den Einsatz des *XML Schema Definition Tool* generierte und deutlich angepasste `.xsd`-Datei ist unter A.3 aufrufbar. Im nächsten Schritt erfolgt die Spezifikation eines konkreten und formell gültigen Deskriptors anhand des Automatisierungsstandards *PackML*. Die erstellte Beschreibung ermöglicht die vollständige Verifikation des *PackML* Zustandsautomaten, siehe Listing A.4. Unabhängig davon erfolgt auch die Erstellung der Deskriptoren für alle negativen Testfälle. Diese weichen vom spezifizierten Schema durch fehlende oder inkorrekte XML-Elemente ab. Alle Dateien für die Verifikation des Zustandsautomaten befinden sich im Testprojektordner `StateMachineDescriptors`. Zum Zweck der Deserialisierung findet die Programmierung der Klasse statt, durch welche die XML-Datei als Klasse innerhalb der Entwicklungsumgebung repräsentiert wird. Die softwaretechnische Umsetzung unterscheidet sich nicht vom durch Abb. 4.16 dargestellten Zustandsautomatendeskriptor.

Durch die Verwendung der .NET-Methoden zur Dateiverwaltung lassen sich die Inhalte der Dateien extrahieren und als Argumente der Testmethoden verwenden. Auf diese Weise kann die Serializierung und Deserialisierung der Deskriptoren anhand der vorgestellten Testszenarien überprüft werden. Eine Durchführung mehrerer TDD-Makrozyklen führt zur Implementierung der Funktionalitäten der Klasse `DescriptorFactory`. Um ihren Zweck zu erfüllen, instanziieren alle Methoden die Systemklasse `XmlSerializer`, die nach Übergabe der Zieltyps die entsprechende Transformation realisiert. Da die Strukturdeskriptoren genauso wie die passive Verwaltungsschalen vom System akzeptiert und behandelt

[1] `https://github.com/eclipse-basyx/basyx-dotnet-sdk/blob/main/BaSyx.Models.Export/aas-spec-v2.0/AssetAdministrationShellEnvironment_V2_0.cs`

werden, unterscheiden sich die Methoden 5.4 und 5.2 sowie 5.1 und 5.3 prinzipiell nicht voneinander. Alle vier Funktionalitäten nutzen die Dienste der Klasse `AssetAdministrationShellEnvironment_V2_0`. Vom Interesse ist die Implementierung der Deserialisierungsmethode `CreateStateMachineDescriptor`. Diese registriert eine *Callback*-Funktion, weicle eine explizite Validierung des übergebenen XML-Inhalts gegen das entsprechende Schema durchführt. Weicht der Inhalt vom Schema ab, so wirft die Funktion eine Ausnahme vom Typ `InvalidOperationException`.

Auswertung

Abbildung 5.3 veranschaulicht die durchgeführten Testszenarien und dient als Nachweis für die Korrekheit des Moduls `DescriptorFactory`.

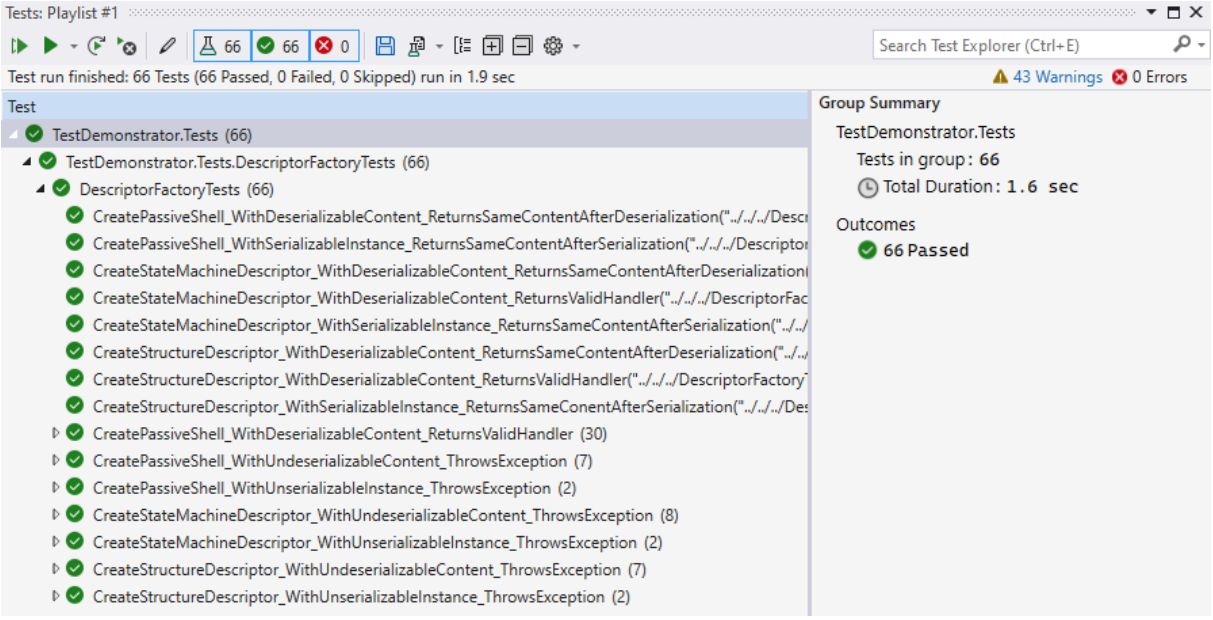


Abbildung 5.3: Testergebnisse - *Descriptor Factory*

5.3.2 Test Ticket

Wie im Rahmen der Konzepterstellung erläutert wurde, definiert die Schnittstelle `ITestTicket` die Bestandteile eines Testrezeptes zusammen mit einem Platzhalter für den Testbericht und weitere zusäztliche Informationen. Jeder Stufe der Test-Pipeline, welche bestimmte Informationen zum Zweck des Soll-/Ist-Vergleichs erfordert, wird ein Deskriptor im Ticket zugewiesen. Im Kontext der Konzeptprüfung setzt sich das Interface `ITestTicket` aus den folgenden Elementen zusammen.

Bezeichner	Datentyp	Beschreibung
`Id`	`Guid`	Globaler Identifikator
`CreatedAt`	`DateTime`	Zeitpunkt der Erstellung (UTC)
`UpdatedAt`	`DateTime?`	Zeitpunkt der letzten Aktualisierung
`Type`	`enum TicketType`	Typ der VWS des Testobjekts
`Uri`	`Uri?`	Uri der VWS des Testobjekts (wenn aktiv)
`PassiveShell`	`IAssetAdministrationShell?`	XML-Inhalt der VWS des Testobjekts (wenn passiv)
`StructureDescriptor`	`IStructureDescriptor`	Beschreibung der Soll-Struktur der VWS des Testobjekts
`StateMachineDescriptor`	`IStateMachineDescriptor?`	Beschreibung von Testszenarien eines Zustandsautomaten (wenn vorhanden)
`TestReport`	`string`	Testbericht, der nach der Verifikation von der Testlösung hochgeladen wird

Tabelle 5.22: Datenmodell: *TestTicket*

Die vom Modul angebotenen Daten und Dienste wurden im Laufe der Konzepterstellung, anhand Abb. 4.16, grob spezifiziert. Die Komponente weist kein Verhalten auf. Im Rahmen einer testgetriebenen Entwicklung sind die automatische und korrekte Initialisierung der Identifikationskennzeichnung `Id` und die Zuweisung des Zeitstempels `CreatedAt` zu verifizieren. Tabelle 5.23 dient zur Spezifikation des Konstruktors. In 5.24 wird das zugeordnete Testszenario beschrieben.

Funktionalität

Komponente	Test Ticket
Signatur	Test Ticket Instanziieren (Konstruktor) `TestTicket()`
Return	Test Ticket Instanz mit gültigem Identifikator und Erstellungszeitstempel
Beschreibung	Die Methode ist für die Erstellung von Objekten der Klasse `Test Ticket` verantwortlich
Testszenarien	TC.TT01 (siehe Tabelle 5.24)

Tabelle 5.23: Funktionsbeschreibung: *Test Ticket* - Konstruktor

Testszenarien

Identifikation	TC.TT01
Testszenario	Instanziierung eines Test-Tickets
Testgruppe	`Demonstrator.Tests.TicketTests`
Funktionalität	`TestTicket()`
Funktionsbeschreibung	siehe Tabelle 5.23
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `ITestTicket`
Vorgehen	Aufruf des Konstruktors mit darauffolgender Verifikation, ob die Instanz mit einem Identifikator und einem gültigen Zeitstempel initialisiert wurde.
Erwartung	Automatische Initialisierung der Klassenattribute `Id` und `CreatedAt` mit gültigen Daten. Alle anderen Variablen sollen automatisch den Wert `null` zugewiesen bekommen.

Tabelle 5.24: Teststrategie: *Test Ticket* - TC.TT01

Implementierung

Als Voraussetzung für die Implementierung des Testfalls dient die konkrete Implementierung der Schnittstelle `ITestTicket`. Hierfür findet die Erstellung der Klasse `TestTicket` statt. Der Konstruktor dieser Klasse wird als Gegenstand einer testgetriebenen Entwicklung angesehen.

Als Testklasse für die *Test Ticket*-Komponente dient das Modul `TicketTests`. Für den Testfall ist die Implementierung einer Vorbereitungsroutine nicht erforderlich. Im Rahmen der Testmethode findet die Instanziierung der Klasse `TestTicket` statt. Nach der Instanziierung werden die Werte der Instanz anhand des spezifizierten Testfalls verifiziert. Um die Anforderungen des Testfalls zu erfüllen, wird im Rahmen des für die Instanziierung verantwortliche Methode eine Identifikationskennzeichnung durch den Einsatz der statischen Klassenmethode `Guid.NewGuid()` generiert und dem Attribut `Id` zugewiesen. Die Erstellung des Zeitstempels erfolgt durch auf Basis der Zeitfunktionalitäten der .NET Umgebung. Die Zeitangabe entspricht dem UTC Format.

Auswertung

Abb. 5.4 weist die korrekte Implementierung der Komponente *Test Ticket* nach.

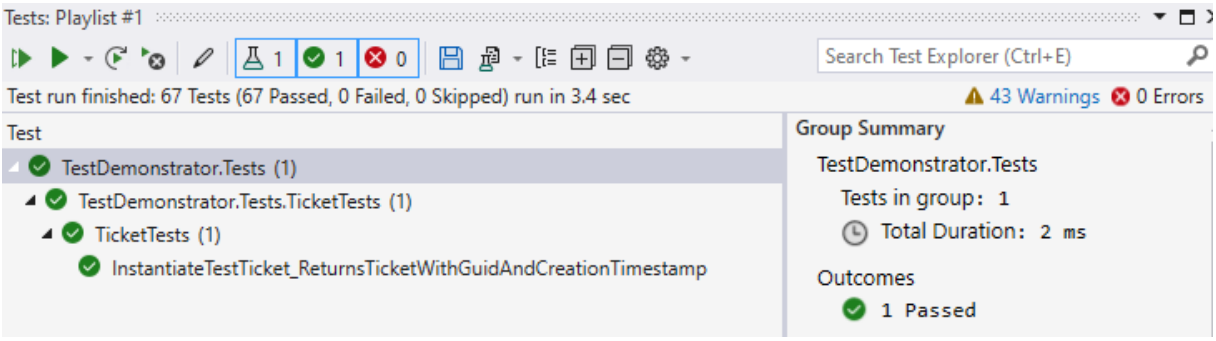


Abbildung 5.4: Testergebnisse - *Test Ticket*

5.3.3 Ticket Builder

Da die Softwarelösung zwischen Testobjekten mit passiven und aktiven Verwaltungsschalen unterscheidet, was als Voraussetzung für unterschiedliche Ausprägungen eines Testrezeptes gilt, ist die Implementierung eines Moduls für die schrittweise Konstruktion mehrerer voneinander unterschiedlichen Objekten, die das Interface `ITestTicket` implementieren, erforderlich. Im Rahmen des Konzeptes wurde dieses Modul als *Ticket Builder* bezeichnet. Wie angegeben, eignet sich für diesen Anwendungsfall die Implementierung der Komponente auf Basis des *Builder*-Entwurfsmusters. Die Schnittstelle der Komponente wird anhand der folgenden Spezifikationen umgesetzt.

Funktionalität

Komponente	Ticket Builder
Signatur	Zuweisung des GUID-Identifikators eines Polymorph der Schnittstelle `ITestTicket` `WithId(Guid id)`
Input	Identifikationskennzeichnung
Return	Polymorph der Schnittstelle `ITicketBuilder`
Beschreibung	Die Methode weist einer Instanz, die das Interface `ITestTicket` implementiert, eine Identifikationskennzeichnung zu. Da der Rückgabewert dem Typ `ITicketBuilder` entspricht, bekommt der Anwender beim Aufrufen dieser Methode kein Ticket als Ergebnis. Die Funktion ist für einen einzigen Schritt der Kreierung eines Tickets verantwortlich.
Testszenarien	siehe Tabelle 5.31

Tabelle 5.25: Funktionsbeschreibung: *Ticket Bulder* - `WithId`

Komponente	Ticket Builder
Signatur	Zuweisung des Typs eines Polymorph der Schnittstelle `ITestTicket` `WithType(TicketType type)`
Input	Typ des Tickets (Active/Passive)
Return	Polymorph der Schnittstelle `ITicketBuilder`
Beschreibung	Die Methode weist einer Instanz, die das Interface `ITestTicket` implementiert, einen Typ zu. Die Funktion ist für einen einzigen Schritt der Kreierung eines Tickets verantwortlich.
Testszenarien	siehe Tabelle 5.31

Tabelle 5.26: Funktionsbeschreibung: *Ticket Bulder* - `WithType`

Komponente	Ticket Builder
Signatur	Zuweisung der URI-Adresse eines Polymorph der Schnittstelle `ITestTicket` `WithUri(Uri uri)`
Input	Die URI der VWS des Testobjekts
Return	Polymorph der Schnittstelle `ITicketBuilder`
Beschreibung	Die Methode weist einer Instanz, die das Interface `ITestTicket` implementiert, ein URI zu. Die Funktion ist für einen einzigen Schritt der Kreierung eines aktiven Tickets verantwortlich.
Testszenarien	siehe Tabelle 5.31

Tabelle 5.27: Funktionsbeschreibung: *Ticket Bulder* - `WithUri`

Komponente	Ticket Builder
Signatur	Zuweisung der VWS des Testobjekts als Attribut eines Polymorph der Schnittstelle `ITestTicket` `WithPassiveAdministrationShell(IAssetAdministrationShell descriptor)`
Input	Polymorph der Schnittstelle `IAssetAdministrationShell`
Return	Polymorph der Schnittstelle `ITicketBuilder`
Beschreibung	Die Methode weist einer Instanz, die das Interface `ITestTicket` implementiert, eine passive VWS zu. Die Funktion ist für einen einzigen Schritt der Kreierung eines passiven Tickets verantwortlich.
Testszenarien	siehe Tabelle 5.31

Tabelle 5.28: Funktionsbeschreibung: *Ticket Bulder* - `WithPassiveAdministrationShell`

Komponente	Ticket Builder
Signatur	Zuweisung des Strukturdeskriptors eines Polymorph der Schnittstelle `ITestTicket` `WithStructureDescriptor(IStructureDescriptor descriptor)`
Input	Polymorph der Schnittstelle `IStructureDescriptor`
Return	Polymorph der Schnittstelle `ITicketBuilder`
Beschreibung	Die Methode weist einer Instanz, die das Interface `ITestTicket` implementiert, einen Strukturdeskriptor zu. Die Funktion ist für einen einzigen Schritt der Kreierung eines Tickets verantwortlich.
Testszenarien	siehe Tabelle 5.31

Tabelle 5.29: Funktionsbeschreibung: *Ticket Bulder* - `WithStructureDescriptor`

Komponente	Ticket Builder
Signatur	Zuweisung des Zustandsautomatendeskriptors einer Klasseninstanz, welche die Schnittstelle `ITestTicket` implementiert `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`
Input	Polymorph der Schnittstelle `IStateMachineDescriptor`
Return	Polymorph der Schnittstelle `ITicketBuilder`
Beschreibung	Die Methode weist einer Instanz, die das Interface `ITestTicket` implementiert, einen Zustandsautomatendeskriptor zu. Die Funktion ist für einen einzigen Schritt der Kreierung eines Tickets verantwortlich.
Testszenarien	siehe Tabelle 5.31

Tabelle 5.30: Funktionsbeschreibung: *Ticket Bulder* - `WithStateMachineDescriptor`

Komponente	Ticket Builder
Signatur	Aufbau eines Polymorph der Schnittstelle `ITestTicket` `ITestTicket Build()`
Return	Polymorph der Schnittstelle `ITestTicket`, der entweder ein aktives oder ein passives Test-Ticket repräsentiert
Beschreibung	Instanziierung eines Polymorph der Schnittstelle `ITestTicket` nach schrittweiser Konstruktion eines Tickets durch den Einsatz bestimmter Methoden des `ITicketBuilder`-Interface. Die Methode stellt die Gültigkeit des Tickets sicher. Die Konstruktion eines ungültigen Test-Rezeptes, bspw. Test-Ticket mit passiver VWS und einer URI, führt zu `InvalidOperationException`.
Testszenarien	TC.TB01, TC.TB02 (siehe Tabellen 5.32 und 5.33)

Tabelle 5.31: Funktionsbeschreibung: *Ticket Bulder* - `Build`

Testszenarien

<table>
<tr><td>Identifikation</td><td>TC.TB01</td></tr>
<tr><td>Testszenario</td><td>Konstruktion passiver Tickets</td></tr>
<tr><td>Testgruppe</td><td>`Demonstrator.Tests.TicketBuilderTests`</td></tr>
<tr><td>Funktionalität</td><td>`WithType(TicketType type)`,
`WithStructureDescriptor(IStructureDescriptor descriptor)`,
`WithPassiveAdministrationShell(IAssetAdministrationShell aas)`,
`Build()`</td></tr>
<tr><td>Funktionsbe-schreibung</td><td>siehe Tabellen 5.26, 5.29, 5.28 und 5.31</td></tr>
<tr><td>Beschreibung</td><td>Durch dieses Testszenario soll die erfolgreiche Erstellung von passiven Test-Tickets verifiziert werden.

Im Kontext der Konzeptprüfung setzt sich jedes passive Ticket aus einer Strukturbeschreibung und einer VWS als XML-Datei zusammen. Der Typ des Tickets muss trotzdem explizit spezifiziert werden. Im Rahmen zukünftiger Arbeiten ist eine Erweiterung der Definition des passiven Tickets durch die vorgestellte `ITicketBuilder`-Schnittstelle jederzeit realisierbar.</td></tr>
<tr><td>Vorgehen</td><td>Schrittweise Konstruktion eines passiven Tickets mit darauffolgendem Aufruf der `Build()`-Methode.</td></tr>
<tr><td>Ressourcen</td><td>Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden und als Strukturdeskriptoren einsetzbar sind, siehe Abb. A.9.</td></tr>
<tr><td>Erwartung</td><td>`TestTicket`-Instanz, die ein passives Ticket repräsentiert. Ein Beispiel ist durch die folgende Tabelle gegeben. Jedes Ticket, dessen `Type`-Attribut mit `TicketType.Passive` initialisiert ist und vom Beispiel (bspw. durch initialisierten Zustandsautomatendeskriptor) abweicht, soll nicht konstruiert werden können. Im Rahmen der `Build`-Methode soll die Komponente eine Ausnahme vom Typ `InvalidOperationException` mit der entsprechenden Benachrichtigung initiiert werden.
<table>
<tr><td>Bezeichner</td><td>Wert</td></tr>
<tr><td>`Id`</td><td>z.B. `4301D1EA-178E-4F7A-9E9B-8BDC6535A7D8`</td></tr>
<tr><td>`CreatedAt`</td><td>z.B. `2022-08-17 14:27:56.9427182`</td></tr>
<tr><td>`UpdatedAt`</td><td>`null`</td></tr>
<tr><td>`Type`</td><td>`TicketType.Passive`</td></tr>
<tr><td>`Uri`</td><td>`null`</td></tr>
<tr><td>`PassiveShell`</td><td>`<aas:aasenv xmlns:xsi=...>`</td></tr>
<tr><td>`StructureDescriptor`</td><td>`<aas:aasenv xmlns:xsi=...>`</td></tr>
<tr><td>`StateMachineDescriptor`</td><td>`null`</td></tr>
<tr><td>`TestReport`</td><td>`null`</td></tr>
</table></td></tr>
</table>

Tabelle 5.32: Teststrategie: *Ticket Builder* - TC.TB01

<table>
<tr><td>Identifikation</td><td>TC.TB02</td></tr>
<tr><td>Testszenario</td><td>Konstruktion aktiver Tickets</td></tr>
<tr><td>Testgruppe</td><td>`Demonstrator.Tests.TicketBuilderTests`</td></tr>
<tr><td>Funktionalität</td><td>`WithType(TicketType type)`,
`WithUri(Uri uri)`,
`WithStructureDescriptor(IStructureDescriptor descriptor)`,
`WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`,
`Build()`</td></tr>
<tr><td>Funktionsbeschreibung</td><td>siehe Tabellen 5.26, 5.27, 5.29, 5.30 und 5.31</td></tr>
<tr><td>Beschreibung</td><td>Durch das Testszenario soll die erfolgreiche Erstellung von aktiven Test-Tickets verifiziert werden. Im Kontext der Konzeptprüfung setzt sich jedes aktive Ticket aus einer Strukturbeschreibung und einem optionalen Zustandsautomatendeskriptor zusammen. Der Typ des Tickets muss trotzdem explizit spezifiziert werden. Als grundlegende Voraussetzung für die Gültigkeit eines aktiven Tickets gilt die Angabe einer korrekten URI-Adresse. Im Rahmen zukünftiger Arbeiten ist eine Erweiterung der Definition durch die vorgestellte `ITicketBuilder`-Schnittstelle jederzeit realisierbar.</td></tr>
<tr><td>Vorgehen</td><td>Schrittweise Konstruktion eines aktiven Tickets mit darauffolgendem Aufruf der `Build()`-Methode.</td></tr>
<tr><td>Ressourcen</td><td>Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.</td></tr>
<tr><td>Erwartung</td><td>`TestTicket`-Instanz, die ein aktives Ticket repräsentiert. Ein Beispiel ist durch die folgende Tabelle gegeben. Jedes Ticket, dessen `Type`-Attribut mit `TicketType.Active` initialisiert ist und vom Beispiel, bspw. durch initialisiertes `PassiveShell`-Attribut, abweicht, soll nicht konstruiert werden können. Im Rahmen der `Build`-Methode soll die Komponente eine Ausnahme vom Typ `InvalidOperationException` mit der entsprechenden Benachrichtigung initiiert werden.
<table>
<tr><td>Bezeichner</td><td>Wert</td></tr>
<tr><td>`Id`</td><td>z.B. `4301D1EA-178E-4F7A-9E9B-8BDC6535A7D8`</td></tr>
<tr><td>`CreatedAt`</td><td>z.B. `2022-08-17 14:27:56.9427182`</td></tr>
<tr><td>`UpdatedAt`</td><td>`null`</td></tr>
<tr><td>`Type`</td><td>`TicketType.Active`</td></tr>
<tr><td>`Uri`</td><td>z.B. `http://localhost:4999`</td></tr>
<tr><td>`PassiveShell`</td><td>`null`</td></tr>
<tr><td>`StructureDescriptor`</td><td>`<aas:aasenv xmlns:xsi=...>`</td></tr>
<tr><td>`StateMachineDescriptor`</td><td>`<descriptor xmlns:xsi=..>` bzw. `null`</td></tr>
<tr><td>`TestReport`</td><td>`null`</td></tr>
</table></td></tr>
</table>

Tabelle 5.33: Teststrategie: *Ticket Builder* - TC.TB02

Identifikation	TC.TB03
Testszenario	Konstruktion passiver Tickets mit ungültiger `Type`-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithPassiveAdministrationShell(IAssetAdministrationShell aas)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.29, 5.28 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die inhaltlich als passive Test-Rezepte gelten und vom Anwender mit `TicketType.Active` gekennzeichnet werden, nicht konstruierbar sind.
Vorgehen	Schrittweise Konstruktion eines passiven Tickets mit Initialisierung des `Type`-Attributs durch den Wert `TicketType.Active` und darauffolgendem Aufruf der `Build()`-Methode.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden und als Strukturdeskriptoren einsetzbar sind, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.34: Teststrategie: *Ticket Builder* - TC.TB03

Identifikation	TC.TB04
Testszenario	Konstruktion passiver Tickets ohne `Type`-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithPassiveAdministrationShell(IAssetAdministrationShell aas)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.29, 5.28 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die inhaltlich als passive Test-Rezepte gelten und vom Anwender ohne eine explizite Spezifikation des `Type`-Attributs konstruiert werden, nicht instanziierbar sind.
Vorgehen	Schrittweise Konstruktion eines passiven Tickets ohne explizite Spezifikation des `Type`-Attributs.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden und als Strukturdeskriptoren einsetzbar sind, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.35: Teststrategie: *Ticket Builder* - TC.TB04

Identifikation	TC.TB05
Testszenario	Konstruktion passiver Tickets mit fehlenden Strukturdeskriptoren
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithPassiveAdministrationShell(IAssetAdministrationShell aas)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.28 und 5.31
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass passive Tickets, für welche keine Strukturbeschreibung spezifiziert wird, nicht instanziiert werden können.
Vorgehen	Schrittweise Konstruktion eines passiven Tickets ohne explizite Spezifikation des `StructureDescriptor`-Attributs.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.36: Teststrategie: *Ticket Builder* - TC.TB05

Identifikation	TC.TB06
Testszenario	Konstruktion passiver Tickets ohne VWS-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.29 und 5.31
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass passive Tickets, für welche keine Verwaltungsschale als Testobjekt spezifiziert wird, nicht instanziiert werden können.
Vorgehen	Schrittweise Konstruktion eines passiven Tickets ohne explizite Spezifikation des `PassiveShell`-Attributs.
Ressourcen	Strukturdeskriptoren, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.37: Teststrategie: *Ticket Builder* - TC.TB06

Identifikation	TC.TB07
Testszenario	Konstruktion passiver Tickets mit Zustandsautomatendeskriptoren
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithPassiveAdministrationShell(IAssetAdministrationShell aas)`, `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.29, 5.28, 5.30 und 5.31
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass passive Tickets, für welche einen Zustandsautomatendeskriptor spezifiziert wird, nicht instanziiert werden können.
Vorgehen	Schrittweise Konstruktion eines passiven Tickets mit Spezifikation des `StructureDescriptor`-Attributs.
Ressourcen	Verwaltungsschalen und Strukturdeskriptoren, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.38: Teststrategie: *Ticket Builder* - TC.TB07

Identifikation	TC.TB08
Testszenario	Konstruktion passiver Tickets mit URI-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithPassiveAdministrationShell(IAssetAdministrationShell aas)`, `WithUri(Uri uri)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.29, 5.28, 5.27 und 5.31
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass passive Tickets, für welche eine URI-Adresse spezifiziert wird, nicht instanziiert werden können.
Vorgehen	Schrittweise Konstruktion eines passiven Tickets mit Spezifikation des `Uri`-Attributs.
Ressourcen	Verwaltungsschalen und Strukturdeskriptoren, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.39: Teststrategie: *Ticket Builder* - TC.TB08

Identifikation	TC.TB09
Testszenario	Konstruktion aktiver Tickets ohne `Type`-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithUri(Uri uri)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.27, 5.29, 5.30 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die inhaltlich als aktive Test-Rezepte gelten und vom Anwender ohne eine explizite Spezifikation des `Type`-Attributs konstruiert werden, nicht instanziierbar sind.
Vorgehen	Schrittweise Konstruktion eines aktiven Tickets ohne explizite Spezifikation des `Type`-Attributs.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden und als Strukturdeskriptoren einsetzbar sind, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.40: Teststrategie: *Ticket Builder* - TC.TB09

Identifikation	TC.TB10
Testszenario	Konstruktion aktiver Tickets mit ungültiger `Type`-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithUri(Uri uri)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.27 5.29, 5.30 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die inhaltlich als aktive Test-Rezepte gelten und vom Anwender mit `TicketType.Passive` gekennzeichnet werden, nicht konstruierbar sind.
Vorgehen	Schrittweise Konstruktion eines aktiven Tickets mit Initialisierung des `Type`-Attributs durch den Wert `TicketType.Passive` und darauffolgendem Aufruf der `Build()`-Methode.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden und als Strukturdeskriptoren einsetzbar sind, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.41: Teststrategie: *Ticket Builder* - TC.TB10

Identifikation	TC.TB11
Testszenario	Konstruktion aktiver Tickets ohne `Uri`-Angabe
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.29, 5.30 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die inhaltlich als aktive Test-Rezepte gelten, ohne Angabe einer URI-Adresse nicht konstruierbar sind.
Vorgehen	Schrittweise Konstruktion eines aktiven Tickets ohne Initialisierung des `Uri`-Attributs.
Ressourcen	Strukturdeskriptoren, die im Rahmen der Klassifikation als gültig festgelegt wurden und als Strukturdeskriptoren einsetzbar sind, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.42: Teststrategie: *Ticket Builder* - TC.TB11

Identifikation	TC.TB12
Testszenario	Konstruktion aktiver Tickets ohne Angabe von Strukturbeschreibungen
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithUri(Uri uri)`, `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.27, 5.30 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die als aktive Test-Rezepte gelten, ohne Angabe einer Strukturbeschreibung nicht konstruierbar sind.
Vorgehen	Schrittweise Konstruktion eines aktiven Tickets ohne explizite Spezifikation des `StructureDescriptor`-Attributs.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.43: Teststrategie: *Ticket Builder* - TC.TB12

Identifikation	TC.TB13
Testszenario	Konstruktion aktiver Tickets mit Angabe passiver VWS
Testgruppe	`Demonstrator.Tests.TicketBuilderTests`
Funktionalität	`WithType(TicketType type)`, `WithUri(Uri uri)`, `WithStructureDescriptor(IStructureDescriptor descriptor)`, `WithStateMachineDescriptor(IStateMachineDescriptor descriptor)`, `WithPassiveAdministrationShell(IAssetAdministrationShell aas)`, `Build()`
Funktionsbeschreibung	siehe Tabellen 5.26, 5.27, 5.29, 5.30, 5.28 und 5.31
Beschreibung	Durch diesen Testfall soll bestätigt werden, dass Tickets, die als aktive Test-Rezepte gelten, nach Angabe einer passiven VWS nicht konstruierbar sind.
Vorgehen	Schrittweise Konstruktion eines aktiven Tickets mit Spezifikation einer passiven VWS zum Zweck der Initialisierung des `PassiveShell`-Attributs.
Ressourcen	Verwaltungsschalen, die im Rahmen der Klassifikation als gültig festgelegt wurden, siehe Abb. A.9.
Erwartung	Die Konstruktion wirft eine Ausnahme vom Typ `InvalidOperationException`

Tabelle 5.44: Teststrategie: *Ticket Builder* - TC.TB13

Implementierung

Wie durch Klassendiagramm 4.20 illustriert und mehrmals erwähnt wurde, produziert die *Ticket Builder*-Komponente ein Polymorph der Schnittstelle `ITestTicket`. Im Rahmen des Konstruktors findet die Instanziierung der `TestTicket`-Klasse als privates Attribut statt. D.h. es existiert eine direkte Abhängigkeit zwischen der *TicketBuilder*- und der *Test Ticket*-Komponente. Jede aufgerufene Methode führt zur Initialisierung eines Bestandteils des privat instanziierten Test-Rezeptes und gibt die `TicketBuilder`-Instanz zurück. Dies ermöglicht das konsequente Aufrufen von weiteren instanzbezogenen Erweiterungsmethoden[2] zum Zweck der Konstruktion eines aktiven bzw. passiven Test-Tickets.

Listing 5.1 veranschaulicht wie ein aktives Ticket durch die API der *TicketBuilder*-Komponente erfolgreich instanziiert werden kann.

```
1  ITicketBuilder builder = new TicketBuilder();
2
3  ITestTicket ticket = builder
4    .WithType(TicketType.Active)
5    .WithUrl(new Uri(uriString))
6    .WithStructureDescriptor(structureDescriptor)
7    .WithStateMachineDescriptor(stateMachineDescriptor) //optional
8    .Build();
```

Listing 5.1: Instanziierung aktiver Test-Tickets

Während alle anderen Methoden zur schrittweisen Initialisierung eines Testrezeptes dienen, erfolgt die eigentliche Validierung und Rückgabe der Ticket-Instanz durch die `Build()`-Methode. Dadurch wird sichergestellt, dass Tickets, welche von den Anforderungen an einem passiven bzw. aktiven Test-Rezept abweichen, nicht konstruiert werden können.

[2] `https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods`

Auswertung

Abb. 5.5 zeigt, dass die Anforderungen an das Modul, welche durch die teballarisch aufgestellten Testszenarien konkretisiert wurden, von der Implementierung erfüllt werden.

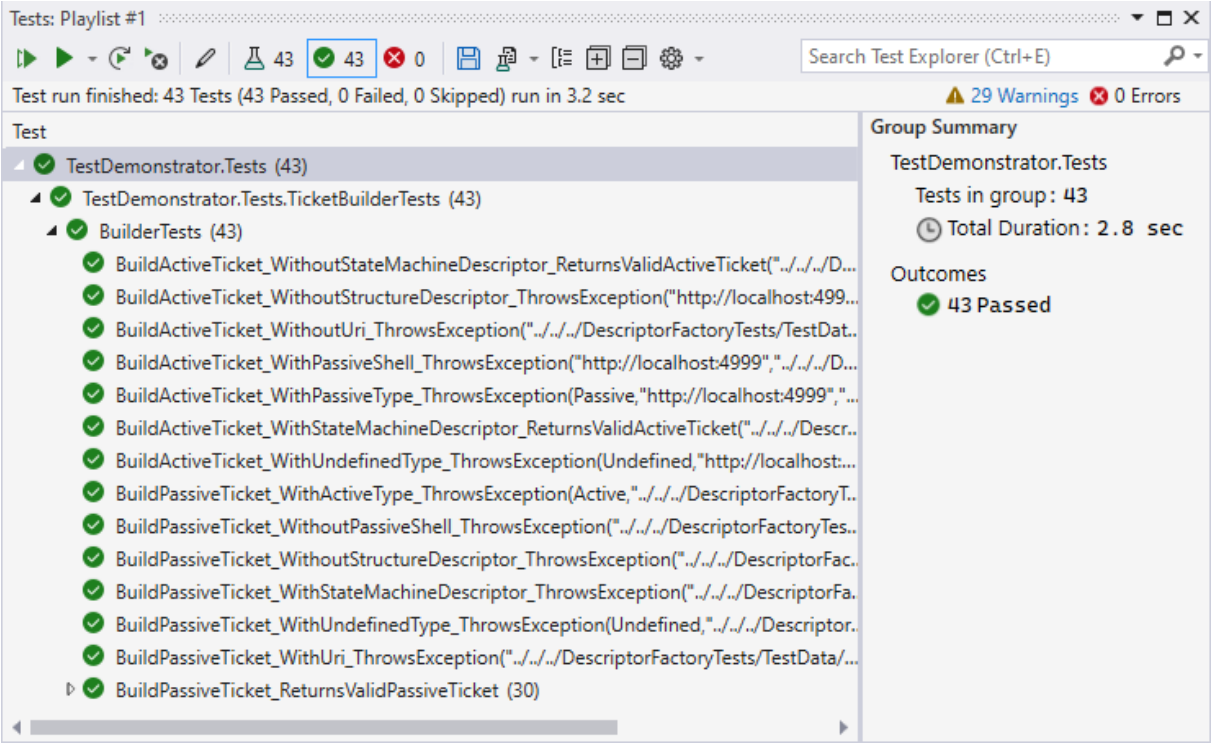


Abbildung 5.5: Testergebnisse - *Ticket Builder*

5.3.4 Test Object

Nach der Implementierung der Schnittstelle `ITicketBuilder` findet die Umsetzung der Komponente *Test Object* statt. Es wird daran erinnert, dass beide Komponenten durch eine Kompotisionsbeziehung miteinander verbunden sind. Wie im Rahmen des Konzeptes spezifiziert wurde, setzt sich jedes Polymorph der Schnittstelle `ITestObject` aus den folgenden Instanzattributen zusammen.

Bezeichner	Datentyp	Beschreibung
`Ticket`	`ITestTicket`	Beschreibung des Testobjekts (Test-Rezept), die zum Zweck des Soll-/Ist-Vergleichs Deskriptoren enthält
`AssetAdministration-Shell`	`IAssetAdministrationShell`	Unvollständige VWS des Testobjekts
`Client`	`IAssetAdministration-ShellClient`	BaSyx Client-Instanz zur Erfassung der grundlegenden AAS-Daten
`SubmodelClient`	`IAssetAdministration-ShellSubmodelClient`	BaSyx Client-Instanz zur Erfassung von Submodellen und Submodell-Elementen
`SubmodelRepository-Client`	`ISubmodelRepositoryClient`	BaSyx Client-Instanz zum Kreieren, Aktualisieren, Löschen und Erfassen von Submodellen
`HttpClient`	`HttpClient`	Client-Instanz zum Senden von HTTP-Anfragen, die von der BaSyx API nicht definiert sind

Tabelle 5.45: Datenmodell: *Test Object*

Da ein Testobjekt im Kontext der Testlösung durch kein Verhalten charakterisiert ist, wird nur der Konstruktor der Klasse `TestObject` als Gegenstand einer testgetriebenen Entwicklung angesehen. Die Klasse gilt als konkrete Implementierung des `ITestObject`-Interface und definiert den folgenden Konstruktor.

Funktionalität

Komponente	Test Object
Signatur	Instanziierung eines Testobjekts `TestObject(ITestTicket ticket)`
Input	Polymorph der Schnittstelle `ITestTicket`
Return	Polymorph der Schnittstelle `ITestObject`
Beschreibung	Die Methode führt zur Konstruktion einer Instanz, welche der Schnittstelle `ITestObject` entspricht. Anhand des Tickets lässt sich die Aussage treffen, ob eine passive oder eine aktive VWS zu validieren ist. Im Rahmen der Konstruktion von Testobjekten, die durch aktive Verwaltungsschalen repräsentiert werden, findet ein Verbindungsaufbau zur VWS der Anlagenkomponente statt. Durch die realisierte HTTP-Verbindung lassen sich zu jedem Zeitpunkt der Testdurchführung die Ist-Werte der Eigenschaften der VWS des Testobjekts erfassen und dadurch aktive Repräsentanten verifizieren. Zum Zweck der Testung passiver Verwaltungsschalen ist keine Netzwerkverbindung zur VWS erforderlich. Die Ist-Werte sind unter dem Ticket-Attribut `PassiveShell` verfügbar, siehe Tabelle 5.22.
Testszenarien	TC.TO01, TC.TO02 (siehe Tabelle 5.47, 5.48)

Tabelle 5.46: Funktionsbeschreibung: *Test Object* - Konstruktor

Testszenarien

Identifikation	TC.TO01
Testszenario	Erfolgreiche Konstruktion von `TestObject`-Instanzen bei Übergabe passiver Tickets
Testgruppe	`Demonstrator.Tests.TestObjectTests`
Funktionalität	`TestObject(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.46
Beschreibung	Instanziierung eines Testobjekts mit passiver VWS
Vorgehen	Übergabe eines validen Tickets mit Strukturbeschreibung und passive VWS als Konstruktor-Argument
Ressourcen	Als Daten für die Instanziierung des Tickets sollen bereits bei der Klassifizierung der gültigen und ungültigen Eingangsdaten verizierten Verwaltungsschalen verwendet werden, siehe Abb. A.9. Die VWS `01_Demo_AAS.xml` kann beispielsweise zum Zweck des Testfalls sowohl als Strukturdeskriptor als auch als VWS verwendet werden.
Erwartung	Erfolgreiche Instanziierung eines Testobjekts

Tabelle 5.47: Teststrategie: *Test Object* - TC.TO01

Identifikation	TC.TO02
Testszenario	Erfolgreiche Konstruktion von `TestObject`-Instanzen bei Übergabe aktiver Tickets
Testgruppe	`Demonstrator.Tests.TestObjectTests`
Funktionalität	`TestObject(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.46
Beschreibung	Instanziierung eines Testobjekts mit aktiver VWS
Vorgehen	Übergabe eines validen Tickets, das eine aktive VWS beschreibt, als Konstruktor-Argument. Damit die Konstruktion des Testobjekts erfolgreich ist und zum Zweck des Aufbaus der Kommunikationsbeziehung, muss die aktive VWS muss zum Beginn des Testfalls gestartet werden.
Ressourcen	Als Strukturdeskriptor wird die VWS `01_Demo_AAS.xml` verwendet.
Erwartung	Erfolgreiche Realisierung der Verbindung zur VWS der Anlagenkomponente und korrekte Instanziierung des Testobjekts

Tabelle 5.48: Teststrategie: *Test Object* - TC.TO02

Identifikation	TC.TO03
Testszenario	Erfolglose Konstruktion von `TestObject`-Instanzen bei Übergabe aktiver Tickets, die nicht erreichbare Verwaltungsschalen repräsentieren
Testgruppe	`Demonstrator.Tests.TestObjectTests`
Funktionalität	`TestObject(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.46
Beschreibung	Instanziierung eines Testobjekts, dessen Ticket eine VWS beschreibt, die nicht erreichbar ist
Vorgehen	Übergabe eines validen Tickets als Konstruktor-Argument, das eine aktive VWS beschreibt, welche auf die spezifizierte URL-Adresse nicht erreichbar ist
Ressourcen	siehe Tabelle 5.47
Erwartung	Der Konstruktor wirft Ausnahme vom Typ `ArgumentException`

Tabelle 5.49: Teststrategie: *Test Object* - TC.TO03

Implementierung

Die Vorgehensweise bei der Konstruktion eines Testobjekts im Kontext des Ticket-orientierten Testprozesses basiert auf dem Typ des Tickets. Im Rahmen der Operation wird sichergestellt, dass die VWS der zu testenden Anlagenkomponente ansprechbar bzw. für einen Testprozess bereit ist. Abb. 5.6 illustriert die Konstruktion der Klasse `TestObject`. Für weitere Informationen siehe Tabelle 5.46.

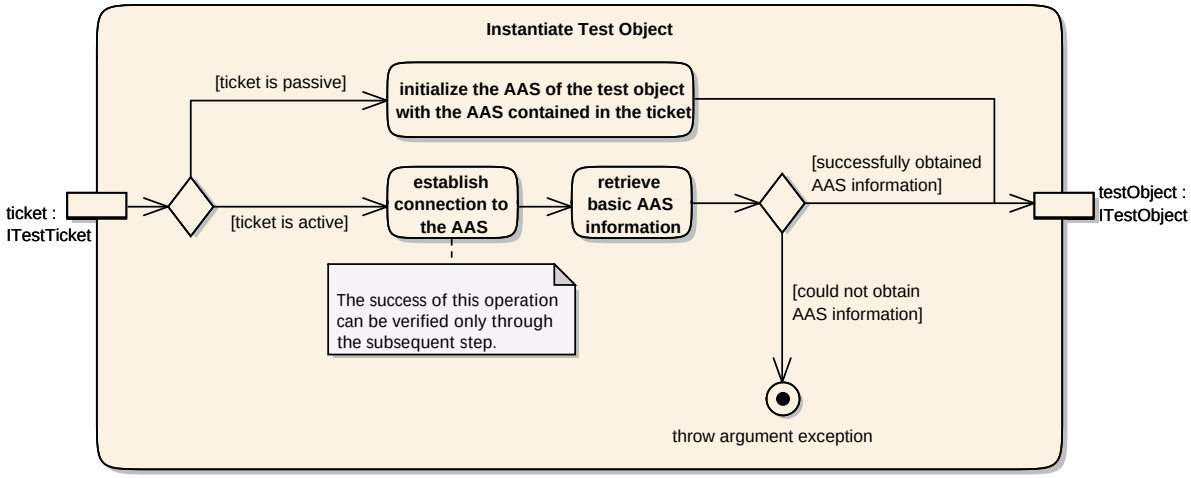


Abbildung 5.6: Aktivitätsdiagramm - Instanziierung der Klasse `TestObject`

Durch den Einsatz der *TicketBuilder*-Komponente wird die Übergabe von ungültigen Testrezepten ausgeschlossen. Für den Fall, dass das Ticket gültig ist, jedoch keine VWS auf die spezifizierte Adresse aktiv ist, wird, wie durch TC.TO03 definiert, eine Ausnahme geworfen.

Auswertung

Dass die vorgestellte Implementierung den Anforderungen der Testszenarien entspricht, wird durch Abb. 5.7 nachgewiesen.

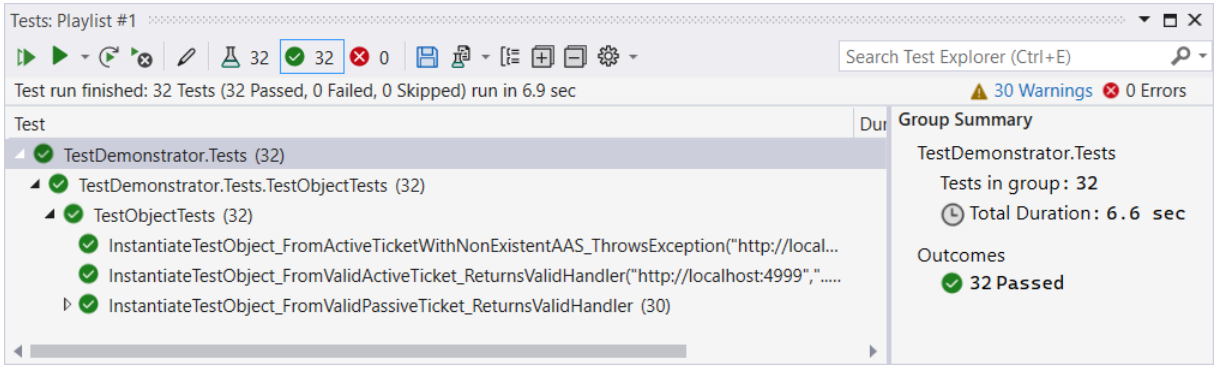


Abbildung 5.7: Testergebnisse - *Test Object*

5.3.5 Test Runner

Der *Test Runner* gilt als zentrale Komponente im Rahmen der Softwarearchitektur. Durch dieses Modul findet die Integration eines Test-Frameworks statt, was als Grundlage für die Durchführung der Test-Pipeline dient. Das Interface `ITestRunner` charakterisiert sich durch eine einzige Methode, die in Abhängigkeit von der ausgewählten Testbibliothek unterschiedlich implementiert werden kann.

Funktionalität

Komponente	Test Runner
Signatur	*Test-Suite* ausführen `XmlNode RunTests(ITestObject testObject, Type type)`
Input	• Polymorph der Schnittstelle `ITestObject` • Der Typ einer Klasse, die als *Test-Suite* definiert ist
Return	Testbericht als XML-Dokument
Beschreibung	Die Methode führt die im Rahmen der *Test-Suite* enthaltenen Tests auf das Testobjekt durch und generiert ein Testbericht als Ergebnis
Testszenarien	TC.TR01, TC.TR02, TC.TR03 (siehe Tabellen 5.51, 5.52 und 5.53)

Tabelle 5.50: Funktionsbeschreibung: *Test Runner* - `RunTests`

Testszenarien

Identifikation	TC.TR01
Testszenario	Ausführung aller Tests einer Testklasse
Testgruppe	`Demonstrator.Tests.TestRunnerTests`
Funktionalität	`XmlNode RunTests(ITestObject testObject, Type testClass)`
Funktionsbeschreibung	siehe Tabelle 5.50
Beschreibung	Durchführung einer Reihe von Testfällen
Vorgehen	Erstellung einer Testklasse mit einem positiven und einem negativen Testfall und darauffolgende Durchführung eines Funktionsaufrufs mit dem Typ der Testklasse als Parameter. Der Funktion soll ein gültiges Polymorph der `ITestObject`-Schnittstelle übergeben werden.
Erwartung	Durchführung der Testfälle mit Rückgabe des entsprechenden Testprotokolls als XML-Datenstruktur.

Tabelle 5.51: Teststrategie: *Test Runner* - TC.TR01

Identifikation	TC.TR02
Testszenario	Erfolglose Testdurchführung bei inkorrektem Funktionsparameter vom Typ `ITestObject`
Testgruppe	`Demonstrator.Tests.TestRunnerTests`
Funktionalität	`XmlNode RunTests(ITestObject testObject, Type testClass)`
Funktionsbeschreibung	siehe Tabelle 5.50
Beschreibung	Aufgrund inkorrekter Parameter erfolglose Initiierung eines Testdurchführungsvorgangs
Vorgehen	Funktionsaufruf mit ungültigem (`null`) Polymorph der Schnittstelle `ITestObject`.
Erwartung	Die Methode wirft eine Ausnahme vom Typ `NullArgumentException`

Tabelle 5.52: Teststrategie: *Test Runner* - TC.TR02

Identifikation	TC.TR03
Testszenario	Erfolglose Testdurchführung bei inkorrektem Funktionsparameter vom Typ `Type`
Testgruppe	`Demonstrator.Tests.TestRunnerTests`
Funktionalität	`XmlNode RunTests(ITestObject testObject, Type testClass)`
Funktionsbeschreibung	siehe Tabelle 5.50
Beschreibung	Aufgrund inkorrekter Parameter erfolglose Initiierung eines Testdurchführungsvorgangs
Vorgehen	Funktionsaufruf mit Übergabe einer Kalsse, die nicht als *Test-Suite* definiert ist, oder deren Wert `null` entspricht.
Erwartung	Die Methode wirft eine Ausnahme vom Typ `NullArgumentException`

Tabelle 5.53: Teststrategie: *Test Runner* - TC.TR03

Implementierung

Im Rahmen der Konzeptprüfung wird ein *Test Runner* entwickelt, der für die Ausführung von Testklassen im Kontext des *NUnit*-Testframeworks verantwortlich ist und durch die Klasse `NUnitRunner` implementiert wird. Laut Dokumentation[3] bietet *NUnit* drei APIs, die eine Testdurchführung ermöglichen. Die *Test Engine API* gilt als Schnittstelle zum Softwaremodul der höchsten Softwareebene des Testframeworks, die eine programmiertechnische Initiierung von Testabläufen ermöglicht. Aus diesem Grund wird diese von den Entwicklern der Softwareumgebung zum Zweck der Integration von *NUnit* in anderen Applikationsdomänen empfohlen.

Eine Analyse der (Test-)Klassen innerhalb der GitHub-Repository[4] der *Test Engine* führt zur Aufstellung des durch Abb. 5.8 dargestellten Klassendiagramms. Dieses beschreibt die wichtigsten Charakteristiken der einzelnen *NUnit*-Komponenten, durch welche die Softwarelösung mit den Funktionalitäten eines Testframeworks ausgestattet werden kann. Als primäre Komponenten gelten die Klassen bzw. Schnittstellen `TestEngineActivator`, `ITestEngine` und `ITestRunner`. Bei der letzteren handelt es sich um eine interne Schnittstelle der Testbibliothek und nicht das `ITestRunner`-Interface, das im Laufe der Konzepterstellung synthetisiert wurde. Dadurch, dass die zwei Schnittstellen in unterschiedlichen Namensräumen definiert werden, ist ein Konflikt aufgrund gleicher Bezeichner ausgeschlossen.

[3] `https://docs.nunit.org/articles/nunit/technical-notes/nunit-internals/NUnit-APIs.html`
[4] `https://github.com/nunit/nunit-console/tree/main/src/NUnitEngine`

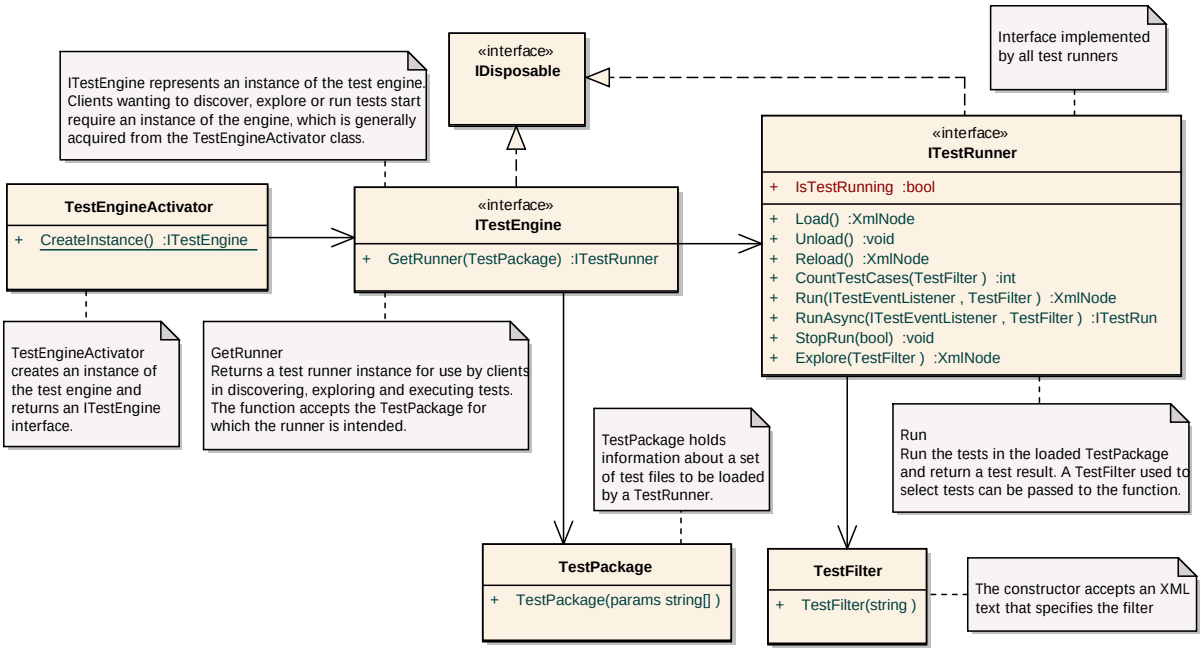


Abbildung 5.8: Eingesetzte Komponenten der *NUnit Test Engine API*

Die statische Klasse `TestEngineActivator` ermöglicht durch die Methode `CreateInstance()` die Konstruktion einer konkreten Instanz der Schnittstelle `ITestEngine`. Dieses Interface implementiert die Methode `GetRunner()`, welche eine Instanz der Klasse `TestPackage` erwartet und ein Polymorph der *NUnit*-Schnittstelle `ITestRunner` zurückgibt. Bei der Instanziierung der Klasse `TestPackage` ist die Spezifikation der Softwareeinheit (sog. Assembly) erforderlich, welche die durchzuführenden Tests enthält. Die Methoden des resultierenden *Test Runner* beziehen sich also nur auf die Tests innerhalb des spezifizierten *Test Package*. Da ein Assembly viele *Test-Suites* beinhalten kann, eignet sich für die zielorientierte Ausführung bestimmter Testklassen die Übergabe von `TestFilter`-Instanzen als Argument der Methode `Run()`. Dadurch lassen sich die Testszenarien einer konkreten Testklasse als Gegenstand der Testdurchführung spezifizieren.

Auswertung

Abb. 5.9 veranschaulicht die Ergebnisse der Verifikation der Komponente *Test Runner*.

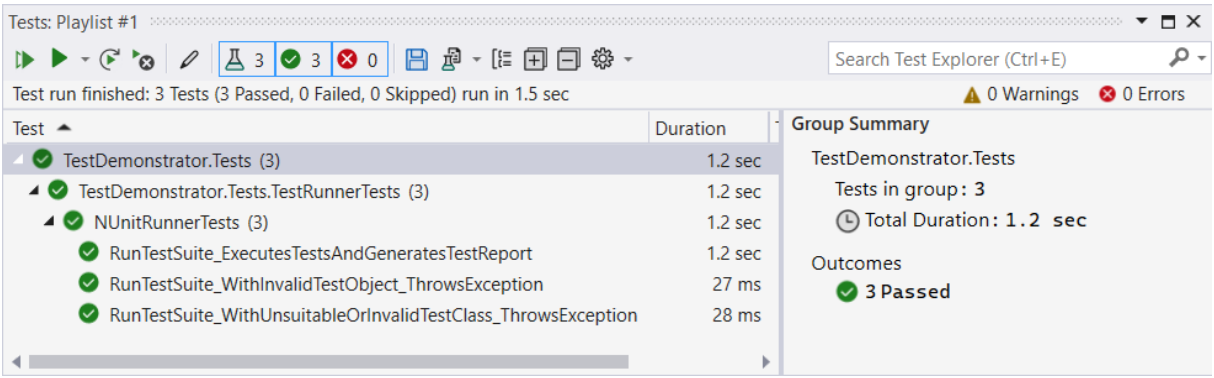


Abbildung 5.9: Testergebnisse - *Test Runner*

Die im Rahmen des Testfalls *TC.TR01* (siehe Tabelle 5.51) ausgeführten Tests sind durch Abb. 5.10 dargestellt.

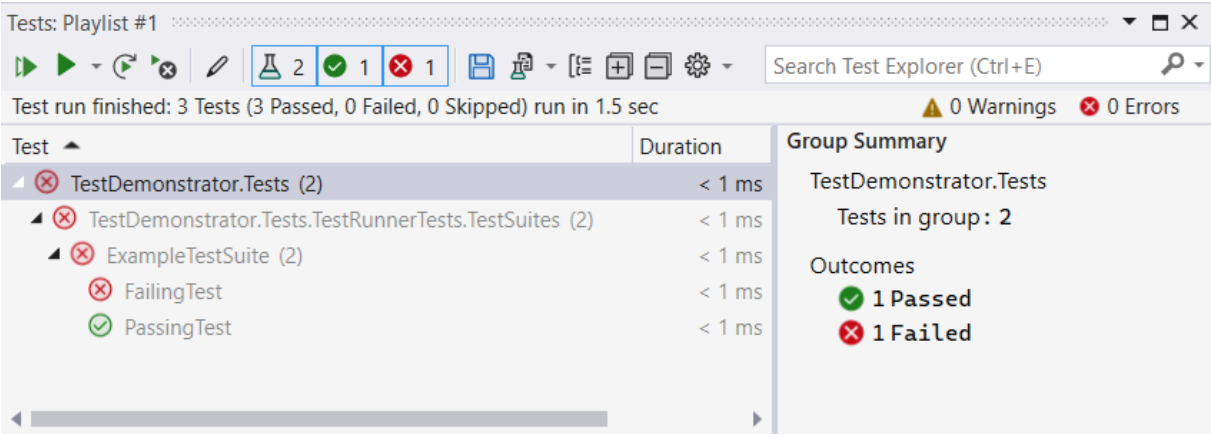


Abbildung 5.10: Testergebnisse - *Test Runner* TC.TR01

5.3.6 Ticket Repository

Durch die *Repository* Komponente findet, wie konzipiert, das Hochladen, Aktualisieren, Löschen und Abrufen von Testrezepten statt. Im Rahmen der Konzeptprüfung wird das Ticketverzeichnis auf Basis einer SQL-Datenbank (DB) implementiert. Eine detaillierte Beschreibung der Dienste des Moduls findet nachfolgend in tabellarischer Form statt.

Funktionalität

Komponente	Ticket DB Repository
Signatur	Hochladen eines Test-Tickets `void Upload(ITestTicket ticket)`
Input	Polymorph der Schnittstelle `ITestTicket`. Der Eingangsparameter repräsentiert das Ticket, das hochgeladen werden muss.
Beschreibung	Die Methode bildet das akzeptierte Argument in einer Entität vom Typ `TestTicketEntity` ab. Die Entität wird als Eintrag in der Datenbanktabelle durch die Verwendung der entsprechenden `DbContext`-Klasse gespeichert.
Testszenarien	TC.RE01 und TC.RE02 (siehe Tabellen 5.61 und 5.62)

Tabelle 5.54: Funktionsbeschreibung: *Ticket Repository* - `Upload`

Komponente	Ticket DB Repository
Signatur	Löschen eines Test-Tickets `void Delete(ITestTicket ticket)`
Input	Polymorph der Schnittstelle `ITestTicket`. Der Eingangsparameter repräsentiert das Ticket, das gelöscht werden muss.
Beschreibung	Die Methode bildet das akzeptierte Argument in einer Entität vom Typ `TestTicketEntity` ab. Die Entität wird von der entsprechenden Datenbank-Tabelle gelöscht.
Testszenarien	TC.RE03 (siehe Tabelle 5.63)

Tabelle 5.55: Funktionsbeschreibung: *Ticket Repository* - `Delete`

Komponente	Ticket DB Repository
Signatur	Aktualisieren eines Test-Tickets `void Update(ITestTicket ticket)`
Input	Polymorph der Schnittstelle `ITestTicket`. Der Eingangsparameter repräsentiert das aktualisierte Ticket.
Beschreibung	Die Methode bildet das Argument in einer Entität vom Typ `TestTicketEntity` ab. Anhand der Identifikationskennzeichnung, die im Kontext der Datenbank als Primärschlüssel gilt, wird der entsprechende Tabelleneintrag, wenn vorhanden, identifiziert und extrahiert. Im nächsten Schritt erfolgt das Überschreiben des alten Tickets zusammen mit der Initialisierung des Ticket-Attributs `UpdatedAt`.
Testszenarien	TC.RE04 und TC.TE05 (siehe Tabellen 5.64 und 5.65)

Tabelle 5.56: Funktionsbeschreibung: *Ticket Repository* - `Update`

Komponente	Ticket DB Repository
Signatur	Löschen eines Test-Tickets anhand seiner Identifikationskennzeichnung `void DeleteTicketById(Guid id)`
Input	Identifikationskennzeichnung des zu löschenden Test-Rezeptes als `Guid`
Beschreibung	Die Methode entfernt das Test-Rezept, dessen Identifikationsnummer mit dem übergebenen Identifikator übereinstimmt, aus der Tabelle.
Testszenarien	TC.RE06, TC.RE07 (siehe Tabellen 5.66 und 5.67)

Tabelle 5.57: Funktionsbeschreibung: *Ticket Repository* - `DeleteTicketById`

Komponente	Ticket DB Repository
Signatur	Abrufen eines Test-Tickets anhand seiner Identifikationskennzeichnung `ITestTicket RetrieveTicketById(Guid id)`
Input	Identifikationskennzeichnung als `Guid`
Return	Polymorph der Schnittstelle `ITestTicket`
Beschreibung	Die Methode ruft eine Test-Rezept Entität ab, deren Identifikationsnummer mit dem übergebenen Identifikator übereinstimmt. Im nächsten Schritt wird die Entität in eine Instanz der Klasse `TestTicket` umgewandelt und von der Methode zurückgegeben.
Testszenarien	TC.RE08, TC.RE09 (siehe Tabellen 5.68 und 5.69)

Tabelle 5.58: Funktionsbeschreibung: *Ticket Repository* - `RetrieveTicketById`

Komponente	Ticket DB Repository
Signatur	Abrufen eines aktiven Test-Tickets anhand seiner URI-Adresse `ITestTicket RetrieveTicketByUrl(string uri)`
Input	URI-Adresse der VWS des Testobjekts
Return	Polymorph der Schnittstelle `ITestTicket`
Beschreibung	Die Methode ruft eine Test-Rezept Entität ab, deren URI-Adresse mit dem Funktionsargument übereinstimmt. Im nächsten Schritt wird die Entität in eine Instanz der Klasse `TestTicket` umgewandelt und von der Methode zurückgegeben.
Testszenarien	TC.RE10, TC.RE11 (siehe Tabellen 5.70 und 5.71)

Tabelle 5.59: Funktionsbeschreibung: *Ticket Repository* - `RetrieveTicketByUrl`

Komponente	Ticket DB Repository
Signatur	Abrufen aller Test-Tickets `IEnumerable<ITestTicket> RetrieveAllTickets()`
Return	Liste aus Instanzen der Klasse `TestTicket`
Beschreibung	Die Methode ruft alle Entitäten im Verzeichnis ab. Jede Entität wird in eine Instanz der Klasse `TestTicket` umgewandelt. Die Instanzen werden von der Methode als Liste bereitgestellt.
Testszenarien	TC.RE12 (siehe 5.72)

Tabelle 5.60: Funktionsbeschreibung: *Ticket Repository* - `RetrieveAllTickets`

Testszenarien

Identifikation	TC.RE01
Testszenario	Hochladen aktiver Tickets
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void Upload(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.54
Beschreibung	Durch dieses Testszenario wird das Hochladen von Testrezepten für aktive Verwaltungsschalen verifiziert.
Vorgehen	Nach der Realisierung einer Verbindung zu einer Test-Datenbank soll die Erstellung eines aktiven Testrezeptes durch den Einsatz der bereits verifizierten Komponenten *Descriptor Factory* und *Ticket Builder* erfolgen. Im nächsten Schritt findet ein Aufruf der Funktion mit dem erstellten Ticket als Argument und eine darauffolgende Überprüfung, ob die Anzahl der Tickets im Verzeichnis mit der Anzahl der hochgeladenen Tickets korrespondiert. Nachfolgend werden die Werte der Instanzattributen mit den Werten der aktuellen Tabellenentität verglichen.
Erwartung	Das Aufrufen der Methode führt zum Inkrementieren der Anzahl von Tabellenentitäten. Die Eigenschaften der Entität korrespondieren mit den Intanzattributen der Ticket-Instanz.

Tabelle 5.61: Teststrategie: *Ticket Repository* - TC.RE01

Identifikation	TC.RE02
Testszenario	Hochladen passiver Tickets
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void Upload(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.54
Beschreibung	Durch dieses Testszenario wird das Hochladen von Testrezepten für passive Verwaltungsschalen verifiziert.
Vorgehen	Siehe Tabelle 5.61. Im Rahmen dieses zusätzlichen Testfalls wird ein passives Ticket konstruiert.
Erwartung	Siehe Tabelle 5.61.

Tabelle 5.62: Teststrategie: *Ticket Repository* - TC.RE02

Identifikation	TC.RE03
Testszenario	Löschen bestehender Tickets
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void Delete(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.55
Beschreibung	Durch dieses Testszenario wird das Löschen von bereits hochgeladenen Testrezepten verifiziert.
Vorgehen	Zum Zweck der Verifikation ist an erster Stelle das Hochladen eines Testrezeptes erforderlich. Als Voraussetzung dafür dient die Realisierung einer Verbindung zu einer Test-Datenbank. Im nächsten Schritt findet ein Methodenaufruf mit der hochgeladenen Ticket-Instanz als Argument statt.
Erwartung	Die Anzahl der Testrezepten im Verzeichnis soll dem Wert `0` entsprechen.

Tabelle 5.63: Teststrategie: *Ticket Repository* - TC.RE03

Identifikation	TC.RE04
Testszenario	Aktualisierung bestehender Tickets
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void Update(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.56
Beschreibung	Durch dieses Testszenario wird das Aktualisieren von bereits hochgeladenen Testrezepten verifiziert.
Vorgehen	Zum Zweck der Verifikation ist die Realisierung einer Verbindung zur Test-Datenbank erforderlich. Nur dann kann ein Testrezept entsprechend hochgeladen werden. Nach diesem Vorgang soll eine Änderung eines beliebigen Attributs der lokalen Ticket-Instanz stattfinden und die Methode aufgerufen werden. Als Argument des Aufrufs gilt die aktualisierte Ticket-Instanz.
Erwartung	Das Aufrufen der Methode führt zur Zuweisung des Entitätsattributs mit dem Bezeichner `CreatedAt`. Die Eigenschaften der Entität korrespondieren mit den Intanzattributen der aktualisierten Ticket-Instanz.

Tabelle 5.64: Teststrategie: *Ticket Repository* - TC.RE04

Identifikation	TC.RE05
Testszenario	Aktualisierung von Tickets bei leerem Ticket-Repository
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void Update(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.56
Beschreibung	Durch dieses Testszenario wird sichergestellt, dass die Aktualisierung von Testrezepten, die sich nicht im Verzeichnis befinden, zu einer Ausnahmesituation führt.
Vorgehen	Nach der Erstellung eines Testrezeptes durch den Einsatz der Komponenten *Descriptor Factory* und *Ticket Builder* soll die von diesem Testfall adressierte Funktionalität aufgerufen werden. Die kreierte Instanz dient dann als Eingangsparameter der Methode.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme vom Typ `ArgumentException`.

Tabelle 5.65: Teststrategie: *Ticket Repository* - TC.RE05

Identifikation	TC.RE06
Testszenario	Löschen eines bestehenden Tickets anhand seiner Identifikationskennzeichnung
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void DeleteTicketById(Guid id)`
Funktionsbeschreibung	siehe Tabelle 5.57
Beschreibung	Durch dieses Testszenario wird die Funktionalität zum Löschen eines Testrezeptes anhand der Übergabe der entsprechenden Identifikationskennzeichnung verifiziert.
Vorgehen	Zum Zweck der Verifikation ist die Realisierung einer Verbindung zur Test-Datenbank erforderlich. Nach der Erstellung eines Testrezeptes durch den Einsatz der Komponenten *Descriptor Factory* und *Ticket Builder* und dem darauffolgenden Aufruf der `Upload`-Methode soll die von diesem Testfall adressierte Funktionalität aufgerufen werden. Das `Id`-Attribut der kreierten Instanz dient als Eingangsparameter der Methode.
Erwartung	Die Anzahl der Testrezepten im Verzeichnis soll dem Wert `0` entsprechen.

Tabelle 5.66: Teststrategie: *Ticket Repository* - TC.RE06

Identifikation	TC.RE07
Testszenario	Löschen eines Tickets anhand seines Identifikators bei leerem Ticketverzeichnis
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`void DeleteTicketById(Guid id)`
Funktionsbeschreibung	siehe Tabelle 5.57
Beschreibung	Durch dieses Testszenario wird sichergestellt, dass das Löschen eines Testrezeptes anhand seiner Kennzeichnung und bei leerem Ticketverzeichnis zu einer Ausnahmesituation führt.
Vorgehen	Nach der Erstellung eines Testrezeptes durch den Einsatz der Komponenten *Descriptor Factory* und *Ticket Builder* und dem darauffolgenden Aufruf der `Upload`-Methode soll die von diesem Testfall adressierte Funktionalität aufgerufen werden. Das `Id`-Attribut der kreierten Instanz dient als Eingangsparameter der Methode.
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.67: Teststrategie: *Ticket Repository* - TC.RE07

Identifikation	TC.RE08
Testszenario	Abrufen eines Tickets anhand seines Identifikators
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`ITestTicket RetrieveTicketById(Guid id)`
Funktionsbeschreibung	siehe Tabelle 5.58
Beschreibung	Durch diesen Testfall wird die Funktionalität zum Abrufen eines bestehenden Testrezeptes aus dem Ticketverzeichnis verifiziert.
Vorgehen	Nach der Erstellung eines Testrezeptes durch den Einsatz der Komponenten *Descriptor Factory* und *Ticket Builder* und dem darauffolgenden Aufruf der `Upload`-Methode soll die von diesem Testfall adressierte Funktionalität aufgerufen werden. Das `Id`-Attribut der kreierten Instanz dient als Eingangsparameter der Methode.
Erwartung	Die Methode gibt ein gültiges Polymorph der Schnittstelle `ITestTicket` zurück. Die Attribute der abgerufenen Instanz korrespondieren mit denen des Hochgeladenen Objektes.

Tabelle 5.68: Teststrategie: *Ticket Repository* - TC.RE08

Identifikation	TC.RE09
Testszenario	Abrufen eines Tickets anhand seines Identifikators bei leerem Ticketverzeichnis
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`ITestTicket RetrieveTicketById(Guid id)`
Funktionsbeschreibung	siehe Tabelle 5.58
Beschreibung	Durch diesen Testfall wird sichergestellt, dass die Übergabe eines Identifikators, mit welchem kein Testrezept assoziiert werden kann, zu einer Ausnahmesituation führt.
Vorgehen	Methodenaufruf bei leerem Ticket-Verzeichnis.
Erwartung	Die Methoden wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.69: Teststrategie: *Ticket Repository* - TC.RE09

Identifikation	TC.RE10
Testszenario	Abrufen eines aktiven Tickets anhand seiner URI-Adresse
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`ITestTicket RetrieveTicketById(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.59
Beschreibung	Durch diesen Testfall wird die Funktionalität zum Abrufen eines bestehenden Testrezeptes anhand der Übergabe einer URI-Adresse verifiziert.
Vorgehen	Nach der Erstellung eines Testrezeptes durch den Einsatz der Komponenten *Descriptor Factory* und *Ticket Builder* und dem darauffolgenden Aufruf der `Upload`-Methode soll die von diesem Testfall adressierte Funktionalität aufgerufen werden. Das `Uri`-Attribut der kreierten Ticket-Instanz dient als Eingangsparameter der Methode.
Erwartung	Die Methode gibt ein gültiges Polymorph der Schnittstelle `ITestTicket` zurück. Die Attribute der abgerufenen Instanz korrespondieren mit denen des Hochgeladenen Objektes.

Tabelle 5.70: Teststrategie: *Ticket Repository* - TC.RE10

Identifikation	TC.RE11
Testszenario	Abrufen eines aktiven Tickets anhand seiner URI-Adresse bei leerem Ticketverzeichnis
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`ITestTicket RetrieveTicketByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.59
Beschreibung	Durch diesen Testfall wird sichergestellt, dass die Übergabe einer URI-Adresse, mit welchem kein Testrezept assoziiert werden kann, zu einer Ausnahmesituation führt.
Vorgehen	Methodenaufruf bei leerem Ticket-Verzeichnis.
Erwartung	Die Methoden wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.71: Teststrategie: *Ticket Repository* - TC.RE11

Identifikation	TC.RE12
Testszenario	Abrufen aller Tickets aus dem Verzeichnis
Testgruppe	`Demonstrator.Tests.TicketRepositoryTests`
Funktionalität	`IEnumerable<ITestTicket> RetrieveAllTickets()`
Funktionsbeschreibung	siehe Tabelle 5.60
Beschreibung	Im Rahmen des Testfalls soll das erfolgreiche Auslesen aller Entitäten im Verzeichnis, ihr korrektes Mapping zu Instanzen der Klasse `TestTicket` und Bereitstellung in Form einer linearen Datenstruktur bestätigt werden.
Vorgehen	Zum Zweck des Testfalls sollen mehrere Tickets durch die Komponenten *Descriptor Factory* und *Ticket Builder* im Verzeichnis hochgeladen werden. Im nächsten Schritt findet ein Methodenaufruf statt.
Erwartung	Die Anzahl der bereitgestellten Test-Rezepten korrespondiert mit der Anzahl der hochgeladenen Tickets. Die Werte der Instanzattribute der zurückgegebenen Ticket-Instanzen entsprechen den Werten der Instanzen, die hochgeladen wurden.

Tabelle 5.72: Teststrategie: *Ticket Repository* - TC.RE12

Implementierung

Die Realisierung der Datenbank-orientierten *Ticket Repository* Komponente findet, wie spezifiziert, anhand der *Code First* Methode statt. Die Datenbank wird also nicht durch SQL-Anfragen, auf Basis eines SQL-Skriptes, sondern durch den Einsatz eines *Object Relational Mapping* (ORM) Frameworks realisiert, welches eine Realisierung der Informationsstruktur auf Basis einer objektorientierten Programmiersprache ermöglicht. Zum Einsatz kommt das ORM-Framework für .NET - *Entity Framework Core*[5]. Die grundlegende Klasse bei der Integration dieser Softwareumgebung entspricht dem abstrakten Typ `DbContext`. Die softwaretechnische Erstellung einer Datenbank umfasst die Implementierung einer Klasse, die als Subklasse dieses Datentyps dient. Diese Subklasse repräsentiert die Datenbank mit ihrer Entitäten. Zum Zweck der Spezifikation des Datenbankservers erfolgt das Überladen der Methode `OnConfiguring(DbContextOptionsBuilder optionsBuilder)`. Jede Datenbanktabelle wird als Instanzattribut der Subklasse deklariert. Da eine Tabelle mehrere Einträge beinhalten kann, entspricht dieses Instanzattribut einer linearen Datenstruktur aus Entitäten (`DbSet<TEntity>`). Jede Entität wird vom Anwender in Form einer Klasse implementiert.

[5] `https://learn.microsoft.com/en-us/ef/core/`

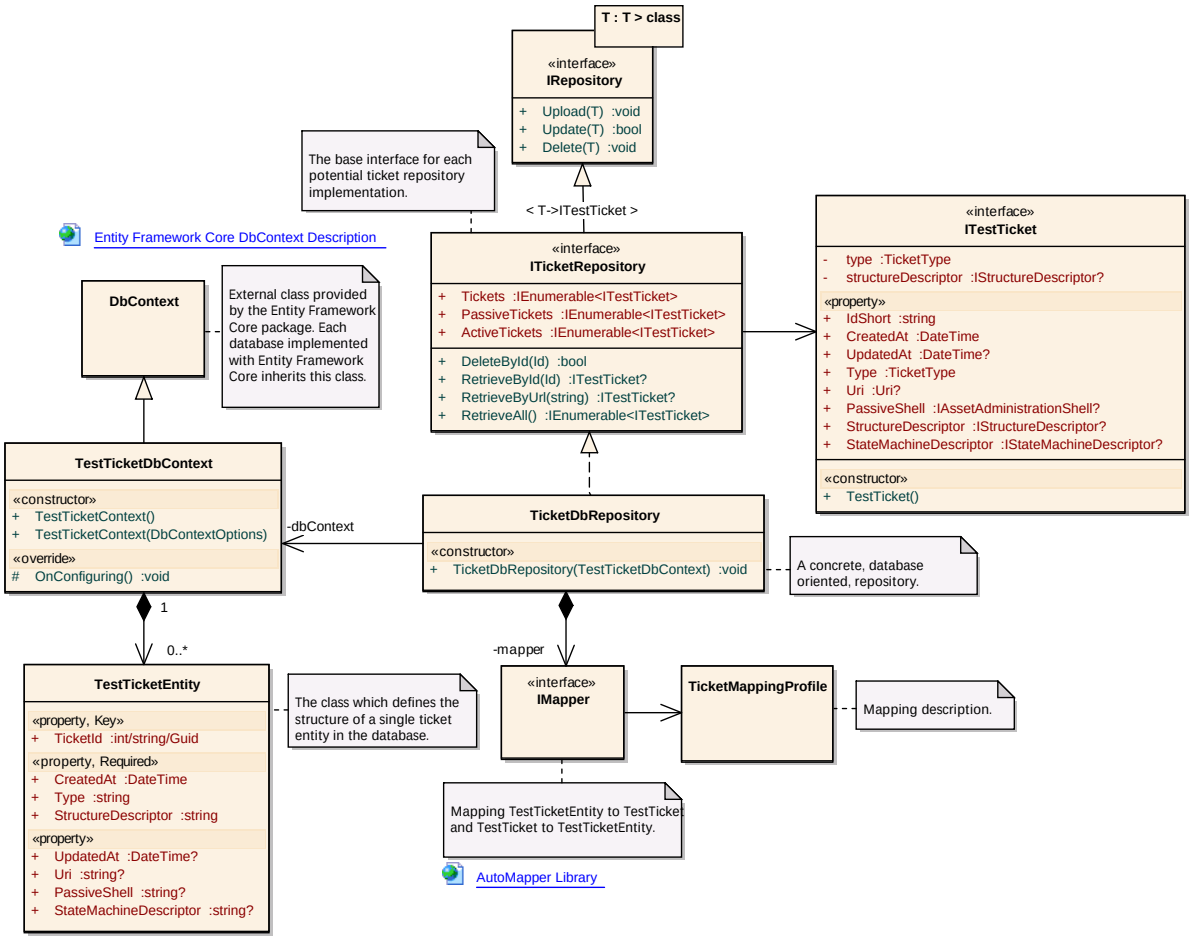


Abbildung 5.11: Klassendiagramm: *Ticket Database Repository*

In Zusammenhang mit der Applikation charakterisiert sich die Klasse, welche die Datenbank repräsentiert, durch den Bezeichner `TestTicketDbContext`. Jede Datenbank-Entität gilt als Instanz der Klasse `TestTicketEntity`. Zum Zweck der losen Kopplung zwischen der konkreten Implementierung des Verzeichnisses und der Softwarearchitektur, welche zur Repräsentation der Testrezepte die Scnittstelle `ITestTicket` verwendet, wandelt die Komponente jede ausgelesene Entität in einem Polymorph der Schnittstelle `ITestTicket` um. Es findet also eine Kontextualisierung statt. Hierfür wird die Bibliothek *AutoMapper*[6] verwendet. Die Spezifikation, wie das Mapping durchgeführt werden soll, findet durch die Implementierung des entsprechenden Profils statt. Nach dem Mapping wird die Kollektion aus Test-Tickets durch die Erweiterungsmethoden[7] der *LINQ*[8]-Bibliothek weiterverarbeitet. Ein Überblick über die konkrete Implementierung der im Rahmen der Konzeption festgelegten Schnittstellen anhand des eingesetzten ORM-Frameworks wird durch Abb. 5.11 dargestellt.

[6] `https://github.com/AutoMapper/AutoMapper`
[7] `https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods`
[8] `https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/`

Auswertung

Abb. 5.12 veranschaulicht die Ergebnisse aus der Verifikation der Komponente *Ticket (Database) Repository*.

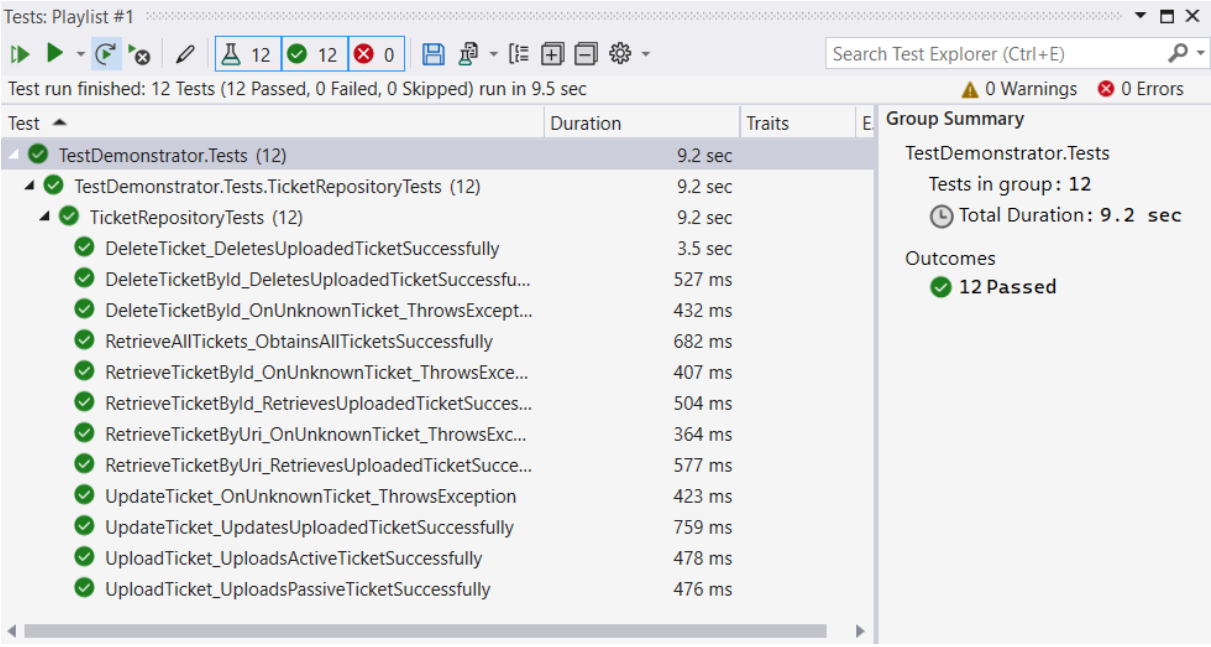


Abbildung 5.12: Testergebnisse - *Ticket (Database) Repository*

5.3.7 Test Object Provider

Wie beschrieben, besteht die Aufgabe des Moduls *Test Object Provider* in dem Auslesen der Tickets aus der *Repository*-Komponente und der darauffolgenden Konstruktion von Testobjekten anhand der Ticket-Informationen. Die Komponente charakterisiert sich durch die folgenden Dienste.

Funktionalität

Komponente	Test Object Provider
Signatur	`TestObjectProvider` instanziieren `TestObjectProvider(ITicketRepository ticketRepository)`
Input	Polymorph der Schnittstelle `ITicketRepository`
Return	Polymorph der Schnittstelle `ITestObjectProvider`
Beschreibung	Die Methode erstellt eine Instanz der Klasse `TestObjectProvider`, welche über einen lesenden Zugriff zum Ticketverzeichnis verfügt.
Testszenarien	TC.TOP01 (siehe Tabelle 5.82)

Tabelle 5.73: Funktionsbeschreibung: *Test Object Provider* - Konstruktor

Komponente	Test Object Provider
Signatur	Testobjekt anhand eines Testrezeptes abrufen `ITestObject RetrieveTestObject(ITestTicket ticket)`
Input	Polymorph der Schnittstelle `ITestTicket`
Return	Polymorph der Schnittstelle `ITestObject`
Beschreibung	Die Methode erstellt eine Instanz der Klasse `TestObject` (siehe 5.3.4) anhand der im Ticket enthaltenen Informationen.
Testszenarien	TC.TOP02 (siehe Tabelle 5.83)

Tabelle 5.74: Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObject`

Komponente	Test Object Provider
Signatur	Testobjekt für jedes Ticket innerhalb des Repository instanziieren `ITestObject RetrieveTestObject(ITestTicket ticket)`
Return	Liste aus Instanzen, die die Schnittstelle `ITestObject` implementieren
Beschreibung	Die Methode erstellt eine Instanz der Klasse `TestObject` (siehe 5.3.4) für jedes Ticket im Verzeichnis. Die Testobjekte werden als abstrakte Datenstruktur zurückgegeben.
Testszenarien	TC.TOP03 (siehe Tabelle 5.84)

Tabelle 5.75: Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObject`

Komponente	Test Object Provider
Signatur	Testobjekt für jedes passive Ticket innerhalb des Repository instanziieren `IEnumerable<ITestObject> RetrievePassiveTestObjects()`
Return	Liste aus Instanzen, die die Schnittstelle `ITestObject` implementieren und sich durch ein passives Ticket charakterisieren. Die Instanzen gelten im Kontext der Softwarearchitektur als passive Testobjekte.
Beschreibung	Die Methode liest alle passiven Tickets aus dem Verzeichnis aus. Für jedes passive Ticket findet die Instanziierung der Klasse `TestObject` statt. Alle Instanzen werden als abstrahierte lineare Datenstruktur von der Funktion zurückgegeben.
Testszenarien	TC.TOP03 (siehe Tabelle 5.84)

Tabelle 5.76: Funktionsbeschreibung: *Test Object Provider* - `RetrievePassiveTestObjects`

Komponente	Test Object Provider
Signatur	Testobjekt für jedes aktive Ticket innerhalb des Repository instanziieren `IEnumerable<ITestObject> RetrieveActiveTestObjects()`
Return	Liste aus Instanzen, die die Schnittstelle `ITestObject` implementieren und sich durch ein aktives Ticket charakterisieren. Die Instanzen gelten im Kontext der Softwarearchitektur als aktive Testobjekte.
Beschreibung	Die Methode liest alle aktiven Tickets aus dem Verzeichnis aus. Für jedes aktive Ticket findet die Instanziierung der Klasse `TestObject` statt. Alle Instanzen werden als abstrahierte lineare Datenstruktur von der Funktion zurückgegeben.
Testszenarien	TC.TOP04 (siehe Tabelle 5.85)

Tabelle 5.77: Funktionsbeschreibung: *Test Object Provider* - `RetrieveActiveTestObjects`

Komponente	Test Object Provider
Signatur	Testobjekt für jedes verarbeitete Ticket innerhalb des Repository instanziieren `IEnumerable<ITestObject> RetrieveProcessedTestObjects()`
Return	Liste aus Instanzen, die die Schnittstelle `ITestObject` implementieren und bereits getestet wurden, deren Ticket also sich durch ein `TestProtocol` charakterisiert.
Beschreibung	Die Methode liest alle Tickets aus dem Verzeichnis aus, die ein Testbericht beinhalten. Für jedes Ticket findet die Instanziierung der Klasse `TestObject` statt. Alle Instanzen werden als abstrahierte lineare Datenstruktur von der Funktion zurückgegeben.
Testszenarien	TC.TOP05 (siehe Tabelle 5.86)

Tabelle 5.78: Funktionsbeschreibung: *Test Object Provider* - `RetrieveProcessedTestObjects`

Komponente	Test Object Provider
Signatur	Testobjekt für jedes unverarbeitete Ticket innerhalb des Repository instanziieren `IEnumerable<ITestObject> RetrieveUnprocessedTestObjects()`
Return	Liste aus Instanzen, die die Schnittstelle `ITestObject` implementieren und z. Zt. nicht getestet wurden, deren Ticket also sich durch kein `TestProtocol` charakterisiert.
Beschreibung	Die Methode liest alle Tickets aus dem Verzeichnis aus, die kein Testbericht beinhalten. Für jedes Ticket findet die Instanziierung der Klasse `TestObject` statt. Alle Instanzen werden als abstrahierte lineare Datenstruktur von der Funktion zurückgegeben.
Testszenarien	TC.TOP06 (siehe Tabelle 5.87)

Tabelle 5.79: Funktionsbeschreibung: *Test Object Provider* - `RetrieveUnprocessedTestObjects`

Komponente	Test Object Provider
Signatur	Testobjekt anhand einer Ticket-Kennung abrufen `ITestObject RetrieveTestObjectById(Guid id)`
Input	Ticket-Kennung als Instanz der Klasse `Guid`
Return	Polymorph der Schnittstelle `ITestObject`
Beschreibung	Ticket abrufen, dessen Kennung mit dem Eingangsparameter übereinstimmt, und darauf basierend eine Instanz der Klasse `TestObject` erstellen und zurückgeben.
Testszenarien	TC.TOP07, TC.TOP08 (siehe Tabellen 5.88 und 5.89)

Tabelle 5.80: Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObjectById`

Komponente	Test Object Provider
Signatur	Testobjekt anhand einer URI-Adresse abrufen `ITestObject RetrieveTestObjectByUrl(string url)`
Input	Die URI-Adresse der VWS des Testobjekts
Return	Polymorph der Schnittstelle `ITestObject`
Beschreibung	Ticket abrufen, dessen URI-Attributwert mit dem Eingangsparameter übereinstimmt, und darauf basierend eine Instanz der Klasse `TestObject` erstellen und zurückgeben.
Testszenarien	TC.TOP09, TC.TOP10 (siehe Tabellen 5.90 und 5.91)

Tabelle 5.81: Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObjectByUrl`

Testszenarien

Identifikation	TC.TOP01
Testszenario	Passives Testobjekt anhand eines hochgeladenen Testrezeptes abrufen
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`ITestObject RetrieveTestObject(ITestTicket ticket)`
Funktionsbeschreibung	siehe Tabelle 5.74
Beschreibung	Im Rahmen des Testfalls wird die erfolgreiche Instanziierung eines Testobjekts nach Übergabe eines mithilfe der *Ticket Builder* Komponente erstellten Testrezeptes überprüft.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank soll ein aktives bzw. passives Ticket erstellt und durch die *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt ist die von diesem Testfall adressierte Methode aufzurufen. Als Argument des Aufrufs gilt die erstellte Ticket-Instanz.
Erwartung	Die Methode gibt ein gültiges Polymorph der Schnittstelle `ITestObject` zurück.

Tabelle 5.82: Teststrategie: *Test Object Provider* - TC.TOP01

Identifikation	TC.TOP02
Testszenario	Bereitstellung eines Testobjekts für jedes Ticket im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`IEnumerable<ITestObject> RetrieveTestObjects()`
Funktionsbeschreibung	siehe Tabelle 5.75
Beschreibung	Im Rahmen des Testfalls wird die erfolgreiche Instanziierung von Testobjekten für alle Tickets im Verzeichnis sichergestellt.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank sollen mehrere Testrezepten auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode.
Erwartung	Die Anzahl der zurückgegebenen Testobjekte stimmt mit der Anzahl der hochgeladenen Tickets überein.

Tabelle 5.83: Teststrategie: *Test Object Provider* - TC.TOP02

Identifikation	TC.TOP03
Testszenario	Erstellung von Testobjekten für alle passiven Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`IEnumerable<ITestObject> RetrievePassiveTestObjects()`
Funktionsbeschreibung	siehe Tabelle 5.76
Beschreibung	Im Rahmen des Testfalls wird die erfolgreiche Instanziierung von Testobjekten für alle passiven Tickets im Verzeichnis sichergestellt.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank sollen mehrere aktive und passive Tickets auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode.
Erwartung	Die Anzahl der zurückgegebenen Testobjekte stimmt mit der Anzahl der hochgeladenen passiven Tickets überein.

Tabelle 5.84: Teststrategie: *Test Object Provider* - TC.TOP03

Identifikation	TC.TOP04
Testszenario	Erstellung von Testobjekten für alle aktiven Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`IEnumerable<ITestObject> RetrieveActiveTestObjects()`
Funktionsbeschreibung	siehe Tabelle 5.77
Beschreibung	Im Rahmen des Testfalls wird die erfolgreiche Instanziierung von Testobjekten für alle aktiven Tickets im Verzeichnis sichergestellt.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank sollen mehrere aktive und passive Tickets auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode.
Erwartung	Die Anzahl der zurückgegebenen Testobjekte stimmt mit der Anzahl der hochgeladenen aktiven Tickets überein.

Tabelle 5.85: Teststrategie: *Test Object Provider* - TC.TOP04

Identifikation	TC.TOP05
Testszenario	Erstellung von Testobjekten für alle bearbeiteten Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`IEnumerable<ITestObject> RetrieveProcessedTestObjects()`
Funktionsbeschreibung	siehe Tabelle 5.78
Beschreibung	Im Rahmen des Testfalls wird die erfolgreiche Instanziierung von Testobjekten für alle bearbeiteten Tickets im Verzeichnis sichergestellt.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank sollen mehrere aktive und passive Tickets auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode.
Erwartung	Die Anzahl der zurückgegebenen Testobjekte stimmt mit der Anzahl der Tickets überein, die sich durch ein Testbericht charakterisieren.

Tabelle 5.86: Teststrategie: *Test Object Provider* - TC.TOP05

Identifikation	TC.TOP06
Testszenario	Erstellung von Testobjekten für alle nicht bearbeiteten Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`IEnumerable<ITestObject> RetrieveUnprocessedTestObjects()`
Funktionsbeschreibung	siehe Tabelle 5.79
Beschreibung	Im Rahmen des Testfalls wird die erfolgreiche Instanziierung von Testobjekten für alle Tickets im Verzeichnis sichergestellt, die keine Testergebnisse beinhalten.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank sollen mehrere Tickets auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode.
Erwartung	Die Anzahl der zurückgegebenen Testobjekte stimmt mit der Anzahl der Tickets überein, die sich durch kein Testbericht charakterisieren.

Tabelle 5.87: Teststrategie: *Test Object Provider* - TC.TOP06

Identifikation	TC.TOP07
Testszenario	Abrufen eines Testobjekts anhand seiner Ticket-Kennung
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`ITestObject? RetrieveTestObjectById(Guid id)`
Funktionsbeschreibung	siehe Tabelle 5.80
Beschreibung	Im Rahmen des Testfalls wird das erfolgreiche Abrufen eines Testobjekts anhand der Kennung seines Testrezeptes überprüft.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank soll ein Testrezept auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode. Als Argument dient die Identifikationskennzeichnung des hochgeladenen Testrezeptes.
Erwartung	Die Methode liefert ein gültiges Polymorph der Schnittstelle `ITestObject` zurück.

Tabelle 5.88: Teststrategie: *Test Object Provider* - TC.TOP07

Identifikation	TC.TOP08
Testszenario	Abrufen eines Testobjekts anhand einer Ticket-Kennung, mit welcher kein Testrezept aassoziierbar ist
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`ITestObject? RetrieveTestObjectById(Guid id)`
Funktionsbeschreibung	siehe Tabelle 5.80
Beschreibung	Im Rahmen des Testfalls wird bestätigt, dass das Abrufen eines Testobjekts anhand einer Identifikationsnummer, für welche kein Testrezept existiert, zu einer Ausnahmesituation führt.
Vorgehen	Aufruf der Methode mit einem eindeutigen `Guid`-Identifikator, mit welchem kein Ticket assoziierbar ist.
Erwartung	Die Methode löst eine Ausnahme vom Typ `ArgumentException` aus.

Tabelle 5.89: Teststrategie: *Test Object Provider* - TC.TOP08

Identifikation	TC.TOP09
Testszenario	Abrufen eines aktiven Testobjekts anhand seiner URI-Adresse
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`ITestObject? RetrieveTestObjectByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.81
Beschreibung	Im Rahmen des Testfalls wird das erfolgreiche Abrufen eines Testobjekts anhand der URI-Adresse seiner VWS überprüft.
Vorgehen	Nach der notwendigen Duchführung der Kommunikationsbeziehung zum Test-Datenbank soll ein aktives Testrezept auf Basis der *Ticket (Database) Repository* Komponente im Verzeichnis hochgeladen werden. Im nächsten Schritt erfolgt ein Aufruf der adressierten Methode. Als Argument dient die URI-Adresse des hochgeladenen Testrezeptes.
Erwartung	Die Methode liefert ein gültiges Polymorph der Schnittstelle `ITestObject` zurück.

Tabelle 5.90: Teststrategie: *Test Object Provider* - TC.TOP09

Identifikation	TC.TOP10
Testszenario	Abrufen eines aktiven Testobjekts anhand einer URI-Adresse, mit welcher kein Ticket assoziierbar ist
Testgruppe	`Demonstrator.Tests.TestObjectProviderTests`
Funktionalität	`ITestObject? RetrieveTestObjectByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.81
Beschreibung	Im Rahmen des Testfalls wird bestätigt, dass das Abrufen eines Testobjekts anhand einer URI-Adresse, mit welcher kein Testrezept assoziiert werden kann, zu einer Ausnahmesituation führt.
Vorgehen	Aufruf der Methode mit einer URI-Adresse, die von keinem Ticket beinhaltet wird.
Erwartung	Die Methode löst eine Ausnahme vom Typ `ArgumentException` aus.

Tabelle 5.91: Teststrategie: *Test Object Provider* - TC.TOP10

Implementierung

Die *Test Object Provider* Komponente basiert auf die Funktionalitäten der Module *Ticket Repository* und *Test Object*. Zum Zweck der Ausnutzung der Dienste des Ticket-Verzeichnisses wird die entsprechende Instanz der *Repository*-Klasse als Argument bei der Instanziierung des Moduls *Test Object Provider* übergeben (*Dependency Inversion*), wodurch der Komponente einen lesenden Zugriff zum Verzeichnis garantiert wird. Für jede durch die Dienste der *Ticket Repository* abgerufene Ticket-Instanz wird eine Instanz der Klasse `ITestObject` kreiert und von der entsprechenden Methode zurückgegeben. Dies ermöglicht die Durchführung einer Test-Pipeline.

Des Weiteren kann die Komponente bei einer potenziellen Umsetzung des Registry-orientierten Testprozesses eingesetzt werden. Hierfür wäre die Implementierung einer weiteren Klasse erforderlich, die zum Zweck dieser Erläuterung als `RegistryTestObjectProvider` bezeichnet wird. Weiterhin ist die Ergänzung der Klasse `TestObject` um einen zweiten Konstruktor erforderlich.

Zum Zweck der Kompatibilität mit den anderen Softwarekomponenten muss der Registry-orientierte *Test Object Provider* die Schnittstelle `ITestObjectProvider` implementieren. Im Rahmen des Konstruktors kann er ein Polymorph der BaSyx-Schnittstelle `IAssetAdministrationShellRegistry`, ein Argument vom Typ `IDescriptorFactory` und eine Instanz der *Ticket Builder* Komponente akzeptieren. Anhand der *Registry*-Instanz lassen sich die Adressen der registrierten Verwaltungsschalen abfragen. Anhand des

zweiten Konstruktors der Klasse `TestObject` lässt sich dann für jede Adresse ein Testobjekt instanziieren.

Der zweite Konstruktor der Klasse `TestObject` soll die URI-Adresse der VWS des Testobjekts und jeweils eine Instanz der Module *Descriptor Factory* und *Ticket Builder* akzeptieren. Durch das erste Argument lässt sich eine Verbindung zur VWS des Testobjekts realisieren. Darauf basierend kann die Sammlung aus Submodell-Elementen abgefragt werden, welche die Ticket-Deskriptoren enthält. Durch das Argument vom Typ `IDescriptorFactory` können die Deskriptoren entsprechend Deserialisiert werden, was als Grundlage für das Extrahieren des Testrezeptes gilt. Dies würde die Durchführung einer Test-Pipeline im Rahmen des Registry-orientierten Testansatzes ermöglichen.

Auswertung

Abb. 5.13 veranschaulicht die Ergebnisse der Verifikation der Komponente *Test Object Provider*.

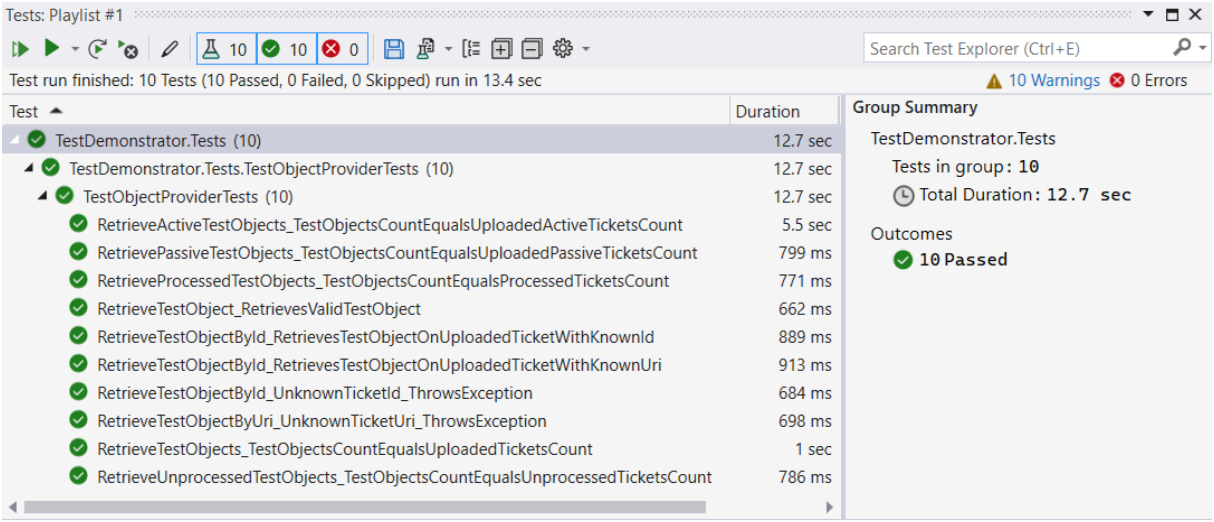


Abbildung 5.13: Testergebnisse - *Test Object Provider*

5.3.8 Test Orchestrator

Das Modul *Test Orchestrator* entspricht einer der wichtigsten Komponenten der *Core*-Ebene. Dieses abstrahiert die Funktionalitäten untergeordneter Module wie z.B. *Test Runner* und *Test Object Provider* und bietet die Applikationsschnittstelle, welche die Durchführung eines mehrstufigen, dynamisch definierbaren und domänespezifischen Testprozesses ermöglicht.

Funktionalität

Komponente	Test Orchestrator
Signatur	Test-Pipeline auf ein Testobjekt durchführen `void ExecuteTestPipeline(ref ITestObject testObject, ITicketRepository repository, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Polymorph der Schnittstelle `ITestObject` • Polymorph der Schnittstelle `ITicketRepository` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode führt die von den Testklassen implementierten Testszenarien durch den Einsatz des Moduls *Test Runner* auf das Testobjekt durch. Die Reihenfolge der durchzuführenden Teststufen definiert die Sequenz der Pipeline. Im Rahmen der Durchführung des Testprozesses wird ein Testbericht von dem ausgewählten Test-Framework generiert und als Teil des Test-Rezeptes im Ticketverzeichnis hochgeladen.
Testszenarien	TC.OR01 und TC.OR02 (siehe Tabellen 5.100 und 5.101)

Tabelle 5.92: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipeline`

Komponente	Test Orchestrator
Signatur	Test-Pipeline auf ein Testobjekt anhand einer Ticket-Kennung durchführen `void ExecuteTestPipelineById(Guid ticketId, ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Ticket-Kennung als Instanz der Klasse `Guid` • Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert das Ticket, dessen Identifikator mit dem ersten Argument der Funktion übereinstimmt. Durch die Dienste der Komponente *Test Object Provider* wird anhand der Ticket-Informationen ein Polymorph der Schnittstelle `ITestObject` instanziiert. Durch die Dienste des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf das Testobjekt statt.
Testszenarien	TC.OR03 and TC.OR04 (siehe Tabellen 5.102 and 5.103)

Tabelle 5.93: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipelineById`

Komponente	Test Orchestrator
Signatur	Test-Pipeline auf ein Testobjekt anhand einer URI-Adresse durchführen `void ExecuteTestPipelineByUri(Uri uri, ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• URI der VWS des Testobjekts als Instanz der Klasse `Uri` • Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert das Ticket, dessen `Uri`-Attribut mit dem ersten Argument der Funktion übereinstimmt. Durch die Dienste der Komponente *Test Object Provider* wird anhand der Ticket-Informationen ein Polymorph der Schnittstelle `ITestObject` instanziiert. Durch die Dienste des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf das Testobjekt statt.
Testszenarien	TC.OR05 and TC.OR06 (siehe Tabellen 5.104 and 5.105)

Tabelle 5.94: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipelineByUri`

Komponente	Test Orchestrator
Signatur	Test-Pipeline für jedes Ticket im Verzeichnis durchführen `ExecuteTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert alle Tickets aus der Ticket-Repository. Durch die Dienste der Komponente *Test Object Provider* wird für jedes Ticket ein Polymorph der Schnittstelle `ITestObject` instanziiert. Auf Basis der Funktionalitäten des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf jedem Testobjekt statt.
Testszenarien	TC.OR07 (siehe Tabellen 5.106)

Tabelle 5.95: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipelines`

Komponente	Test Orchestrator
Signatur	Test-Pipeline für jedes passive Ticket im Verzeichnis durchführen `ExecutePassiveTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert alle passiven Tickets aus der Ticket-Repository. Durch die Dienste der Komponente *Test Object Provider* wird für jedes Ticket ein Polymorph der Schnittstelle `ITestObject` instanziiert. Auf Basis der Funktionalitäten des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf allen passiven Testobjeken statt.
Testszenarien	TC.OR08 (siehe Tabellen 5.107)

Tabelle 5.96: Funktionsbeschreibung: *Test Orchestrator* - `ExecutePassiveTestPipelines`

Komponente	Test Orchestrator
Signatur	Test-Pipeline für jedes aktive Ticket im Verzeichnis durchführen `void ExecuteActiveTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert alle aktiven Tickets aus der Ticket-Repository. Durch die Dienste der Komponente *Test Object Provider* wird für jedes Ticket ein Polymorph der Schnittstelle `ITestObject` instanziiert. Auf Basis der Funktionalitäten des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf allen aktiven Testobjeken statt.
Testszenarien	TC.OR09 (siehe Tabellen 5.108)

Tabelle 5.97: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteActiveTestPipelines`

Komponente	Test Orchestrator
Signatur	Test-Pipeline für jedes bearbeitete Ticket im Verzeichnis wiederholen `void ExecuteProcessedTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert alle bearbeiteten Tickets aus der Ticket-Repository. Durch die Dienste der Komponente *Test Object Provider* wird für jedes Ticket ein Polymorph der Schnittstelle `ITestObject` instanziiert. Auf Basis der Funktionalitäten des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf alle bereits getesteten Testobjeken statt.
Testszenarien	TC.OR10 (siehe Tabellen 5.109)

Tabelle 5.98: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteProcessedTestPipelines`

Komponente	Test Orchestrator
Signatur	Test-Pipeline für jedes Ticket im Verzeichnis durchführen, das ein nicht getestetes Testobjekt repräsentiert `void ExecuteUnprocessedTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Input	• Polymorph der Schnittstelle `ITestObjectProvider` • Polymorph der Schnittstelle `ITestRunner` • Liste aus Klassen (Spezialisierungen der abstrakten Klasse `Type`), welche die Teststufen der Test-Pipeline implementieren
Beschreibung	Die Methode extrahiert alle Tickets aus der Ticket-Repository, die nicht getestete Testobjekte repräsentieren. Durch die Dienste der Komponente *Test Object Provider* wird für jedes Ticket ein Polymorph der Schnittstelle `ITestObject` instanziiert. Auf Basis der Funktionalitäten des Moduls *Test Runner* findet die Durchführung der übergebenen Teststufen auf alle nicht verifizierten Testobjeken statt.
Testszenarien	TC.OR11 (siehe Tabellen 5.110)

Tabelle 5.99: Funktionsbeschreibung: *Test Orchestrator* - `ExecuteUnprocessedTestPipelines`

Testszenarien

Identifikation	TC.OR01
Testszenario	Durchführung einer Test-Pipeline
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipeline(ref ITestObject testObject, ITicketRepository repository, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.92
Beschreibung	Anhand dieses Testfalls soll die Durchführbarkeit der Test-Pipeline und die erfolgreiche Erstellung eines Testberichtes nachgewiesen werden.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die Methode zum Ausführen der Pipeline für jedes durch das Modul *Test Object Provider* bereitgestellte Testobjekt aufgerufen werden. Als Voraussetzung dafür gilt die korrekte Definition der Test-Pipeline.
Ressourcen	Als Ressourcen gelten zwei passive und zwei aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Testklassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden durchgeführt. Die entsprechende Ticket-Entität wird durch einen Testbericht erweitert, der Informationen zu jedem Schritt der Pipeline enthält.

Tabelle 5.100: Teststrategie: *Test Orchestrator* - TC.OR01

Identifikation	TC.OR02
Testszenario	Durchführung einer Test-Pipeline, die durch ungültige Testklassen definiert ist
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipeline(ref ITestObject testObject, ITicketRepository repository, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.92
Beschreibung	Anhand dieses Testfalls soll die fehlerhafte Anwendung der Funktionalität durch die inkorrekte Spezifikation einer Test-Pipeline simuliert werden. Es wird bestätigt, dass dies zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die Methode zum Ausführen der Pipeline für jedes durch das Modul *Test Object Provider* bereitgestellte Testobjekt aufgerufen werden.
Ressourcen	Als Ressourcen gelten zwei passive und zwei aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die keine Tests implementieren und im Kontext der objektorientierten Programmierung als normale Klassen und nicht als Test-Suites anzusehen sind.
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`

Tabelle 5.101: Teststrategie: *Test Orchestrator* - TC.OR02

Identifikation	TC.OR03
Testszenario	Durchführung einer Test-Pipeline für ein bestimmtes Test-Ticket anhand seiner Identifikationsnummer
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipelineById(Guid ticketId, ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.93
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für vordefinierte Testrezepte überprüft werden. Die Spezifikation des Rezepts erfolgt durch die Identifikationsnummer des Tickets.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die Methode zum Ausführen der Pipeline für jedes hochgeladene Ticket aufgerufen werden.
Ressourcen	Als Ressourcen gelten zwei passive und zwei aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden durchgeführt. Die entsprechende Ticket-Entität wird durch einen Testbericht erweitert, der Informationen zu jedem Schritt der Pipeline enthält.

Tabelle 5.102: Teststrategie: *Test Orchestrator* - TC.OR03

Identifikation	TC.OR04
Testszenario	Durchführung einer Test-Pipeline anhand einer `TicketId`, die nicht registriert ist
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipelineById(Guid ticketId, ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.93
Beschreibung	Anhand dieses Testfalls soll sichergestellt werden, dass die Durchführung der Test-Pipeline für Testrezepte, die nicht registriert sind, zu einer Ausnahme führt.
Vorgehen	Die Methode soll mit einem Ticket-Identifikator aufgerufen werden, durch den kein Testrezept im Verzeichnis identifizierbar ist.
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`

Tabelle 5.103: Teststrategie: *Test Orchestrator* - TC.OR04

Identifikation	TC.OR05
Testszenario	Durchführung einer Test-Pipeline für ein bestimmtes Test-Ticket anhand der vom Ticket spezifizierten URI-Adresse
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipelineByUri(Uri uri, ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.94
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für vordefinierte Testrezepte überprüft werden. Die Spezifikation des Rezepts erfolgt durch die URI-Adresse des Tickets.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die Methode zum Ausführen der Pipeline für jedes hochgeladene und aktive Ticket aufgerufen werden. Zur Spezifikation des Testrezeptes dienen die von den Tickets spezifizierten URI-Adressen.
Ressourcen	Als Ressourcen gelten zwei aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden durchgeführt. Die aktiven Ticket-Entitäten werden durch einen Testbericht erweitert, der Informationen zu jedem Schritt der Pipeline enthält.

Tabelle 5.104: Teststrategie: *Test Orchestrator* - TC.OR05

Identifikation	TC.OR06
Testszenario	Durchführung einer Test-Pipeline anhand einer URI-Adresse, durch welche kein Ticket assoziiert werden kann
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipelineByUri(Uri uri, ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.94
Beschreibung	Anhand dieses Testfalls soll sichergestellt werden, dass die Durchführung der Test-Pipeline durch die Übergabe einer URI-Adresse, mit welcher kein Ticket assoziierbar ist, zu einer Ausnahme führt.
Vorgehen	Die Methode soll mit einer URI-Adresse aufgerufen werden, die keinen VWS-Server repräsentiert.
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`

Tabelle 5.105: Teststrategie: *Test Orchestrator* - TC.OR06

Identifikation	TC.OR07
Testszenario	Durchführung einer Test-Pipeline für alle Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.95
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für alle Testrezepte überprüft werden.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die Methode zum Ausführen der Pipeline für alle Tickets aufgerufen werden.
Ressourcen	Als Ressourcen gelten passive und aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden durchgeführt. Alle Ticket-Entitäten werden durch einen Testbericht erweitert, der Informationen zu jedem Schritt der Pipeline enthält.

Tabelle 5.106: Teststrategie: *Test Orchestrator* - TC.OR07

Identifikation	TC.OR08
Testszenario	Durchführung einer Test-Pipeline für alle passiven Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecutePassiveTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.96
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für alle passiven Testrezepte überprüft werden.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die von diesem Testfall adressierte Methode zum Ausführen der Pipeline für alle passiven Tickets aufgerufen werden.
Ressourcen	Als Ressourcen gelten passive und aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden nur für die Testobjekte durchgeführt, die durch ein passives Ticket repräsentiert werden. Alle passiven Ticket-Entitäten werden durch einen Testbericht erweitert, der Informationen zu jedem Schritt der Pipeline enthält.

Tabelle 5.107: Teststrategie: *Test Orchestrator* - TC.OR08

Identifikation	TC.OR09
Testszenario	Durchführung einer Test-Pipeline für alle aktiven Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteActiveTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.97
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für alle aktiven Testrezepte überprüft werden.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die von diesem Testfall adressierte Methode zum Ausführen der Pipeline für alle aktiven Tickets aufgerufen werden.
Ressourcen	Als Ressourcen gelten passive und aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden nur für die Testobjekte durchgeführt, die durch ein aktives Ticket repräsentiert werden. Alle aktiven Ticket-Entitäten werden durch einen Testbericht erweitert, der Informationen zu jedem Schritt der Pipeline enthält.

Tabelle 5.108: Teststrategie: *Test Orchestrator* - TC.OR09

Identifikation	TC.OR10
Testszenario	Wiederholte durchführung einer Test-Pipeline für alle Tickets im Verzeichnis
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteProcessedTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.98
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für alle Testrezepte überprüft werden, die bereits bearbeitet worden sind.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die von diesem Testfall adressierte Methode zum Ausführen der Pipeline für alle bereits bearbeiteten Tickets aufgerufen werden.
Ressourcen	Als Ressourcen gelten passive und aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden nur für die Testobjekte durchgeführt, die durch ein Ticket repräsentiert werden, dessen `TestReport`-Attribut bereits initialisiert ist.

Tabelle 5.109: Teststrategie: *Test Orchestrator* - TC.OR10

Identifikation	TC.OR11
Testszenario	Durchführung einer Test-Pipeline für alle Tickets im Verzeichnis, die kein Testbericht beinhalten
Testgruppe	`Demonstrator.Tests.TestOrchestratorTests`
Funktionalität	`void ExecuteUnprocessedTestPipelines(ITestObjectProvider testObjectProvider, ITestRunner testRunner, ICollection<Type> testClasses)`
Funktionsbeschreibung	siehe Tabelle 5.99
Beschreibung	Anhand dieses Testfalls soll die Durchführung der Test-Pipeline für alle Testrezepte überprüft werden, die nicht bearbeitet worden sind.
Vorgehen	Nach dem programmiertechnischen Erstellen und Hochladen von mehreren aktiven und passiven Tickets in einer Test-Datenbank durch die APIs der Komponenten *Ticket Builder* und *Ticket (Database) Repository* soll die von diesem Testfall adressierte Methode zum Ausführen der Pipeline aufgerufen werden.
Ressourcen	Als Ressourcen gelten passive und aktive Verwaltungsschalen. Das Starten der aktiven Vertreter erfolgt zusammen mit dem Hochladen der Tickets im Rahmen der *Arrange*-Phase. Als Pipeline-Stufen gelten zwei Klassen, die als *Test-Doubles* gelten.
Erwartung	Die Teststufen werden nur für die Testobjekte durchgeführt, die durch ein Ticket repräsentiert werden, dessen `TestReport`-Attribut nicht initialisiert ist.

Tabelle 5.110: Teststrategie: *Test Orchestrator* - TC.OR11

Implementierung

Wie von der tabellarischen Beschreibung der Funktionalitäten und ihrer Testszenarien angegeben, basiert das Modul *Test Orchestrator* auf die Funktionalitäten der Komponenten *Ticket Repository*, *Test Object*, *Test Object Provider*, und *Test Runner*. Zum Zweck der Flexibilität der API werden diese Module nicht als Argumente des Konstruktors sondern bei jedem Aufruf der entsprechenden Instanzmethode

übergeben. Durch die Einhaltung der implementierten Schnittstellen lassen sich unterschiedlich definierte Test-Pipelines für unterschiedliche *Ticket Repository*-Implementierungen, durch den Einsatz unterschiedlicher Test-Frameworks durchführen, was über eine hohe Wandlungsfähigkeit der Testlösung in Abhängigkeit von dem spezifischen Anwendungsfall spricht. Als Grundlage dafür dient die lose Kopplung zwischen den Komponenten durch die Anwendung der Methode *Dependency Injection*. Abb. 5.14 stellt die Rolle des Moduls *Test Orchestrator* im Kontext der Softwarearchitektur dar.

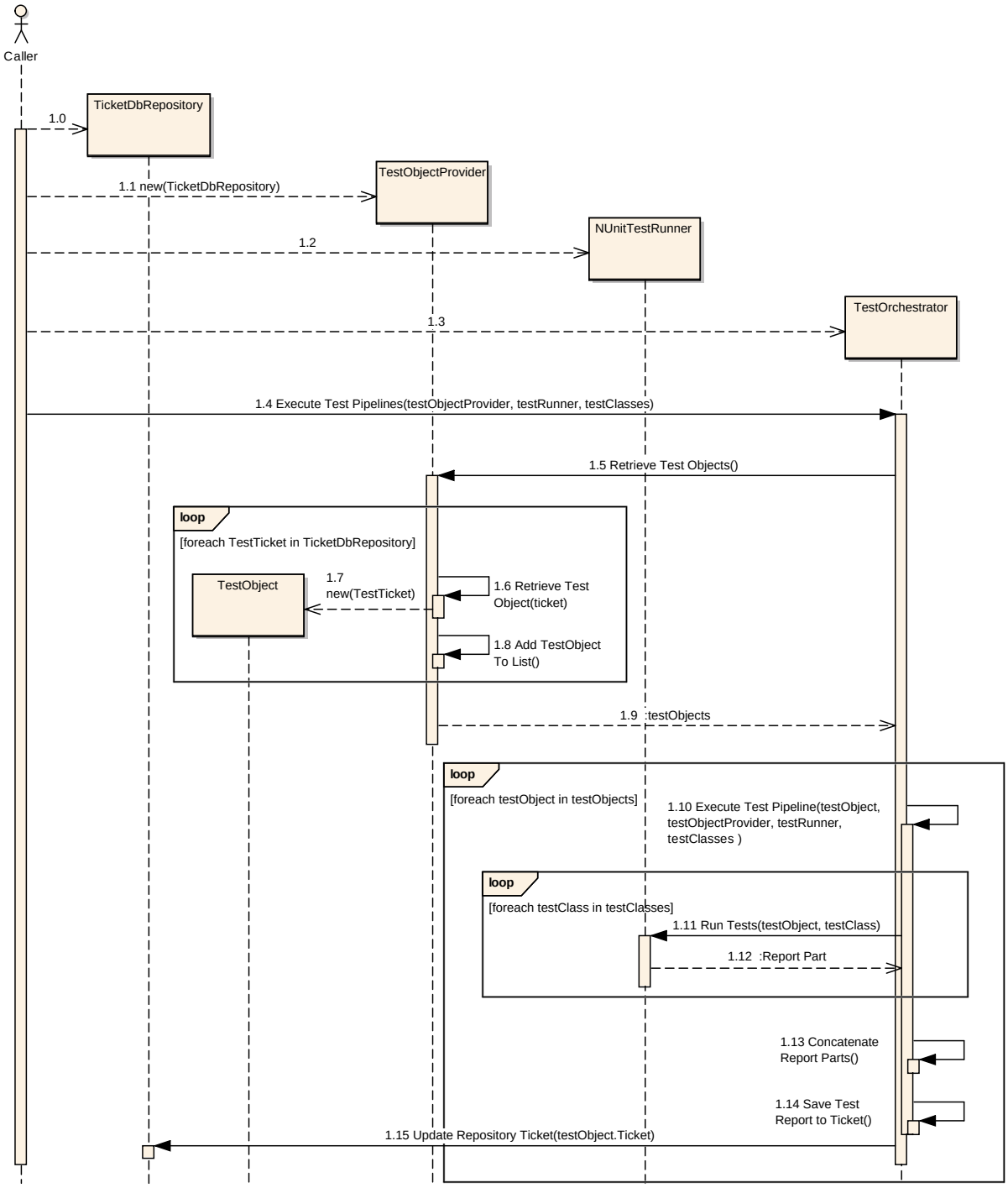


Abbildung 5.14: Prinzipielle Funktionsweise der Komponente *Test Orchestrator*

Die Komponente nutzt die Dienste der Komponente *Test Object Provider*, um ein konkretes Testrezept, unterschiedliche Klassen von Testrezepten oder alle Tickets, die durch die Dienste vom *Ticket Repository* hochgeladen oder aktualisiert wurden, abzurufen. Jedes Ticket wird zu einer Instanz der Klasse `ITestObject` transformiert. Wie spezifiziert, erfolgt die Spezifikation der Stufen Test-Pipeline durch die Angabe einzelner Testklassen. Das Triggern des Testprozesses findet durch den Dienst des Moduls *Test Runner* statt.

Auswertung

Die folgende Abbildung dient als Nachweis für die bestandenen Testszenarien.

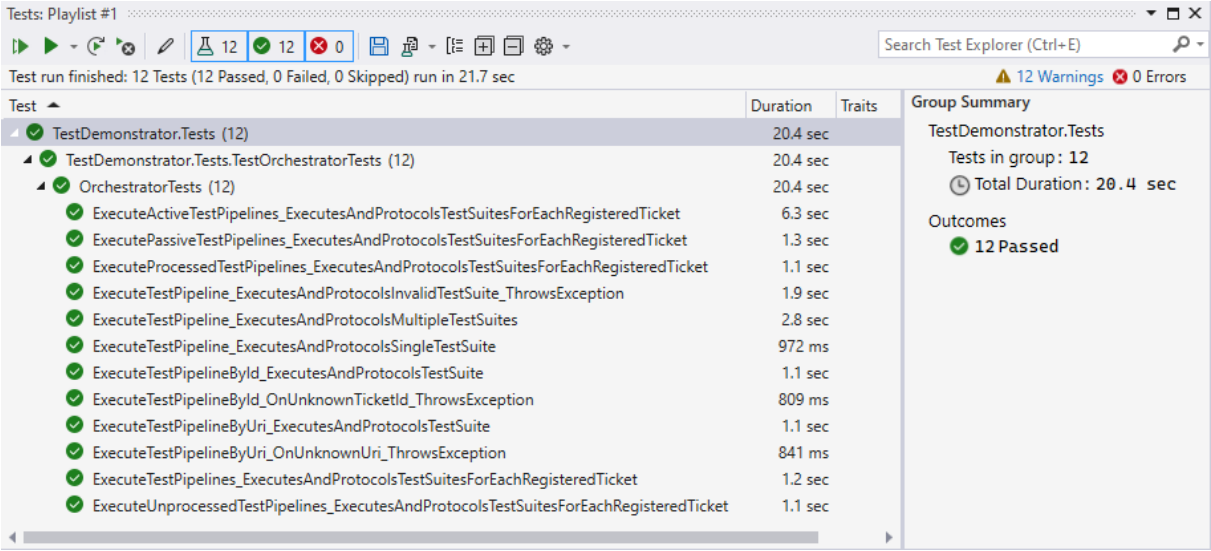


Abbildung 5.15: Testergebnisse - *Test Orchestrator*

5.4 Dienste

Im Rahmen dieses Abschnitts wird die Implementierung und Testung der Komponenten der *Service*-Ebene beschrieben. Ihre Korrektheit wurde manuell und über die GUI sichergestellt.

5.4.1 Ticket Repository Service

Wie im Rahmen der Konzepterstellung erläutert wurde, besteht die Aufgabe der Komponente *Ticket Repository Service* in der Adaptierung der Informationen, die zum Zweck der Interaktion mit dem entsprechenden Repository-Submodell vom Anwender bereitgestellt werden, an den Funktionalitäten der Komponenten *Descriptor Factory*, *Ticket Builder* und *Ticket Repository*. Das Modul kapselt die Dienste dieser Bestandteile der Softwarearchitektur und gilt als Grundlage für die Implementierung des Submodells, welches das Ticketverzeichnis repräsentiert. Eine weitere wichtige Aufgabe des Dienstes entspricht der Fehlerbehandlung, was im Rahmen der testgetriebenen Entwicklung als Voraussetzung für eine hohe Anzahl von Testszenarien gilt.

Funktionalität

Komponente	Ticket Repository Service
Signatur	Hochladen von Testrezepten für aktive Verwaltungsschalen im Ticket-Repository `string UploadActiveTicket(string uriString, string structureContent, string statesContent)`
Input	• URI-Adresse des Testobjekts als `string` • Strukturdeskriptor in XML-Format als `string` • Zustandsautomatendeskriptor in XML-Format als `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Instanziieren und Hochladen aktiver Testrezepte im Ticket-Repository verantwortlich
Testszenarien	TC.TRS01, TC.TRS02, TC.TRS03, TC.TRS04, TC.TRS05 (siehe Tabellen 5.119, 5.120, 5.121, 5.122 und 5.123)

Tabelle 5.111: Funktionsbeschreibung: *Ticket Repository Service* - `UploadActiveTicket`

Komponente	Ticket Repository Service
Signatur	Hochladen von Testrezepten für passive Verwaltungsschalen im Ticket-Repository `string UploadPassiveTicket(string passiveShellContent, string structureContent)`
Input	• Passive VWS des Testobjekts in XML-Format als `string` • Strukturdeskriptor in XML-Format als `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Instanziieren und Hochladen passiver Testrezepte im Ticket-Repository verantwortlich
Testszenarien	TC.TRS06, TC.TRS07, TC.TRS08 (siehe Tabellen 5.124, 5.125, 5.126)

Tabelle 5.112: Funktionsbeschreibung: *Ticket Repository Service* - `UploadPassiveTicket`

Komponente	Ticket Repository Service
Signatur	Aktualisierung eines aktiven Testrezeptes anhand seiner Identifikationsnummer `string UpdateActiveTicket(string guid, string uriString, string structureContent, string statesContent)`
Input	• Id des zu aktualisierenden Tickets als `string` • URI-Adresse des Testobjekts als `string` • Strukturdeskriptor in XML-Format als `string` • Zustandsautomatendeskriptor in XML-Format als `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Aktualisieren aktiver Testrezepte verantwortlich
Testszenarien	TC.TRS09, TC.TRS10, TC.TRS11, TC.TRS12, TC.TRS13, TC.TRS14 (siehe Tabellen 5.127, 5.128, 5.129, 5.130, 5.131, 5.132)

Tabelle 5.113: Funktionsbeschreibung: *Ticket Repository Service* - `UpdateActiveTicket`

Komponente	Ticket Repository Service
Signatur	Aktualisierung eines passiven Testrezeptes anhand seiner Identifikationsnummer `string UpdatePassiveTicket(string guid, string passiveShellContent, string structureContent)`
Input	• Id des zu aktualisierenden Tickets als `string` • Passive VWS des Testobjekts in XML-Format als `string` • Strukturdeskriptor in XML-Format als `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Aktualisieren passiver Testrezepte verantwortlich
Testszenarien	TC.TRS15, TC.TRS16, TC.TRS17 (siehe Tabellen 5.133, 5.134 und 5.135)

Tabelle 5.114: Funktionsbeschreibung: *Ticket Repository Service* - `UpdatePassiveTicket`

Komponente	Ticket Repository Service
Signatur	Löschen eines Testrezeptes aus dem Ticket-Repository anhand seiner Identifikationsnummer `DeleteTicket(string guid)`
Input	• Id des zu aktualisierenden Tickets `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Löschen von Testrezepten verantwortlich
Testszenarien	TC.TRS18, TC.TRS19, TC.TRS20 (siehe Tabellen 5.136, 5.137 und 5.138)

Tabelle 5.115: Funktionsbeschreibung: *Ticket Repository Service* - `DeleteTicket`

Komponente	Ticket Repository Service
Signatur	Abrufen eines Testrezeptes aus dem Ticket-Repository anhand seiner Identifikationsnummer `string RetrieveTicketById(string guid, ref ITestTicket? ticket)`
Input	• Id des zu aktualisierenden Tickets `string` • Polymorph der Schnittstelle `ITestTicket`, dessen Wert `null` entspricht
Output	Polymorph der Schnittstelle `ITestTicket`, dessen Wert dem Inhalt des Testrezeptes entspricht
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Auslesen von Testrezepten aus dem Verzeichnis verantwortlich
Testszenarien	TC.TRS21, TC.TRS22, TC.TRS23 (siehe Tabellen 5.139, 5.140 und 5.141)

Tabelle 5.116: Funktionsbeschreibung: *Ticket Repository Service* - `RetrieveTicketById`

Komponente	Ticket Repository Service
Signatur	Abrufen eines aktiven Testrezeptes aus dem Ticket-Repository anhand einer URI-Adresse `string RetrieveTicketByUri(string uri, ref ITestTicket? ticket)`
Input	• Die vom Ticket spezifizierte URI-Adresse als `string` • Polymorph der Schnittstelle `ITestTicket`, dessen Wert `null` entspricht
Output	Polymorph der Schnittstelle `ITestTicket`, dessen Wert dem Inhalt des Testrezeptes entspricht
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Die Methode ist für das Auslesen von aktiven Testrezepten aus dem Verzeichnis anhand einer URI-Adresse verantwortlich
Testszenarien	TC.TRS24, TC.TRS25, TC.TRS26 (siehe Tabellen 5.142, 5.143 und 5.144)

Tabelle 5.117: Funktionsbeschreibung: *Ticket Repository Service* - `RetrieveTicketByUri`

Komponente	Ticket Repository Service
Signatur	Abrufen aller Ticket-Identifikatoren `string RetrieveTicketIds()`
Output	Array aus Ticket-Identifikatoren
Beschreibung	Die Methode ist für das Auslesen allet Ticket-Identifikatoren verantwortlich

Tabelle 5.118: Funktionsbeschreibung: *Ticket Repository Service* - `RetrieveTicketIds`

Testszenarien

Identifikation	TC.TRS01
Testszenario	Hochladen eines aktiven Testrezeptes durch Übergabe von Argumenten, die entweder `null` oder leeren Zeichenketten entsprechen
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadActiveTicket(string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.111
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit Argumenten, die nicht verarbeitet werden können, zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Die Methode wird mehrmals aufgerufen. Bei jedem Aufruf entspricht ein unterschiedliches Argument dem Wert `null` oder `string.Empty`.
Erwartung	Die adressierte Funktionalität wirft eine Ausnahme vom Typ `ArgumentException` und informiert den Anwender, welches Argument von den Anforderungen des Systems abweicht.

Tabelle 5.119: Teststrategie: *Ticket Repository Service* - TC.TRS01

Identifikation	TC.TRS02
Testszenario	Hochladen eines aktiven Testrezeptes durch Übergabe einer ungültigen URI-Adresse
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadActiveTicket(string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.111
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einer Zeichenkette, die mit einer URI-Adresse nicht assoziiert werden kann, zu einer Ausnahme vom Typ `UriFormatException` führt.
Vorgehen	Die Methode wird mehrmals mit Zeichenketten aufgerufen, die keine URI-Adressen repräsentieren.
Erwartung	Die adressierte Funktionalität wirft eine Ausnahme vom Typ `UriFormatException`.

Tabelle 5.120: Teststrategie: *Ticket Repository Service* - TC.TRS02

Identifikation	TC.TRS03
Testszenario	Hochladen eines aktiven Testrezeptes durch Übergabe eines ungültigen Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadActiveTicket(string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.111
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden Strukturdeskriptor zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einem Strukturdeskriptor aufgerufen, der vom AAS-Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme.

Tabelle 5.121: Teststrategie: *Ticket Repository Service* - TC.TRS03

Identifikation	TC.TRS04
Testszenario	Hochladen eines aktiven Testrezeptes durch Übergabe eines ungültigen Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadActiveTicket(string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.111
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden Zustandsautomatendeskriptor zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einem Zustandsautomatendeskriptor aufgerufen, der vom in Listing A.3 definierten Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme vom Typ `InvalidOperationException`.

Tabelle 5.122: Teststrategie: *Ticket Repository Service* - TC.TRS04

Identifikation	TC.TRS05
Testszenario	Erfolgreiches Hochladen eines aktiven Testrezeptes
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadActiveTicket(string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.111
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einer gültigen URI-Adresse und mit Ticket-Deskriptoren, welche den XML-Spezifikationen entsprechen, zum erfolgreichen Hochladen eines aktiven Tickets führt.
Vorgehen	Die Methode wird mit korrekten Eingangsargumenten aufgerufen.
Erwartung	Das Aufrufen der Funktion führt zum Hochladen eines aktiven Testrezeptes und zur entsprechenden Benachrichtigung.

Tabelle 5.123: Teststrategie: *Ticket Repository Service* - TC.TRS05

Identifikation	TC.TRS06
Testszenario	Hochladen eines passiven Testrezeptes durch Übergabe einer ungültigen passiven VWS
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadPassiveTicket(string passiveShellContent, string structureContent)`
Funktionsbeschreibung	siehe Tabelle 5.112
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden VWS-Inhalt zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einer XML-Zeichenkette aufgerufen, die vom AAS-Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme.

Tabelle 5.124: Teststrategie: *Ticket Repository Service* - TC.TRS06

Identifikation	TC.TRS07
Testszenario	Hochladen eines passiven Testrezeptes durch Übergabe eines ungültigen Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadPassiveTicket(string passiveShellContent, string structureContent)`
Funktionsbeschreibung	siehe Tabelle 5.112
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden Strukturdeskriptor zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einer XML-Zeichenkette aufgerufen, die vom AAS-Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme.

Tabelle 5.125: Teststrategie: *Ticket Repository Service* - TC.TRS07

Identifikation	TC.TRS08
Testszenario	Erfolgreiches Hochladen eines passiven Testrezeptes
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UploadPassiveTicket(string passiveShellContent, string structureContent)`
Funktionsbeschreibung	siehe Tabelle 5.112
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem gültigen VWS-Inhalt und einem Strukturdeskriptor, der dem XML-Schema entspricht, zum erfolgreichen Hochladen eines aktiven Tickets führt.
Vorgehen	Die Methode wird mit korrekten Eingangsargumenten aufgerufen.
Erwartung	Das Aufrufen der Funktion führt zum Hochladen eines passiven Testrezeptes und zur entsprechenden Benachrichtigung.

Tabelle 5.126: Teststrategie: *Ticket Repository Service* - TC.TRS08

Identifikation	TC.TRS09
Testszenario	Aktualisierung eines aktiven Testrezeptes durch Übergabe von Argumenten, die entweder `null` oder leeren Zeichenketten entsprechen
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdateActiveTicket(string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.113
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit Argumenten, die nicht verarbeitet werden können, zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Die Methode wird mehrmals aufgerufen. Bei jedem Aufruf entspricht ein unterschiedliches Argument dem Wert `null` oder `string.Empty`.
Erwartung	Die adressierte Funktionalität wirft eine Ausnahme vom Typ `ArgumentException` und informiert den Anwender, welches Argument von den Anforderungen des Systems abweicht.

Tabelle 5.127: Teststrategie: *Ticket Repository Service* - TC.TRS09

Identifikation	TC.TRS10
Testszenario	Aktualisieren eines aktiven Testrezeptes durch Übergabe eines ungültigen Ticket-Identifikators
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdateActiveTicket(string guid, string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.113
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem ungültigen Ticket-Identifikator, zu einer Ausnahme vom Typ `FormatException` führt.
Vorgehen	Die Methode wird mehrmals mit Zeichenketten aufgerufen, die keine Identifikationsdaten repräsentieren.
Erwartung	Die adressierte Funktionalität wirft eine Ausnahme vom Typ `FormatException`.

Tabelle 5.128: Teststrategie: *Ticket Repository Service* - TC.TRS10

Identifikation	TC.TRS11
Testszenario	Aktualisieren eines aktiven Testrezeptes durch Übergabe einer ungültigen URI-Adresse
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdateActiveTicket(string guid, string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.113
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einer Zeichenkette, die mit einer URI-Adresse nicht assoziiert werden kann, zu einer Ausnahme vom Typ `UriFormatException` führt.
Vorgehen	Die Methode wird mehrmals mit Zeichenketten aufgerufen, die keine URI-Adressen repräsentieren.
Erwartung	Die adressierte Funktionalität wirft eine Ausnahme vom Typ `UriFormatException`.

Tabelle 5.129: Teststrategie: *Ticket Repository Service* - TC.TRS11

Identifikation	TC.TRS12
Testszenario	Aktualisieren eines aktiven Testrezeptes durch Übergabe eines ungültigen Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdateActiveTicket(string guid, string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.113
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden Strukturdeskriptor zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einem Strukturdeskriptor aufgerufen, der vom AAS-Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme.

Tabelle 5.130: Teststrategie: *Ticket Repository Service* - TC.TRS12

Identifikation	TC.TRS13
Testszenario	Aktualisieren eines aktiven Testrezeptes durch Übergabe eines ungültigen Zustandsautomatendeskriptors
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdateActiveTicket(string guid, string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.113
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden Zustandsautomatendeskriptor zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einem Zustandsautomatendeskriptor aufgerufen, der vom in Listing A.3 definierten Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme vom Typ `InvalidOperationException`.

Tabelle 5.131: Teststrategie: *Ticket Repository Service* - TC.TRS13

Identifikation	TC.TRS14
Testszenario	Erfolgreiches Aktualisieren eines aktiven Testrezeptes
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdateActiveTicket(string guid, string uriString, string structureContent, string statesContent)`
Funktionsbeschreibung	siehe Tabelle 5.113
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einer gültigen URI-Adresse und mit Ticket-Deskriptoren, welche den XML-Spezifikationen entsprechen, zum erfolgreichen Hochladen eines aktiven Tickets führt.
Vorgehen	Die Methode wird mit korrekten Eingangsargumenten nach dem Hochladen eines aktiven Testrezeptes aufgerufen. Als erstes Argument gilt die Identifikationsnummer des bereits hochgeladenen Test-Tickets.
Erwartung	Das Aufrufen der Funktion führt zum Aktualisieren des spezifizierten Testrezeptes und zur entsprechenden Benachrichtigung.

Tabelle 5.132: Teststrategie: *Ticket Repository Service* - TC.TRS14

Identifikation	TC.TRS15
Testszenario	Aktualisieren eines passiven Testrezeptes durch Übergabe einer ungültigen passiven VWS
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdatePassiveTicket(string guid, string passiveShellContent, string structureContent)`
Funktionsbeschreibung	siehe Tabelle 5.114
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden VWS-Inhalt zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einer XML-Zeichenkette aufgerufen, die vom AAS-Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme.

Tabelle 5.133: Teststrategie: *Ticket Repository Service* - TC.TRS15

Identifikation	TC.TRS16
Testszenario	Aktualisieren eines passiven Testrezeptes durch Übergabe eines ungültigen Strukturdeskriptors
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdatePassiveTicket(string guid, string passiveShellContent, string structureContent)`
Funktionsbeschreibung	siehe Tabelle 5.114
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem vom XML-Schema abweichenden Strukturdeskriptor zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einer XML-Zeichenkette aufgerufen, die vom AAS-Schema abweicht.
Erwartung	Das Aufrufen der Funktion führt zu einer Ausnahme.

Tabelle 5.134: Teststrategie: *Ticket Repository Service* - TC.TRS16

Identifikation	TC.TRS17
Testszenario	Erfolgreiches Aktualisieren eines passiven Testrezeptes
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string UpdatePassiveTicket(string guid, string passiveShellContent, string structureContent)`
Funktionsbeschreibung	siehe Tabelle 5.114
Beschreibung	Durch dieses Testszenario soll sichergestellt werden, dass das Aufrufen der Methode mit einem gültigen VWS-Inhalt und einem Strukturdeskriptor, der dem XML-Schema entspricht, zum erfolgreichen Aktualisieren eines aktiven Tickets führt.
Vorgehen	Die Methode wird mit korrekten Eingangsargumenten aufgerufen. Als Voraussetzung für den Testfall gilt das Hochladen eines oder mehrerer Test-Tickets.
Erwartung	Das Aufrufen der Funktion führt zum Aktualisieren eines passiven Testrezeptes und zur entsprechenden Benachrichtigung.

Tabelle 5.135: Teststrategie: *Ticket Repository Service* - TC.TRS17

Identifikation	TC.TRS18
Testszenario	Löschen eines Tickets durch Übergabe eines Identifikators, dessen Wert `null` oder leerer Zeichenkette entspricht
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string DeleteTicket(string guid)`
Funktionsbeschreibung	siehe Tabelle 5.115
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einem vom Format abweichenden Identifikator zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Die Methode wird mit einem Argument aufgerufen, dessen Wert `null` entspricht oder mit einer leeren Zeichenkette korrespondiert.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme vom Typ `ArgumentException`

Tabelle 5.136: Teststrategie: *Ticket Repository Service* - TC.TRS18

Identifikation	TC.TRS19
Testszenario	Löschen eines Tickets durch Übergabe eines Identifikators, mit welchem keine Ticket-Entität assoziierbar ist
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string DeleteTicket(string guid)`
Funktionsbeschreibung	siehe Tabelle 5.115
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einem Identifikator, durch welchen kein Ticket charakterisiert ist, zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einem Identifikator aufgerufen, der mit keinem Ticket im Repository assoziiert werden kann.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme.

Tabelle 5.137: Teststrategie: *Ticket Repository Service* - TC.TRS19

Identifikation	TC.TRS20
Testszenario	Erfolgreiches Löschen eines Test-Tickets aus dem Repository
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string DeleteTicket(string guid)`
Funktionsbeschreibung	siehe Tabelle 5.115
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einem Identifikator, der der Kennung einer im Verzeichnis registrierten Ticket-Entität entspricht, zum Löschen des Test-Rezeptes aus dem Verzeichnis führt.
Vorgehen	An erster Stelle erfolgt das Hochladen eines Tickets im Verzeichnis. Im nächsten Schritt wird die Methode mit dem Identifikator des registrierten Testrezeptes aufgerufen.
Erwartung	Die Anzahl der Tickets im Repository entspricht dem Wert `0`.

Tabelle 5.138: Teststrategie: *Ticket Repository Service* - TC.TRS20

Identifikation	TC.TRS21
Testszenario	Abrufen eines Tickets durch Übergabe eines Identifikators, dessen Wert `null` oder leerer Zeichenkette entspricht
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string RetrieveTicketById(string guid)`
Funktionsbeschreibung	siehe Tabelle 5.116
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einem vom Format abweichenden Identifikator zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Die Methode wird mit einem Argument aufgerufen, dessen Wert `null` entspricht oder mit einer leeren Zeichenkette korrespondiert.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme vom Typ `ArgumentException`

Tabelle 5.139: Teststrategie: *Ticket Repository Service* - TC.TRS21

Identifikation	TC.TRS22
Testszenario	Abrufen eines Tickets durch Übergabe eines Identifikators, mit welchem keine Ticket-Entität assoziierbar ist
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string RetrieveTicketById(string guid)`
Funktionsbeschreibung	siehe Tabelle 5.116
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einem Identifikator, durch welchen kein Ticket charakterisiert ist, zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einem Identifikator aufgerufen, der mit keinem Ticket im Repository assoziiert werden kann.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme.

Tabelle 5.140: Teststrategie: *Ticket Repository Service* - TC.TRS22

Identifikation	TC.TRS23
Testszenario	Erfolgreiches Abrufen eines Test-Tickets aus dem Repository
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string RetrieveTicketById(string guid)`
Funktionsbeschreibung	siehe Tabelle 5.116
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einem Identifikator, der der Kennung einer im Verzeichnis registrierten Ticket-Entität entspricht, zum Löschen des Test-Rezeptes aus dem Verzeichnis führt.
Vorgehen	An erster Stelle erfolgt das Hochladen eines Tickets im Verzeichnis. Im nächsten Schritt wird die Methode mit dem Identifikator des registrierten Testrezeptes aufgerufen.
Erwartung	Die Methode gibt das hochgeladene Ticket als Polymorph der Schnittstelle `ITestTicket` zurück.

Tabelle 5.141: Teststrategie: *Ticket Repository Service* - TC.TRS23

Identifikation	TC.TRS24
Testszenario	Abrufen eines Tickets durch Übergabe einer URI-Adresse, dessen Wert `null` oder leerer Zeichenkette entspricht
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string RetrieveTicketByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.117
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einer vom Format abweichenden URI-Adresse zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Die Methode wird mit einem Argument aufgerufen, dessen Wert `null` entspricht oder mit einer leeren Zeichenkette korrespondiert.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme vom Typ `ArgumentException`

Tabelle 5.142: Teststrategie: *Ticket Repository Service* - TC.TRS24

Identifikation	TC.TRS25
Testszenario	Abrufen eines Tickets durch Übergabe einer URI-Adresse, mit welchem keine Ticket-Entität assoziierbar ist
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string RetrieveTicketByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.117
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einer URI-Adresse, durch welchen kein Ticket charakterisiert ist, zu einer Ausnahme führt.
Vorgehen	Die Methode wird mit einer URI-Adresse aufgerufen, der mit keinem Ticket im Repository assoziiert werden kann.
Erwartung	Das Aufrufen der Methode führt zu einer Ausnahme.

Tabelle 5.143: Teststrategie: *Ticket Repository Service* - TC.TRS25

Identifikation	TC.TRS26
Testszenario	Erfolgreiches Abrufen eines Test-Tickets aus dem Repository
Testgruppe	`Demonstrator.Tests.TicketRepositoryServiceTests`
Funktionalität	`string RetrieveTicketByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.117
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der adressierten Methode mit einer URI-Adresse, die der Kennung einer im Verzeichnis registrierten Ticket-Entität entspricht, zum Abrufen des Test-Rezeptes aus dem Verzeichnis führt.
Vorgehen	An erster Stelle erfolgt das Hochladen eines Tickets im Verzeichnis. Im nächsten Schritt wird die Methode mit der URI-Adresse des registrierten Testrezeptes aufgerufen.
Erwartung	Die Methode gibt das hochgeladene Ticket als Polymorph der Schnittstelle `ITestTicket` zurück.

Tabelle 5.144: Teststrategie: *Ticket Repository Service* - TC.TRS26

Implementierung

Zum Zweck der Erfüllung der von den Testszenarien definierten Anforderungen durch die Funktionalitäten des *Ticket Repository*-Dienstes, werden die konkreten Instanzen der Komponenten *Descriptor Factory*, *Ticket Builder* und *Ticket Repository* als Argumente des Konstruktors der Klasse `TicketRepositoryService` übergeben. Dadurch findet die Kapselung aller Funktionalitäten statt, welche die Interaktion des Benutzers mit dem *Ticket Repository Submodel* ermöglichen. Durch das Modul *Descriptor Factory* erfolgt, wie spezifiziert, die Deserialisierung der vom Anwender des Submodells bereitgestellten Ticket-Deskriptoren. Auf Basis der Komponente *Ticket Builder* findet die Zuordnung der Deskriptoren zu einem aktiven bzw. passiven Polymorph der Schnittstelle `ITestTicket` statt. Das Hochladen, Aktualisieren, Löschen und Abrufen von Testrezepten wird durch die Funktionalitäten der Schnittstelle `ITicketRepository` realisiert. Werden vom Anwender inkorrekte Eingangsdaten bereitgestellt, so reagiert jede Methode mit der entsprechenden Fehlermeldung.

Auswertung

Aufgrund der hohen Anzahl von möglichen Fehlersituationen charakterisiert sich jede Methode des Dienstes durch mehrere Testfälle, wodurch ein höherer Anweisungs- bzw. Entscheidungsüberdeckungsgrad garantiert wird. Die folgende Abbildung illustriert die im Rahmen der testgetriebenen Entwicklung erfüllten Testszenarien.

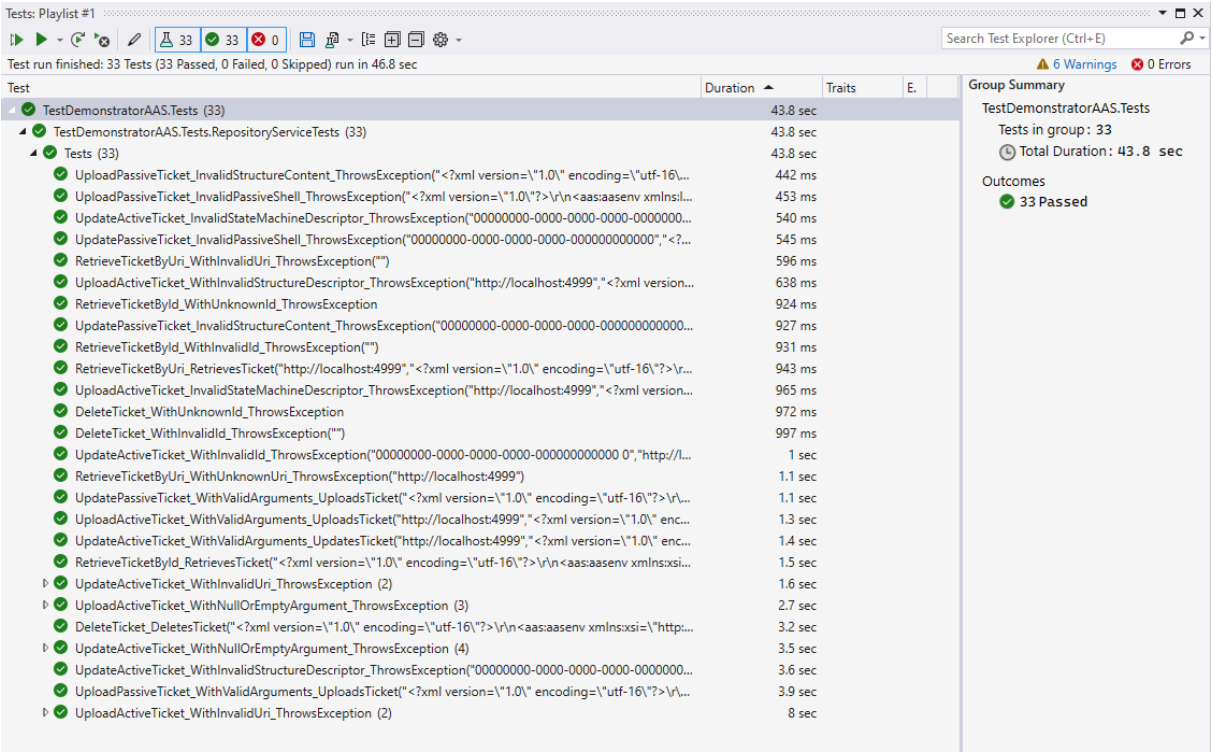


Abbildung 5.16: Testergebnisse - *Ticket Repository Service*

5.4.2 Test Orchestration Service

Der *Test Orchestration Service* stellt Funktionalitäten bereit, welche von den Operationen des für die Durchführung der Test-Pipeline verantwortlichen Submodells aufgerufen werden. Die Aufgabe der Komponente besteht also in der Adaptierung der vom Anwender des Systems bereitgestellten Informationen in einer für die Module der Kernebene geeigneten Form. Aus diesem Grund gelten die Methoden dieses Dienstes als *Wrapper*-Methoden, welche die Dienste des Moduls *Test Orchestrator* kapseln und auf inkorrekte Eingangsdaten durch die entsprechenden Ausnahmen reagieren.

Funktionalität

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für ein Testobjekt anhand seines Ticket-Identifikators `string ExecuteTestPipelineById(string ticketId)`
Input	Identifikationsnummer des Testrezeptes als `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst kapselt die durch Tabelle 5.93 beschriebene Methode und ermöglicht das Triggern einer Test-Pipeline auf Basis der Identifikationsnummer des dem Testobjekt zugeordneten Tickets über das Submodell der VWS. Des Weiteren prüft die Funktion die Gültigkeit des Eingangsarguments.
Testszenarien	TC.TOS01, TC.TOS02 und TC.TOS03 (siehe Tabellen 5.152, 5.153 und 5.154)

Tabelle 5.145: Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteTestPipelineById`

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für ein Testobjekt anhand seiner URI-Adresse `string ExecuteTestPipelineByUri(string uri)`
Input	URI-Adresse des Testobjekts als `string`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst kapselt die durch Tabelle 5.94 beschriebene Methode und ermöglicht das Triggern einer Test-Pipeline durch die Angabe der URI-Adresse des Testobjekts und über das Submodell der VWS. Des Weiteren prüft die Funktion die Gültigkeit des Eingangsarguments.
Testszenarien	TC.TOS04, TC.TOS05 und TC.TOS06 (siehe Tabellen 5.155, 5.156 und 5.157)

Tabelle 5.146: Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteTestPipelineByUri`

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für alle Testrezepte, die ein aktives Testobjekt repräsentieren `string ExecuteActiveTestPipelines()`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst kapselt die durch Tabelle 5.97 beschriebene Methode und ermöglicht das Triggern einer Test-Pipeline für alle aktiven Testrezepten über das Submodell der VWS.
Testszenarien	Da die Methode keine Argumente akzeptiert, keine Logik implementiert, als reine Schnittstelle zwischen der VWS-Ebene und dem *Core*-Layer dient und nur die angegebene und bereits verifizierte Methode des Moduls *Test Orchestrator* aufruft, gilt seine Verifikation als abgeschlossen.

Tabelle 5.147: Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteActiveTestPipelines`

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für alle Testrezepte, die ein passives Testobjekt repräsentieren `string ExecutePassiveTestPipelines()`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst kapselt die durch Tabelle 5.96 beschriebene Methode und ermöglicht das Triggern einer Test-Pipeline für alle aktiven Testrezepten über das Submodell der VWS.
Testszenarien	Da die Methode keine Argumente akzeptiert, keine Logik implementiert, als reine Schnittstelle zwischen der VWS-Ebene und dem *Core*-Layer dient und nur die angegebene und bereits verifizierte Methode des Moduls *Test Orchestrator* aufruft, gilt seine Verifikation als abgeschlossen.

Tabelle 5.148: Funktionsbeschreibung: *Test Orchestration Service* - `ExecutePassiveTestPipelines`

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für alle Testrezepte, die ein Testbericht enthalten `string ExecuteProcessedTestPipelines()`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst kapselt die durch Tabelle 5.98 beschriebene Methode und ermöglicht das Triggern einer Test-Pipeline für alle verarbeiteten Testrezepten über das Submodell der VWS.
Testszenarien	Da die Methode keine Argumente akzeptiert, keine Logik implementiert, als reine Schnittstelle zwischen der VWS-Ebene und dem *Core*-Layer dient und nur die angegebene und bereits verifizierte Methode des Moduls *Test Orchestrator* aufruft, gilt seine Verifikation als abgeschlossen.

Tabelle 5.149: Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteProcessedTestPipelines`

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für alle Testrezepte, die kein Testbericht enthalten `string ExecuteUnprocessedTestPipelines()`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst kapselt die durch Tabelle 5.99 beschriebene Methode und ermöglicht das Triggern einer Test-Pipeline für alle nicht verarbeiteten Testrezepten über das Submodell der VWS.
Testszenarien	Da die Methode keine Argumente akzeptiert, keine Logik implementiert, als reine Schnittstelle zwischen der VWS-Ebene und dem *Core*-Layer dient und nur die angegebene und bereits verifizierte Methode des Moduls *Test Orchestrator* aufruft, gilt seine Verifikation als abgeschlossen.

Tabelle 5.150: Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteUnprocessedTestPipelines`

Komponente	Test Orchestration Service
Signatur	Durchführen einer Test-Pipeline für alle Testrezepte `string ExecuteTestPipelines()`
Return	Benachrichtigung über den Erfolg der Operation oder `ArgumentException`
Beschreibung	Dieser Dienst ermöglicht das Triggern einer Test-Pipeline für alle Testrezepten über das Submodell der VWS.
Testszenarien	Da die Methode keine Argumente akzeptiert, keine Logik implementiert, als reine Schnittstelle zwischen der VWS-Ebene und dem *Core*-Layer dient und nur die angegebene und bereits verifizierte Methode des Moduls *Test Orchestrator* aufruft, gilt seine Verifikation als abgeschlossen.

Tabelle 5.151: Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteTestPipelines`

Testszenarien

Identifikation	TC.TOS01
Testszenario	Durchführung einer Test-Pipeline für ein Testobjekt durch Eingabe eines Ticket-Identifikators, dessen Wert `null` oder leerer Zeichenkette entspricht
Testgruppe	`Demonstrator.Tests.TestOrchestrationServiceTests`
Funktionalität	`string ExecuteTestPipelineById(string ticketId)`
Funktionsbeschreibung	siehe Tabelle 5.145
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der Methode mit einem ungültigen Ticket-Identifikator als Argument zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Aufruf der Methode mit dem Wert `null` oder einer leeren Zeichenkette als Argument
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.152: Teststrategie: *Test Orchestration Service* - TC.TOS01

Identifikation	TC.TOS02
Testszenario	Durchführung einer Test-Pipeline für ein Testobjekt durch Eingabe eines Ticket-Identifikators, mit welchem kein Ticket assoziierbar ist
Testgruppe	`Demonstrator.Tests.TestOrchestrationServiceTests`
Funktionalität	`string ExecuteTestPipelineById(string ticketId)`
Funktionsbeschreibung	siehe Tabelle 5.145
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der Methode mit einem nicht zugeordneten Ticket-Identifikator als Argument zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Aufruf der Methode mit einem zufälligen Ticket-Identifikator als Argument
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.153: Teststrategie: *Test Orchestration Service* - TC.TOS02

Identifikation	TC.TOS03
Testszenario	Erfolgreiche Durchführung einer Test-Pipeline für ein Testobjekt durch Eingabe eines Ticket-Identifikators
Testgruppe	`Demonstrator.Tests.TestOrchestrationServiceTests`
Funktionalität	`string ExecuteTestPipelineById(string ticketId)`
Funktionsbeschreibung	siehe Tabelle 5.145
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der Methode mit einem gültigen Ticket-Identifikator zu einer erfolgreichen Durchführung und Protokollierung der Test-Pipeline führt und die entsprechende Benachrichtigung zurückgibt.
Vorgehen	Erstellen und Hochladen einer Ticket-Instanz und darauffolgende Initiierung einer Test-Pipeline durch Eingabe seiner Identifikationsnummer.
Erwartung	Die Methode initialisiert das `TestReport`-Attribut der Ticket-Instanz und gibt die passende Benachrichtigung zurück.

Tabelle 5.154: Teststrategie: *Test Orchestration Service* - TC.TOS03

Identifikation	TC.TOS04
Testszenario	Durchführung einer Test-Pipeline für ein Testobjekt durch Eingabe einer URI-Adresse, deren Wert `null` oder leerer Zeichenkette entspricht
Testgruppe	`Demonstrator.Tests.TestOrchestrationServiceTests`
Funktionalität	`string ExecuteTestPipelineByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.146
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der Methode mit einer ungültigen Uri-Adresse als Argument zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Aufruf der Methode mit dem Wert `null` oder einer leeren Zeichenkette als Argument
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.155: Teststrategie: *Test Orchestration Service* - TC.TOS04

Identifikation	TC.TOS05
Testszenario	Durchführung einer Test-Pipeline für ein Testobjekt durch Eingabe einer URI-Adresse, mit welchem kein aktives Ticket assoziierbar ist
Testgruppe	`Demonstrator.Tests.TestOrchestrationServiceTests`
Funktionalität	`string ExecuteTestPipelineByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.146
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der Methode mit einer nicht zugeordneten URI-Adresse als Argument zu einer Ausnahme vom Typ `ArgumentException` führt.
Vorgehen	Aufruf der Methode mit einer gültigen URI-Adresse bei nicht hochgeladenen Test-Tickets
Erwartung	Die Methode wirft eine Ausnahme vom Typ `ArgumentException`.

Tabelle 5.156: Teststrategie: *Test Orchestration Service* - TC.TOS05

Identifikation	TC.TOS06
Testszenario	Erfolgreiche Durchführung einer Test-Pipeline für ein Testobjekt durch Eingabe einer URI-Adresse
Testgruppe	`Demonstrator.Tests.TestOrchestrationServiceTests`
Funktionalität	`string ExecuteTestPipelineByUri(string uri)`
Funktionsbeschreibung	siehe Tabelle 5.146
Beschreibung	Durch diesen Testfall wird sichergestellt, dass das Aufrufen der Methode mit einer gültigen URI-Adresse bei hochgeladenem aktiven Test-Ticket zu einer erfolgreichen Durchführung und Protokollierung der Test-Pipeline führt und die entsprechende Benachrichtigung zurückgibt.
Vorgehen	Erstellen und Hochladen einer aktiven Ticket-Instanz und darauffolgende Initiierung einer Test-Pipeline durch Eingabe der bei der Erstellung spezifizierten URI-Adresse.
Erwartung	Die Methode initialisiert das `TestReport`-Attribut der Ticket-Instanz und gibt die passende Benachrichtigung zurück.

Tabelle 5.157: Teststrategie: *Test Orchestration Service* - TC.TOS06

Implementierung

Zum Zweck der Implementierung des für die Testdurchführung verantwortlichen Submodells übernimmt der *Test Orchestration Service* die Funktionalitäten der Komponente *Test Orchestrator*. Aus diesem Grund erfordert der Konstruktor des Dienstes die Übergabe mehrerer Kernkomponenten. Diese werden nach dem *Dependency Inversion* Prinzip dem Dienst bereitgestellt, wobei als abstrakte Abhängigkeiten die Module *Test Orchestrator*, *Test Object Provider*, *Test Runner* und die entsprechenden Testklassen gelten. Während die meisten Methoden keine Logik implementieren und als reine Schnittstellen zwischen der Ebene der VWS und dem *Core*-Layer gelten, erfolgt die Abstraktion aller Funktionalitäten, welche architekturspezifische Datentypen akzeptieren, durch die Methoden des Orchestrierungsdienstes.

Auswertung

Abb. 5.17 dient als Überblick über die erfolgreich durchgeführten Testszenarien.

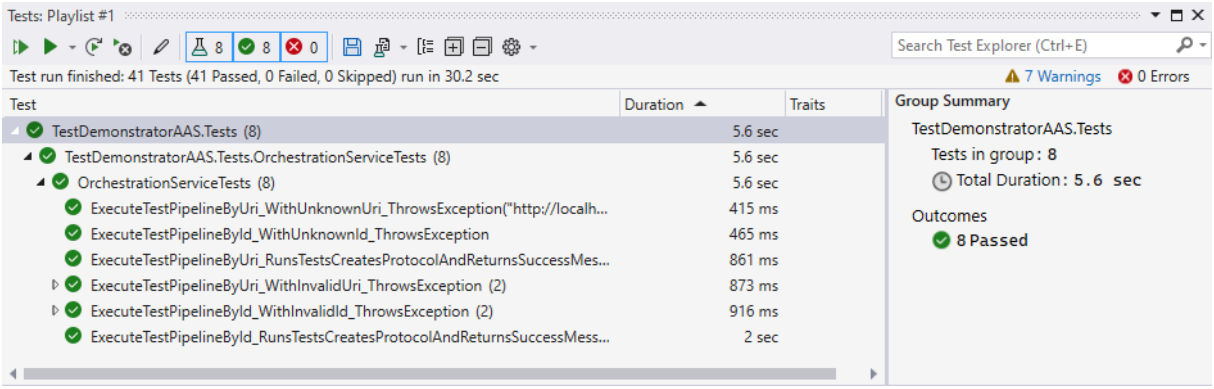


Abbildung 5.17: Testergebnisse - *Test Orchestration Service*

5.5 Verwaltungsschale

Die Implementierung der VWS der Testlösung findet auf Basis der zwei Service-Komponenten und mittels der Softwareumgebung *Eclipse BaSyx* statt. Zum Zweck der Aufteilung der Softwarearchitektur in voneinander unabhängigen Softwareschichten findet die Erstellung eines zweiten Projekts in Visual Studio, das die übergeordneten Komponenten, nämlich die Service-Module und die VWS-Implementierung, enthält. Wie durch Abb. 4.29 dargestellt und im Rahmen der Konzeptionsphase spezifiziert wurde, erfolgt die Bereitstellung der Dienste der Service-Komponenten über die VWS durch die Operationen und Eigenschaften von zwei Submodellen. Während das erste Submodell für das Ticket Management verantwortlich ist, charakterisiert sich das zweite Teilmodell durch Operationen zum Initiieren des Testdurchführungsprozesses. Die Aufrufhierarchie wird entsprechend Abb. 4.11 umgesetzt. Die zwei Teilmodelle entsprechen den Kernelementen der vorgeschlagenen Testautomatisierungslösung. Eine Ergänzung der VWS um weitere Submodelle wird zum Zweck des produktiven Einsatzes und im Rahmen zukünftiger Arbeiten als notwendig betrachtet. Im Folgenden werden die Submodelle der zum Zweck der Konzeptprüfung entwickelten Verwaltungsschale beschrieben.

5.5.1 Ticket Management Submodel

In diesem Unterabschnitt erfolgt eine tabellarische Beschreibung der implementierten Elemente des *Ticket Management* Submodells.

Submodell-Elemente

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`TicketsCount`
Typ	`Property<int>`
Beschreibung	Über diese Eigenschaft kann die Anzahl der Tickets im Verzeichnis abgefragt werden
Ablauf	siehe Abb. 5.18

Tabelle 5.158: Submodellelement: *Ticket Management SM* - `TicketsCount`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`PassiveTicketsCount`
Typ	`Property<int>`
Beschreibung	Über diese Eigenschaft kann die Anzahl der passiven Tickets im Verzeichnis abgefragt werden
Ablauf	siehe Abb. 5.18, die Anzahl der passiven Test-Rezepte wird nach einer einfachen Filterung auf Basis des Ticketattributs `Type` abgefragt.

Tabelle 5.159: Submodellelement: *Ticket Management SM* - `PassiveTicketsCount`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`ActiveTicketsCount`
Typ	`Property<int>`
Beschreibung	Über diese Eigenschaft kann die Anzahl der aktiven Tickets im Verzeichnis abgefragt werden
Ablauf	siehe Abb. 5.18, die Anzahl der aktiven Test-Rezepte wird nach einer einfachen Filterung auf Basis des Ticketattributs `Type` abgefragt.

Tabelle 5.160: Submodellelement: *Ticket Management SM* - `ActiveTicketsCount`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`ProcessedTicketsCount`
Typ	`Property<int>`
Beschreibung	Über diese Eigenschaft kann die Anzahl der Tickets, die ein Testbericht enthalten, abgefragt werden
Ablauf	siehe Abb. 5.18, die Anzahl der Test-Rezepte, die ein Testbericht enthalten wird nach einer einfachen Filterung auf Basis des Ticketattributs `TestReport` abgefragt.

Tabelle 5.161: Submodellelement: *Ticket Management SM* - `ProcessedTicketsCount`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`UnprocessedTicketsCount`
Typ	`Property<int>`
Beschreibung	Über diese Eigenschaft kann die Anzahl der Tickets, die kein Testbericht enthalten, abgefragt werden
Ablauf	siehe Abb. 5.18, die Anzahl der Test-Rezepte, die ein Testbericht enthalten wird nach einer einfachen Filterung auf Basis des Ticketattributs `TestReport` abgefragt.

Tabelle 5.162: Submodellelement: *Ticket Management SM* - `UnprocessedTicketsCount`

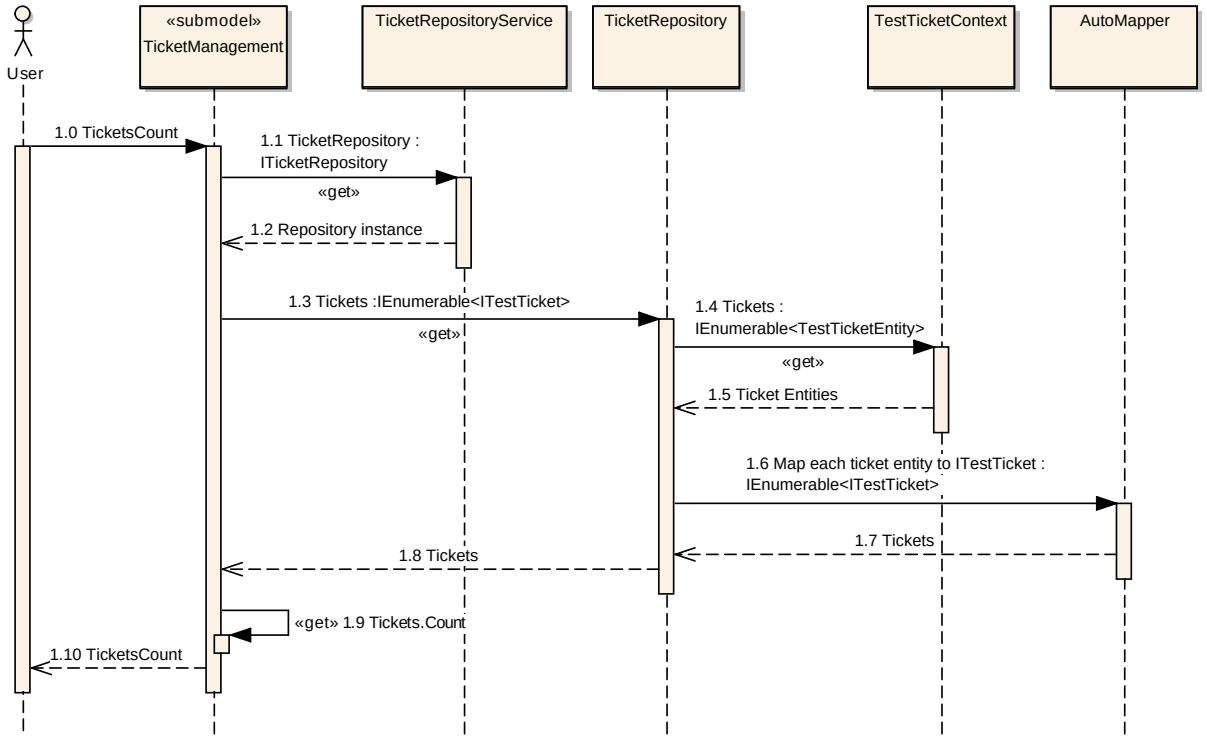


Abbildung 5.18: Submodellelement `TicketsCount` - Anfragebearbeitung

Abbildung 5.18 veranschaulicht den Ablauf nach dem Aussenden einer HTTP-Anfrage zum Auslesen der Anzahl von Test-Rezepten. Die Operation verwendet den *Ticket Repository Service*, um die konkrete Implementierung des Ticketverzeichnisses abzurufen. Darauffolgend wird die Anzahl der Tickets ausgelesen und entsprechend bereitgestellt. Die Notwendigkeit der Komponenten *Test Ticket Context* und *AutoMapper* wurde bereits im Rahmen des Kapitels erläutert.

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`UploadActiveTicket`
Typ	`Operation`
Input	• URI-Adresse des Testobjekts • Strukturdeskriptor in XML-Format • Zustandsautomatendeskriptor in XML-Format
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Hochladen aktiver Testrezepten verantwortlich

Tabelle 5.163: Submodellelement: *Ticket Management SM* - `UploadActiveTicket`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`UploadPassiveTicket`
Typ	`Operation`
Input	• Typ 1 VWS in XML-Format • Strukturdeskriptor in XML-Format
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Hochladen passiver Testrezepten verantwortlich

Tabelle 5.164: Submodellelement: *Ticket Management SM* - `UploadPassiveTicket`

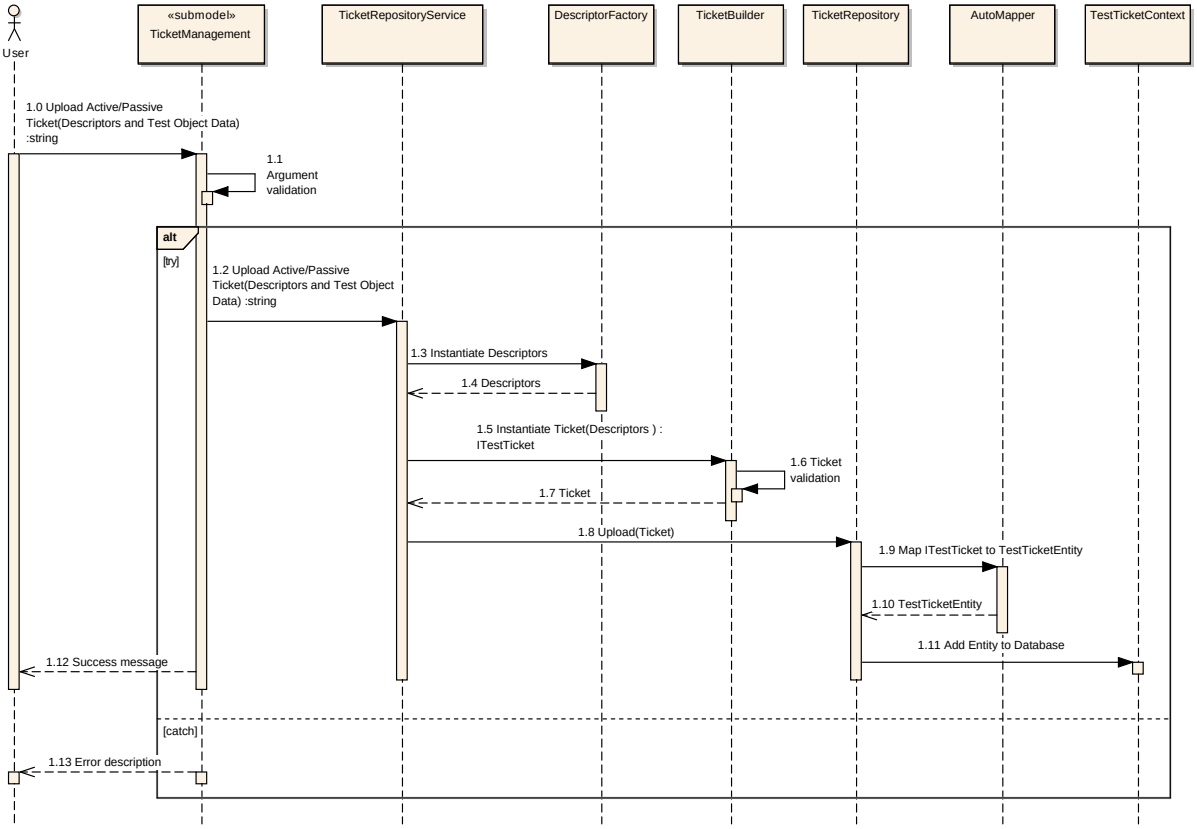


Abbildung 5.19: Submodellelement `UploadTicket` - Anfragebearbeitung

Abbildung 5.19 illustriert den vereinfachten Ablauf bei dem Aussenden einer HTTP-Anfrage zum Hochladen eines Test-Tickets. Im Rahmen der Bearbeitung findet eine Validierung und Instanziierung der Ticket-Deskriptoren statt. Im nächsten Schritt erfolgt das Erstellen, Validieren und Hochladen des erstellten Test-Rezeptes. Für den Fall, dass ein Fehler auftritt, wird dem Anwender die entsprechende Benachrichtigung gesendet.

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`UpdateActiveTicket`
Typ	`Operation`
Input	• Identifikationsnummer des zu aktualisierenden Tickets • URI-Adresse des Testobjekts • Strukturdeskriptor in XML-Format • Zustandsautomatendeskriptor in XML-Format
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Aktualisieren aktiver Testrezepten verantwortlich

Tabelle 5.165: Submodellelement: *Ticket Management SM* - `UpdateActiveTicket`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`UpdatePassiveTicket`
Typ	`Operation`
Input	• Identifikationsnummer des zu aktualisierenden Tickets • Typ 1 VWS in XML-Format • Strukturdeskriptor in XML-Format
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Aktualisieren passiver Testrezepten verantwortlich

Tabelle 5.166: Submodellelement: *Ticket Management SM* - `UpdatePassiveTicket`

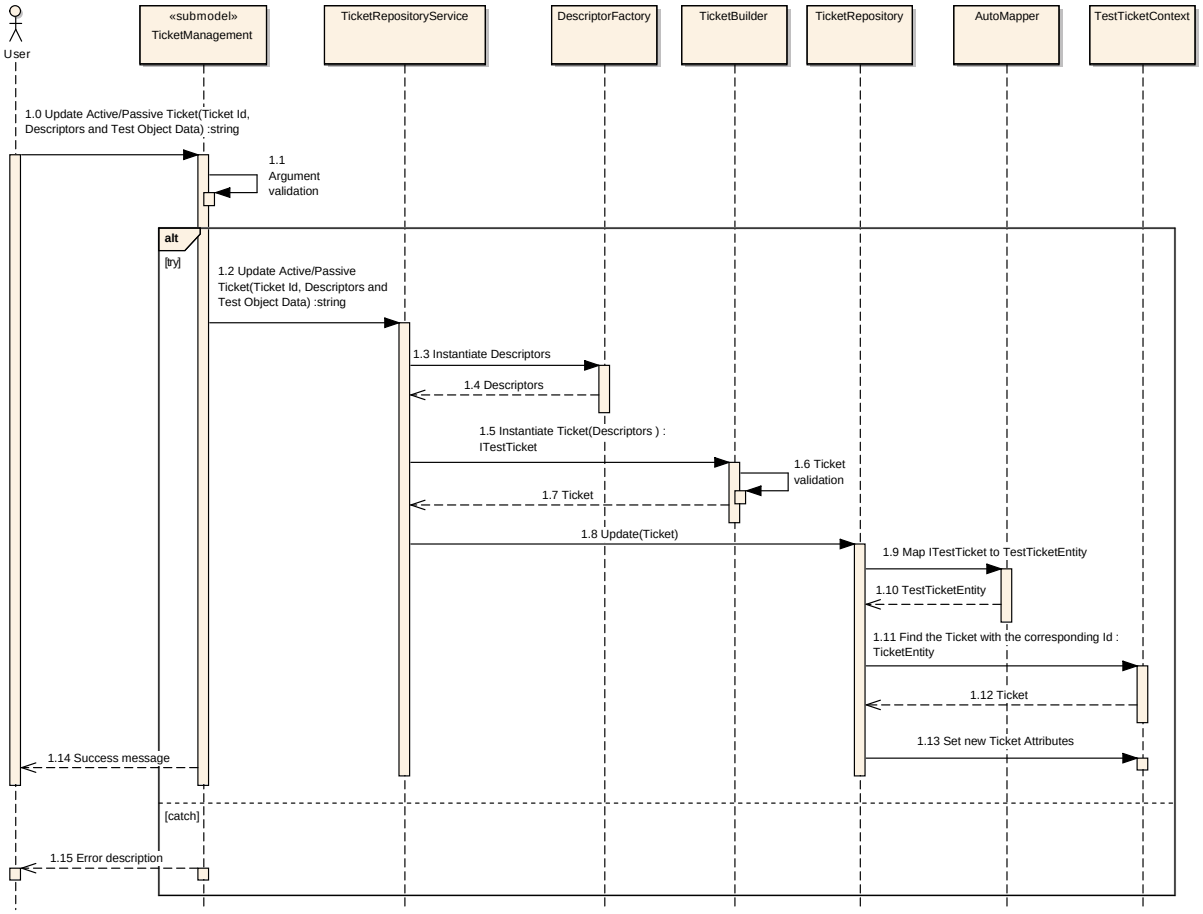


Abbildung 5.20: Submodellelement `UploadTicket` - Anfragebearbeitung

Das durch Abb. 5.20 dargestellte Sequenzdiagramm visualisiert den Ablauf der Aktualisierung einer Ticket-Entität. Durch die Eingabe der Identifikationsnummer des zu aktualisierenden Test-Rezeptes und auf Basis der bereitgestellten Ticket-Deskriptoren erfolgt die Identifikation des Tickets und die Aktualisierung seiner Attribute.

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`DeleteTicket`
Typ	`Operation`
Input	Identifikationsnummer des zu löschenden Tickets
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Löschen von Testrezepten aus dem Verzeichnis verantwortlich

Tabelle 5.167: Submodellelement: *Ticket Management SM* - `DeleteTicket`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`RetrieveTicketById`
Typ	`Operation`
Input	• Identifikationsnummer des zu löschenden Tickets
Output	• Erstellungszeitstempel des abgerufenen Tickets • Aktualisierungszeitstempel (wenn vorhanden) • Typ des Tickets • URI-Adresse der VWS des Testobjekts (wenn vorhanden) • VWS des Testobjekts in XML-Format (wenn vorhanden) • Strukturdeskriptor in XML-Format • Zustandsautomatendeskriptor in XML-Format (wenn vorhanden) • Testbericht in XML-Format (wenn vorhanden)
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Abrufen von Testrezepten verantwortlich

Tabelle 5.168: Submodellelement: *Ticket Management SM* - `RetrieveTicketById`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`RetrieveTicketByUri`
Typ	`Operation`
Input	• URI-Adresse der VWS eines Testobjekts, das durch ein Ticket im Verzeichnis beschrieben wird
Output	• Erstellungszeitstempel des abgerufenen Tickets • Aktualisierungszeitstempel (wenn vorhanden) • Typ des Tickets • URI-Adresse der VWS des Testobjekts (wenn vorhanden) • VWS des Testobjekts in XML-Format (wenn vorhanden) • Strukturdeskriptor in XML-Format • Zustandsautomatendeskriptor in XML-Format (wenn vorhanden) • Testbericht in XML-Format (wenn vorhanden)
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Abrufen von Testrezepten verantwortlich

Tabelle 5.169: Submodellelement: *Ticket Management SM* - `RetrieveTicketByUri`

Komponente	Test Demonstrator AAS
Submodell	`TicketManagement`
Eigenschaft	`RetrieveTicketIds`
Typ	`Operation`
Output	Array aus GUID-Identifikatoren
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für das Abrufen der Identifikationsdaten aller Tickets verantwortlich

Tabelle 5.170: Submodellelement: *Ticket Management SM* - `RetrieveTicketIds`

5.5.2 Test Execution Submodel

In diesem Unterabschnitt befindet sich die Beschreibung der Bestandteile des *Test Execution* Submodells.

Komponente	Test Demonstrator AAS
Submodell	`TestExecution`
Eigenschaft	`ExecuteTestPipelineById`
Typ	`Operation`
Input	Ticket Identifikator (GUID)
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für die Initiierung einer Test-Pipeline für ein Testobjekt auf Basis seines Ticket-Identifikators verantwortlich

Tabelle 5.171: Submodellelement: *Test Execution SM* - `ExecuteTestPipelineById`

Komponente	Test Demonstrator AAS
Submodell	`TestExecution`
Eigenschaft	`ExecuteTestPipelineByUri`
Typ	`Operation`
Input	Die URI-Adresse eines Testobjekts, das durch ein hochgeladenes Test-Rezept charakterisiert ist
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für die Initiierung einer Test-Pipeline für ein Testobjekt auf Basis seiner URI-Adresse verantwortlich

Tabelle 5.172: Submodellelement: *Test Execution SM* - `ExecuteTestPipelineById`

Komponente	Test Demonstrator AAS
Submodell	`TestExecution`
Eigenschaft	`ExecuteActiveTestPipelines`
Typ	`Operation`
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für die Initiierung einer Test-Pipeline für alle Test-Rezepte verantwortlich, die ein aktives Testobjekt beschreiben

Tabelle 5.173: Submodellelement: *Test Execution SM* - `ExecuteActiveTestPipelines`

Komponente	Test Demonstrator AAS
Submodell	`TestExecution`
Eigenschaft	`ExecutePassiveTestPipelines`
Typ	`Operation`
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für die Initiierung einer Test-Pipeline für alle Test-Rezepte verantwortlich, die ein passives Testobjekt beschreiben

Tabelle 5.174: Submodellelement: *Test Execution SM* - `ExecutePassiveTestPipelines`

Komponente	Test Demonstrator AAS
Submodell	`TestExecution`
Eigenschaft	`ExecuteTestPipelines`
Typ	`Operation`
Return	Benachrichtigung über den Abschlussstatus der Operation
Beschreibung	Die Operation ist für die Initiierung einer Test-Pipeline für alle hochgeladenen Test-Rezepte verantwortlich

Tabelle 5.175: Submodellelement: *Test Execution SM* - `ExecuteTestPipelines`

Das durch Abb. A.7 dargestellte Sequenzdiagramm stellt die vereinfachte Sequenz der Durchführung einer Test-Pipeline für alle hochgeladenen Testrezepten auf softwaretechnischer Ebene dar. Die Ableitung und Beschreibung der Softwarearchitektur, welche den illustrierten und im Rahmen der Konzepterstellung beschriebenen Ablauf ermöglicht, gilt zusammen mit der erfolgreichen und testgetriebenen Implementierung der zugrundeliegenden Komponenten und Funktionalitäten als das Ergebnis dieser Abschlussarbeit.

5.6 Testklassen

Zum Zweck der Konzeptprüfung fand eine Entwicklung von zwei Testklassen statt, welche die strukturelle Verifikation von Verwaltungsschalen und zustandsorientierte Testung von Führungskomponenten ermöglichen. In diesem Unterabschnitt erfolgt die Beschreibung der zwei Testmodule. Diese werden als Stufen der Test-Pipeline eingesetzt und durch den Einsatz des Test-Frameworks *NUnit* implementiert. Die Testklassen sollen also durch die Architekturkomponente `NUnitTestRunner` ausgeführt werden. Alle zum Zweck dieser Ausführung notwendigen Informationen sind als Teil der Klasse `TestObject` verfügbar, wodurch die Testmodule auf Basis einer einzigen Instanz operieren können. Die Implementierung der Testkomponente findet nach dem 3A-Muster statt. Für die Erstellung des Testprotokolls wird die .NET Klasse `ConsoleTraceListener` verwendet. Diese gilt als Bestandteil des Namensraums `System.Diagnostics`.

Listing A.1 gilt als Vorlage bei der Erstellung von Testklassen im Kontext der vorgeschlagenen Softwarearchitektur. Die Methode mit der Bezeichnung `OneTimeSetUp` wird von der `NUnitTestRunner`-Instanz zum Beginn der Ausführung der entsprechenden Test-Stufe aufgerufen. Diese instanziiert die Klasse `ConsoleTraceListener` und ruft das aktuelle Testobjekt ab. In Abhängigkeit vom Testfall kann die Methode erweitert werden. Die mit dem `OneTimeTearDown`-Attribut gekennzeichnete Methode entspricht der zuletzt aufgerufenen Funktionalität jeder Testebene und wird für das Speichern des Testprotokolls eingesetzt. Die Methoden `SetUp` und `TearDown` werden vor bzw. nach jedem Testfall ausgeführt. Die Erstellung des Testprotokolls findet automatisch durch das *NUnit*-Framework in XML-Format[9] statt. Auf Basis von Visualisierungswerkzeugen wie *nunit3viewer*[10] und *Extent Reports*[11] können die Bestandteile des Testprotokolls anwenderfreundlich dargestellt werden.

5.6.1 Strukturtests

Im Rahmen der ersten Pipeline-Stufe wird die strukturelle Organisation der Verwaltungsschale des Testobjekts mit der vom Strukturdeskriptor definierten Beschreibung verglichen. Zum Zweck des Verifikationsprozesses wird eine rekursive Funktion implementiert, welche das sequenzielle Traversieren des gesamten Baums aus Submodellen, Elementen und Kollektionen ermöglicht. Bei jedem Schritt der Traversierung werden die Charakteristiken (Identifikator, Anzahl von untergeordneten Elementen u.a.) des aktuellen VWS-Elements mit den Informationen in der Strukturbeschreibung verglichen. Die Implementierung des Testfalls wird durch das folgende Sequenzdiagramm illustriert.

[9] `https://docs.nunit.org/articles/nunit/technical-notes/usage/Test-Result-XML-Format.html`
[10] `https://github.com/tkovacs-dev/nunit3viewer`
[11] `https://www.extentreports.com/`

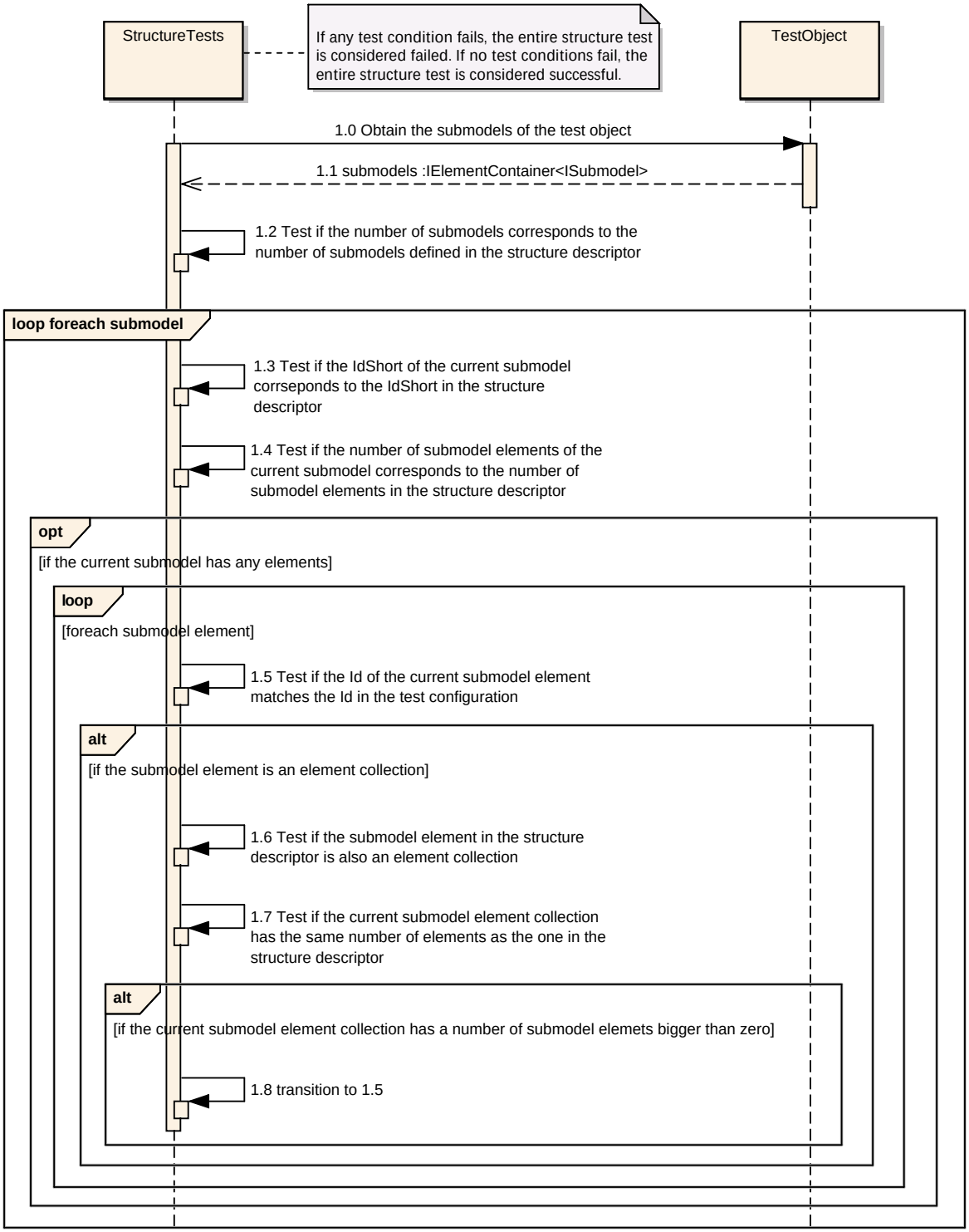


Abbildung 5.21: Sequenzdiagramm: Strukturverifikation

Die dargestellte Sequenz führt zur vollständigen Verifikation der kompletten Struktur einer Verwaltungsschale. Damit den Test erfolgreich abgeschlossen wird, muss die Struktur des Testobjekts dem Strukturdeskriptor auf 100% entsprechen. Abweichungen der Identifikationsdaten, der Anzahl und Reihenfolge von Strukturelementen dienen als Voraussetzungen für eine ohne Erfolg abgeschlossene Pipeline-Stufe.

5.6.2 Zustandsbezogene Tests

Wie im dritten Kapitel erläutert, muss für die Beschreibung des Soll-Verhaltens einer Führungskomponente ein Zustandsautomatendeskriptor als Teil des Test-Rezeptes vorgelegt werden. Zum Zweck der formellen Überprüfung seiner Gültigkeit erfolgt die Spezifikation eines XML-Schemas, das die zulässigen XML-Elemente zusammen mit ihren Attributen beschreibt. Der realisierte Deskriptor setzt sich aus drei Hauptelementen zusammen - `onetimesetup`, `paths` und `onetimeteardown`. Listing 5.6 veranschaulicht die allgemeine Struktur eines Zustandsautomatendeskriptors.

```
1  <?xml version="1.0" encoding="utf-16" ?>
2  <descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="StateMachineDescriptorXsd.xsd">
3    <onetimesetup>
4      ...
5    </onetimesetup>
6    <paths>
7      ...
8    </paths>
9    <onetimeteardown>
10     ...
11   </onetimeteardown>
12 </descriptor>
```

Listing 5.2: Allgemeine Struktur eines Zustandsautomatendeskriptors

Durch das erste und das letzte Element erfolgt die Spezifikation der Aktionen, welche vor und nach dem gesamten Testablauf duchzuführen sind. Diese Sektionen gelten als optional. Jede Aktion setzt sich aus mehreren Attributen zusammen und definiert die Identifikationsdaten und Werte der VWS-Eigenschaften, welche zum Zweck der Vor- bzw. Nachbereitung des Testobjekts und entsprechend dem Anwendungsfall zu setzen sind. Des Weiteren enthält die Aktion auch Daten, die das Auslesen der spezifizierten Eigenschaften gewährleisten. Dadurch kann den Erfolg der Vor- bzw. Nachbereitungsphase garantiert werden. Während die Aktionen als Testszenarien gelten und in Zusammenhang mit dieser Abschlussarbeit nur für die Verifikation des Zustandsautomaten einer Führungskomponente eingesetzt werden, bilden diese eine generelle Basis für die Beschreibung von atomaren Unit-Tests im Kontext der VWS. Listing 5.3 stellt die Informationsstruktur eines `action`-Elements dar.

```
1  ...
2    <action submodel="SM_Unit1_(SuperiorStateMachine)"
         setter="SMC_Statemachine/Prop_ControlCommand" getter="SMC_Statemachine/Prop_State"
         set="8" setType="System.Int32" expect="Aborted" expectType="System.String"
         wait="100"/>
3  ...
```

Listing 5.3: Allgemeine Struktur einer Aktion im Kontext des Zustandsautomatendeskriptors

Die `paths`-Struktur beinhaltet alle Pfaden, die zum Zweck der Verifikation des Zustandsautomaten durchgequert werden müssen. Vor der Definition jedes Pfades kann eine Reihe aus weiteren Vor- bzw. Nachbereitungsaktionen im Rahmen einer `setup`- bzw. `teardown`-Sektion spezifiziert werden. Die Angabe des Submodells und der Eigenschaften zum Setzen bzw. Auslesen des aktuellen Zustands der Führungskomponente erfolgt durch das Element `properties`. Des Weiteren charakterisiert sich jeden Pfad durch eine Reihe aus Zuständen, die durch die entsprechende Zeichenkette identifiziert werden. Zu jedem `state`-Element gehört eine Transition. Anhand dieses XML-Elements wird die Übergangsbedingung zum nachfolgend spezifizierten und erwarteten Zustand definert. Durch das Attribut `wait` wird die Zeit zum Durchführen der Transition in Millisekunden angegeben. Listing 5.4 veranschaulicht die Struktur eines

`path`-Elements.

```
1  ...
2    <path>
3      <setup>
4        <!--Setting initial state-->
5        <action submodel="SM_Unit1_(SuperiorStateMachine)"
             setter="SMC_Statemachine/Prop_ControlCommand" getter="SMC_Statemachine/Prop_State"
             set="8" setType="System.Int32" expect="Aborted" expectType="System.String"
             wait="100"/>
6        <!--Booking the first subunit-->
7        <action submodel="SM_Unit2_(SubUnit1)" setter="SMC_Statemachine/Prop_UnitSelect"
             getter="SMC_Statemachine/Prop_UnitSelect" set="1" setType="System.Int32"
             expect="True" expectType="System.Boolean" wait="100"/>
8        <!--Booking the second subunit-->
9        <action submodel="SM_Unit3_(SubUnit2)" setter="SMC_Statemachine/Prop_UnitSelect"
             getter="SMC_Statemachine/Prop_UnitSelect" set="1" setType="System.Int32"
             expect="True" expectType="System.Boolean" wait="100"/>
10     </setup>
11     <!--Specifying read/write properties-->
12     <properties submodel="SM_Unit1_(SuperiorStateMachine)"
           setter="SMC_Statemachine/Prop_ControlCommand" getter="SMC_Statemachine/Prop_State"/>
13     <!--Test path definition-->
14     <state id="Aborted">
15       <transition event="Clear" target="Stopped" wait="100"/>
16     </state>
17     <state id="Stopped">
18       <transition event="Reset" target="Idle" wait="100"/>
19     </state>
20     <state id="Idle">
21       <transition event="Start" target="Execute" wait="100"/>
22     </state>
23     <teardown>
24       <!--Unbooking the first subunit-->
25       <action submodel="SM_Unit2_(SubUnit1)" setter="SMC_Statemachine/Prop_UnitSelect"
             getter="SMC_Statemachine/Prop_UnitSelect" set="0" setType="System.Int32"
             expect="False" expectType="System.Boolean" wait="100"/>
26       <!--Unbooking the second subunit-->
27       <action submodel="SM_Unit3_(SubUnit2)" setter="SMC_Statemachine/Prop_UnitSelect"
             getter="SMC_Statemachine/Prop_UnitSelect" set="0" setType="System.Int32"
             expect="False" expectType="System.Boolean" wait="100"/>
28     </teardown>
29   </path>
30 ...
```

Listing 5.4: Allgemeine Struktur eines Pfades im Kontext des Zustandsautomatendeskriptors

Die hiermit vorgeschlagene Informationsstruktur eines Zustandsautomatendeskriptors ist auch für verschachtelte Zustandsmaschinen geeignet. Eine beispielhafte Beschreibung des *PackML*-Zustandsautomaten befindet sich unter A.4. Das vorgestellte Metamodell wird durch Listing A.3 beschrieben. Zum Zweck der Deserialisierung einer anhand des Schemas definierten und konkreten Zustandsautomatenbeschreibung fand die Implementierung der Klasse `StateMachineDescriptor` statt. Für die technische Umsetzung des Testalgorithmus erfolgt die Erstellung der Klasse `StateMachineTests`. Diese entspricht der Vorlage in A.1. Der Verifikationsprozess setzt sich aus zwei Phasen zusammen. Im Rahmen der ersten und sekundären Phase erfolgt das Sicherstellen, dass alle durch die Aktionen spezifizierten Submodellen und Eigenschaften als gültige Elemente der VWS des Testobjekts gelten. Im Verlauf der primären Testphase findet die Traversierung der Zustandsmaschine statt. Dieser Prozess wird konzeptionell durch das folgende Sequenz-

diagramm dargestellt. Die UML-konforme Beschreibung der Vor- und Nachbereitungsphasen befindet sich unter A.8.

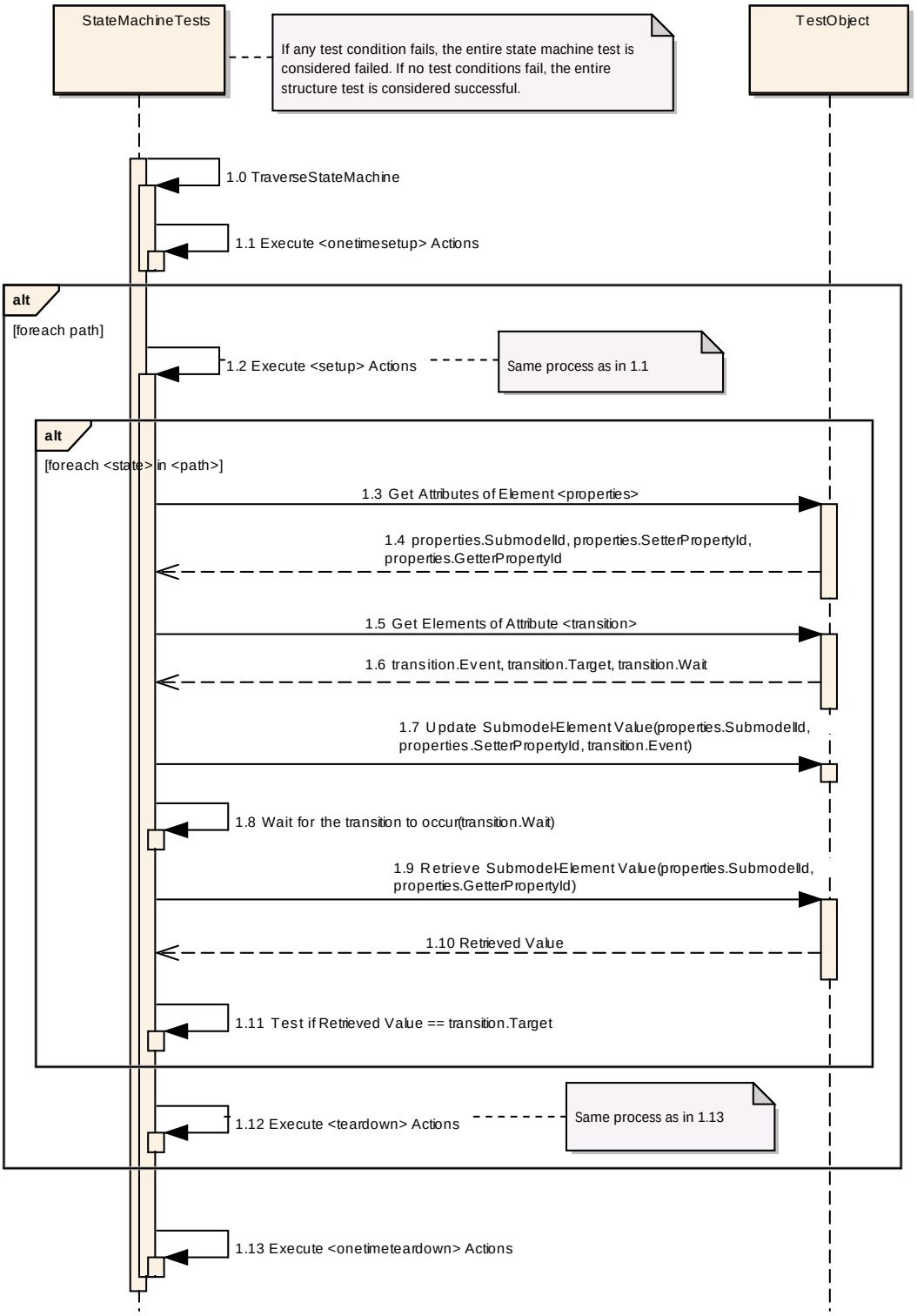


Abbildung 5.22: Traversierung des Zustandsautomaten anhand der Informationen im Zustandsautomatendeskriptor

Aus dem Sequenzdiagramm geht hervor, dass alle durch den Deskriptor spezifizierten Testszenarien sequenziell abgearbeitet werden. Die Nutzung des Deskriptors setzt voraus, dass die Eigenschaften zum Auslesen und zum Setzen des aktuellen Zustands der Führungskomponente im gleichen Submodell definiert sind.

5.7 Inbetriebnahme

Zum Zweck der Inbetriebnahme findet eine Instanziierung der implementierten Klassen nach dem *Dependency Injection* Entwurfsmuster statt. Die Spezifikation der Adresse des SQL-Servers erfolgt durch den Einsatz einer Projektkonfigurationsdatei mit dem Namen `appsettings.json`. Zum Auslesen der Datei wird eine Klasse mit dem Bezeichner `ConfigurationBuilder` verwendet. Im nächsten Schritt findet die Übergabe der ausgelesenen Server-Adresse als Parameter der Methode einer `DbContextOptions`-Instanz statt. Diese spezifiziert alle für den Verbindungsaufbau notwendigen Informationen und wird als Argument des Konstruktors `TicketDbRepository` übergeben. Das folgende Listing veranschaulicht die Instanziierung der Softwarekomponenten. Das entsprechende Sequenzdiagramm wird durch Abb. A.6 dargestellt.

```
1  IConfigurationRoot configuration = new ConfigurationBuilder()
2    .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
3    .AddJsonFile("appsettings.json")
4    .Build();
5
6  DbContextOptions<TestTicketContext> options =
7    new DbContextOptionsBuilder<TestTicketContext>()
8      .UseSqlServer(configuration.GetConnectionString("DefaultConnection"))
9      .Options;
10
11 IDescriptorFactory descriptorFactory = new DescriptorFactory();
12 ITicketBuilder ticketBuilder = new TicketBuilder();
13 ITicketRepository ticketRepository = new TicketDbRepository(options);
14 ITestRunner testRunner = new NUnitTestRunner();
15 ITestObjectProvider testObjectProvider = new TestObjectProvider(ticketRepository);
16 ITestOrchestrator testOrchestrator = new TestOrchestrator();
17
18 ICollection<Type> testPipeline = new List<Type>()
19   { typeof(StructureTests), typeof(StateMachineTests) };
20
21 ITicketRepositoryService repositoryService = new TicketRepositoryService
22   (ticketRepository, ticketBuilder, descriptorFactory);
23
24 ITestOrchestrationService orchestrationService = new TestOrchestrationService
25   (testOrchestrator, testObjectProvider, testRunner, testPipeline);
26
27 TestManagementShell testManagementAAS = new TestManagementShell
28   (orchestrationService, repositoryService);
29
30 RunAssetAdministrationShellServer
31   (testManagementAAS.AdministrationShell, httpPort, httpsPort);
```

Listing 5.5: Instanziierung der implementierten Softwarekomponenten

Von besonderem Interesse ist die Definition der Test-Pipeline. Hierfür werden die implementierten Testklassen als Elemente einer Liste aus Instanzen der abstrakten Klasse `Type` durch den Einsatz des `typeof`-Operators deklariert. Durch diese Schnittstelle wird die Erweiterbarkeit der Test-Pipeline garantiert. Der letzte Funktionsaufruf führt zum Starten des AAS-Servers. Die Struktur der VWS wird in A.17 dargestellt. Eine Inbetriebnahmeanleitung befindet sich unter A.23.

Zum Zweck der Durchführung eines Testprozesses findet das Hochladen aller lesbaren Typ 1 Verwaltungsschalen statt, die von der IDTA bereitgestellt werden. Dies erfordert ihre Umwandlung in XML-Dateien durch den Einsatz vom *AAS Package Explorer*. Die gleichen XML-Dateien werden auch als Strukturdeskriptoren eingesetzt, was als Voraussetzung für den Erfolg des strukturorientierten Testprozesses gilt.

Im nächsten Schritt findet die Generierung einer VWS durch den Einsatz des in [Sch21] vorgestellten Generators statt. Die VWS stellt eine Schnittstelle zu einem *PackML* Zustandsautomaten bereit und charakterisiert sich durch die in Abb. A.30 dargestellte Struktur, für weitere Informationen siehe [Sch22c]. Vom Interesse für die Verifikation des Zustandsautomaten sind die Submodelle *SM_Unit1_(SuperiorStateMachine)*, *SM_Unit2_(SubUnit2)* und *SM_Unit3_(SubUnit3)*. Diese bieten die Eigenschaften zum Setzen und Auslesen des aktuellen Zustands einer Zustandsmaschine an. Der erstellte Deskriptor ist unter A.4 zu finden. Aus der aktiven VWS wird eine Strukturbeschreibung, also eine passive VWS, durch den Einsatz von *Eclipse BaSyx* generiert und als Strukturdeskriptor eingesetzt. Abb. 5.23 veranschaulicht die hochgeladenen Test-Tickets.

	TicketId	CreatedAt	UpdatedAt	Type	Uri	PassiveShell	StructureDescriptor	StateMachineDescriptor	TestReport
1	7ECF6B7B-9585-42E9-A802-111780364FE1	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
2	ED0F1493-BF53-46FB-9901-163508B093D5	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
3	DD197B93-D79B-464C-807E-2A2B240FD66C	2022-10-19 17:43:1...	NULL	Active	http://localhost:5081/	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<descriptor xmlns:xsi="htt...	NULL
4	915CDFB0-6EBC-452B-A38C-2B48C7DC89F2	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
5	51AE9278-F306-42C6-8CB2-2D3CE1FF4ED0	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
6	B1FA5429-B108-429E-B613-2D534623CDE8	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
7	1FFE6608-5F4F-4C45-8356-397C6EA4E4A3	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
8	72D5A5D2-011A-41C9-A6C0-54CC141B9246	2022-10-19 15:56:1...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
9	38BF119F-C47E-47DF-9CE8-58A51B77E7D7	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
10	332B0C61-BBCF-4175-99A9-63D012426D2F	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
11	71FD37E5-B0EF-40F8-8980-6606FE1C33D4	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
12	5F1BB179-66C3-451D-B145-6E5C5BABF55A	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
13	E665763B-0619-49B5-9B29-8F9E8AB58103	2022-10-19 15:56:1...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
14	1A236629-0BC4-44A1-9A10-9B962D972587	2022-10-19 15:56:1...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
15	C93F09D0-4C34-4719-AFF3-A7E2BA904151	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
16	CE15CE88-CBDF-4C14-B5AF-B4B31D162617	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
17	AA7E897B-DC9D-450A-9F88-BC907803DB53	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
18	B041695F-2FA2-476E-AD9F-D2F445A89567	2022-10-19 15:55:3...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
19	CCBC0CA8-82F6-4269-876A-D3C485BC6D92	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
20	105E4597-B2D4-4B5C-ABD4-D6D59BFA909F	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
21	F02E3B03-3F8F-41F4-9D2C-D738CB69EA88	2022-10-19 15:56:1...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
22	1EED92F5-8707-49CB-B912-E504AD2E4C25	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
23	6225D9CC-80FD-4E63-9DF0-EC19E87B9BFF	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
24	FFA54219-0C18-4047-9FA3-EFB419CB160D	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
25	94706195-D81A-4226-B568-F617B474C338	2022-10-19 15:56:1...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL
26	B11BF8BA-4D26-4494-AFC4-FBF19841B62D	2022-10-19 15:56:0...	NULL	Passive	NULL	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	<aas:aasenv xmlns:xsi="http://www.w3.org/2001/XML...	NULL	NULL

Abbildung 5.23: Inhalt des Ticket-Verzeichnisses nach einem Upload von 26 Testrezepten

Abb. A.31 zeigt, dass die VWS der Testlösung eine korrekte Anzahl von Testrezepten bereitstellt. Nach dem erfolgreichen Upload findet die Initiierung einer Test-Pipeline für jedes Ticket im Verzeichnis statt. Verwendet wird die `ExecuteTestPipelines`-Operation (siehe Tabelle 5.175) des *TestExecution*-Submodells. Die Anfrage wird manuell gesendet, da die GUI keine explizite Spezifikation der Bearbeitungszeit anbietet. Es wird davon ausgegangen, dass die Durchführung aller Pipelines mehr als 60s dauern wird. Abb. 5.25 stellt den Inhalt des Verzeichnisses nach dem Testvorgang dar. Jedes Ticket charakterisiert sich durch einen Testbericht mit der folgenden Sektionen.

```
1 <?xml version="1.0" encoding="utf-16" ?>
2 <PipelineReport>
3   <StructureTests>
4     ...
5   </StructureTests>
6   <StateMachineTests>
7     ...
8   </StateMachineTests>
9 </PipelineReport>
```

Listing 5.6: Allgemeine Struktur eines Zustandsautomatendeskriptors

Jeder als Teil der Test-Pipeline definierte Testklasse (siehe Listing 5.5, Zeile: 19) wird im Testbericht automatisch eine Sektion zugewiesen. Die Inhalt der Sektion kann in Abhängigkeit vom eingesetzten

Test-Framework variieren oder vom Entwickler der Test-Klassen definiert werden. Ausschnitte aus dem Testprotokoll des aktiven Testobjekts befinden sich unter A.5 und A.6. Diese dienen als Nachweis für die Funktionalität der prototypisch entwickelten Testautomatisierungslösung und bestätigen die Realisierbarkeit der vorgeschlagenen Softwarearchitektur.

	TicketId	CreatedAt	UpdatedAt	Type	Uri	PassiveShell	StructureDescriptor	StateMachineDescript...	TestReport
1	7ECF6B...	2022-10-1...	2022-10-19 18:28:43.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
2	ED0F14...	2022-10-1...	2022-10-19 18:28:48.7...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
3	915CDF...	2022-10-1...	2022-10-19 18:29:00.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
4	51AE927...	2022-10-1...	2022-10-19 18:29:05.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
5	B1FA542...	2022-10-1...	2022-10-19 18:29:09.5...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
6	1FFE660...	2022-10-1...	2022-10-19 18:29:16.8...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
7	72D5A5...	2022-10-1...	2022-10-19 18:29:23.6...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
8	38BF119...	2022-10-1...	2022-10-19 18:29:30.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
9	332B0C...	2022-10-1...	2022-10-19 18:29:35.7...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
10	71FD37...	2022-10-1...	2022-10-19 18:29:38.9...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
11	E11977...	2022-10-1...	2022-10-19 18:30:09.3...	Active	http://localhost:5081/	NULL	<aas:aasenv xmln...	<descriptor xmlns:xsi...	<PipelineReport><StructureTests>...
12	5F1BB1...	2022-10-1...	2022-10-19 18:30:14.5...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
13	E665763...	2022-10-1...	2022-10-19 18:30:15.3...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
14	1A23662...	2022-10-1...	2022-10-19 18:30:26.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
15	C93F09...	2022-10-1...	2022-10-19 18:30:29.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
16	CE15CE...	2022-10-1...	2022-10-19 18:30:35.1...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
17	AA7E897...	2022-10-1...	2022-10-19 18:30:41.8...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
18	B041695...	2022-10-1...	2022-10-19 18:30:50.9...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
19	CCBC0C...	2022-10-1...	2022-10-19 18:30:59.9...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
20	105E459...	2022-10-1...	2022-10-19 18:31:06.2...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
21	F02E3B...	2022-10-1...	2022-10-19 18:31:07.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
22	1EED92...	2022-10-1...	2022-10-19 18:31:17.0...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
23	6225D9...	2022-10-1...	2022-10-19 18:31:21.5...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
24	FFA5421...	2022-10-1...	2022-10-19 18:31:27.8...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
25	9470619...	2022-10-1...	2022-10-19 18:31:28.7...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...
26	B11BF8...	2022-10-1...	2022-10-19 18:31:47.4...	Passive	NULL	<aas:aasenv x...	<aas:aasenv xmln...	NULL	<PipelineReport><StructureTests>...

Abbildung 5.24: Inhalt des Ticket-Verzeichnisses nach der Durchführung aller Test-Pipelines

5.8 Zusammenfassung

Im Rahmen des Kapitels fand die testgetriebene Implementierung der konzipierten Softwarearchitektur statt. Vor der Umsetzung jeder Komponente wurden seine Funktionalitäten, ihre Aufgaben und Schnittstellen, zusammen mit den entsprechenden Testszenarien detailliert und in tabellarischer Form beschrieben. Nach der Umsetzung der Komponenten erfolgte die Implementierung der einzelnen Testklassen. Es wurde gezeigt, wie die strukturorientierten und die zustandsbezogenen Verifikationsprozessen implementiert wurden. Im Rahmen des letzten Unterabschnittes wurde die Inbetriebnahme der Testautomatisierungslösung beschrieben und ein Testprozess für 26 Testobjekte erfolgreich durchgeführt. Als aktives Testobjekt diente eine *SoftPLC* zusammen mit einer generierten VWS. Dadurch konnte bestätigt werden, dass die vorgeschlagene Softwareinfrastruktur praktisch umgesetzt werden kann und eine erste Basis für die automatisierte Verifikation von Anlagenkomponenten darstellt.

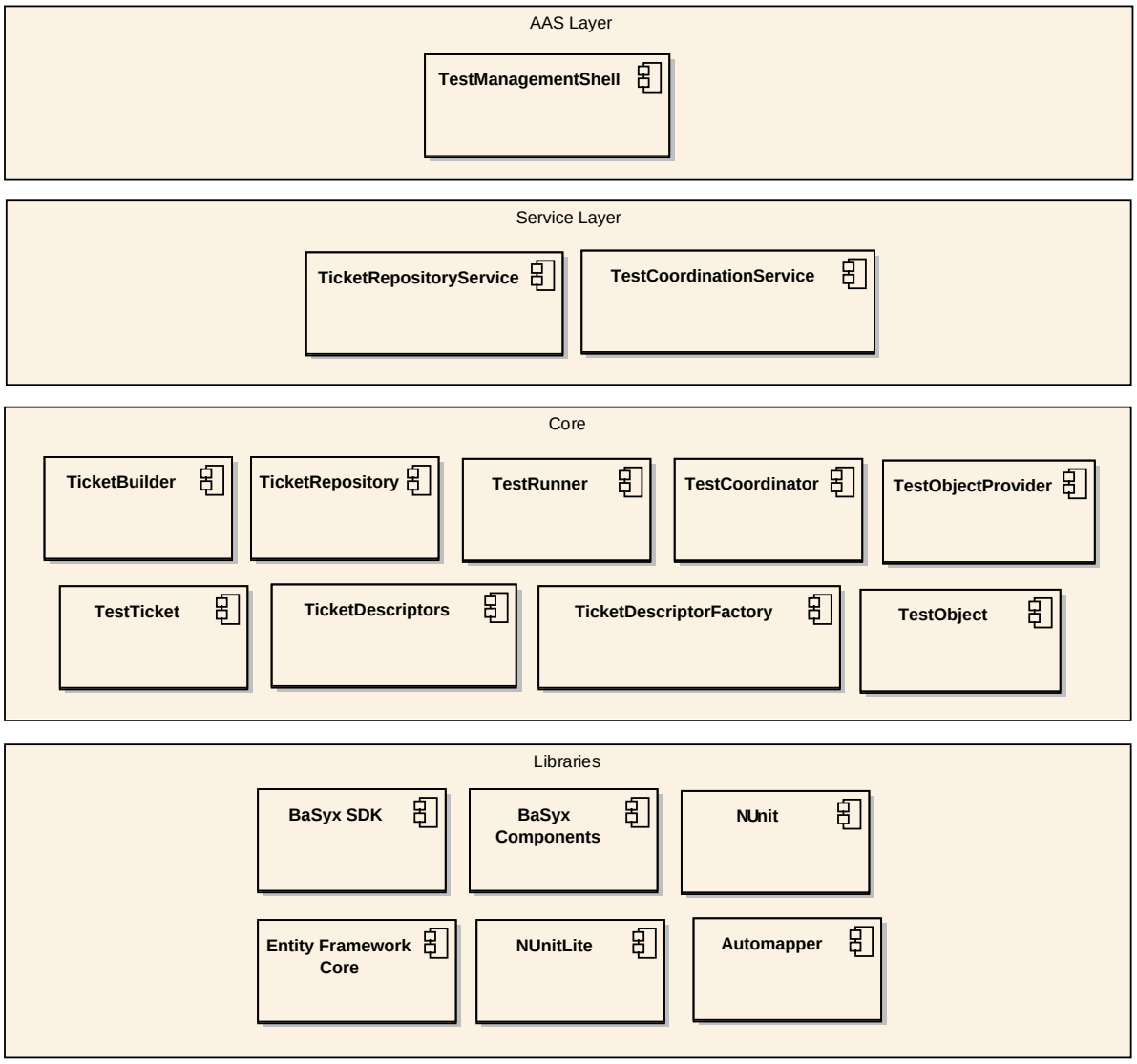


Abbildung 5.25: Softwarearchitektur

Kapitel 6

Fazit

6.1 Zusammenfassung

Im Verlauf dieser Abschlussarbeit fand eine Konzeption und partielle Entwicklung einer flexiblen und generischen Softwarearchitektur zum automatisierten Testen von Anlagenkomponenten durch die Nutzung der Verwaltungsschale statt. Dabei wurden folgende Zielstellungen abgearbeitet:

- Identifikation und detaillierte Beschreibung von zwei Möglichkeiten für die Durchführung eines automatisierten Testprozesses unter Berücksichtigung der Softwareinfrastruktur *Eclipse BaSyx*
- Festlegung der potenziellen Kernfunktionalitäten einer als VWS implementierten Testautomatisierungslösung
- Aufstellung der Hauptanforderungen, welche von einer vollständig implementierten Version der vorgeschlagenen Testlösung zu erfüllen wären
- Modellierung und Beschreibung der notwendigen Kommunikationsinfrastruktur und des prinzipiellen Testdurchführungsprozesses zum Zweck der Identifikation bestimmter Architekturkomponenten
- Feingranulare Synthese der Softwaremodule, welche eine Möglichkeit zur Realisierung der spezifizierten Funktionalitäten darstellen und zum Erfüllen der aufgelisteten Anforderungen führen
- Testgetriebene Entwicklung einer flexiblen und generischen Softwarearchitektur, welche die Realisierung eines anwendungsspezifischen und redefinierbaren Testdurchführungsprozesses auf Basis der VWS ermöglicht
- Implementierung von Testklassen, welche von der Softwarearchitektur für die Durchführung konkreter Testsequenzen verwendet werden, eine Realisierung von Strukturtests ermöglichen und zum Verifizieren von Führungskomponenten dienen

Da die vorgeschlagene Softwarearchitektur durch eine hohe Erweiterbarkeit charakterisiert ist und aufgrund der beschriebenen APIs eine unkomplizierte softwaretechnische Integration weiterer Pipeline-Stufen garantiert, liegt ihr Nutzwert nicht in der Ausführung und Protokollierung der konkreten, zum Zweck der Konzeptprüfung definierten Test-Stufen, sondern in der softwaretechnischen Infrastruktur, welche die generelle Spezifikation von Testebenen, die Festlegung ihrer Sequenz und somit die Durchführung einer anwendungsspezifischen Test-Pipeline anhand bereitgestellter Testrezepte ermöglicht. Eine applikationsspezifische Anpassung und sogar komplette Neudefinition der Testsequenzen ist durch den Einsatz der Schnittstelle der Komponente *Test Orchestration Service* leicht realisierbar, was über eine hohe Abstraktionsebene der konzipierten Testautomatisierungslösung spricht, sein eigentlicher Nutzwert entspricht und die Wiederverwendbarkeit der Testplattform garantiert.

Aufgrund des generischen Metamodells der Verwaltungsschale und der durch dieses angebotenen Möglichkeiten zum Speichern von unterschiedlichen Arten von Daten, lässt sich die VWS als ein sehr gut geeignetes Werkzeug zum Persistieren und Austausch von Modellen, Testrezepten, Testprotokollen und anderen Testartefakten bezeichnen, welche als Bestandteile einer Testautomatisierungslösung und/oder Hilfsmittel für die Durchführbarkeit eines Testprozesses dienen. Ein weiterer bedeutender Vorteil der

Nutzung der VWS als eigenständige oder Bestandteil einer Testautomatisierungsarchitektur besteht in der Möglichkeit zur Laufzeit-Rekonfiguration und Neudefinition des gesamten Testprozesses.

Die in Abschnitt 2.2.2 diskutierte Testautomatisierungsarchitektur (siehe A.1) lässt sich vollständig durch den Einsatz der Verwaltungsschale umsetzen. Im Rahmen der Arbeit konnte gezeigt werden, wie die Testdefinitions-, Testdurchführungs- und durch die BaSys-Plattform zum Teil vorgegebene Testadaptierungsschicht im Kontext einer VWS-orientierten Testautomatisierungsarchitektur implementiert werden können.

6.2 Kritischer Rückblick

Als Ziel dieser Arbeit diente die Konzeption und selektive Implementierung einer flexiblen, wiederverwendbaren, vom konkreten Anwendungsfall unabhängigen und somit für den Einsatz in wandelbaren Produktionsanlagen geeigneten Softwarearchitektur. Die partiell implementierte Lösung stellt eine Basis für das automatisierte Testen von Industrie 4.0 Komponenten dar. Dies erforderte die Durchführung einer entsprechenden Literaturrecherche und die Auseinandersetzung mit einem breiten Spektrum von Ansätzen und technischen Werkzeugen wie z.B. *MoMuT*, *Eclipse Papyrus*, *Eclipse BaSyx*, *NUnit*, *AASX Package Explorer*, *SpecFlow*, *Selenium*, *FMPy*, *FactoryIO*[1], *ExtentReports*, *MS SQL Server*, *XML Schema Definition Tool*, *Entity Framework Core* u.v.a.

Im Rahmen der Konzeptionsphase wurden die allgemeinen Anwendungsszenarien sowie die Kernfunktionalitäten einer beispielhaften Testautomatisierungslösung anhand von Use-Case Diagrammen diskutiert. Darauf basierend wurden zwei Testumgebungen vorgeschlagen, welche die Umsetzbarkeit der definierten Dienste garantieren. Die Beschreibung der Kommunikationsbeziehungen zwischen den einzelnen Komponenten jeder Testumgebung fand durch den Einsatz von Sequenzdiagrammen statt. Im nächsten Schritt geschah die Spezifikation der im Rahmen der Konzeption getroffenen Annahmen und Abhängigkeiten. Darauffolgend wurden bestimmte Anforderungen diskutiert, welche bei einer vollständigen Implementierung der vorgeschlagenen Testinfrastrukturen zu berücksichtigen sind. Im Rahmen der Konzepterstellung wurden weiterhin die APIs und die Abhängigkeiten der primären Softwaremodule diskutiert und die Architekturkomponenten detailliert und auf Basis mehrerer Klassendiagramme beschrieben. Als Ergebnis des Kapitels und signifikantes Softwareartefakt dieser Abschlussarbeit dient die durch Abb. 4.30 dargestellte Beschreibung der synthetisierten Softwarearchitektur zum automatisierten Testen individueller Anlagenkomponenten.

Als Ergebnis der Arbeit gilt die erfolgreiche und testgetriebene Implementierung der Softwarearchitektur einer Testumgebung, durch welche Industrie 4.0 Komponenten auf Basis redefinierbarer Testsequenzen und anhand eines Ticket-Systems verifiziert werden können. Im Rahmen der Implementierung wurden 12 Softwarekomponenten entwickelt. Die Verifikation der Architektur fand auf Basis von 99 tabellarisch beschriebenen Testfällen und 224 ausgeführten Integrations- und Unit-Tests statt. Die Funktionalitäten der einzelnen Module wurden detailliert und nach Zuordnung der entsprechenden Testszenarien in Tabellenform dargelegt.

Der Nachteil der Realisierung eines digitalen Vertreters, der zur Verifikation weiterer Verwaltungsschalen eingesetzt wird, besteht in der hohen Anzahl von Nachrichten, die im Rahmen eines einzigen Testvorgangs über das Netzwerk zwischen der VWS der Testlösung und dem Testobjekt ausgetauscht werden. Obwohl die Vorteile dieses Ansatzes vielfältig sind, lässt es sich behaupten, dass die Verifikation verteilter Systeme durch Koordinatoren i.A. als ineffizient angesehen werden kann. Dieses Problem lässt sich durch die Realisierung einer Testumgebung für reflektive Testobjekte lösen. Dieser Aspekt wird im folgenden Abschnitt als Gegenstand weiterer Untersuchungen definiert.

Obwohl die implementierte Lösung eine anwendungsspezifische Definition der einzelnen Test-Stufen und ihrer Reihenfolge im Kontext der Test-Pipeline ermöglicht, bietet diese keine Möglichkeit zur automatisierten und dynamischen Integration weiterer Deskriptoren als Teil des Test-Rezeptes. Aktuell erfordert

[1] `https://factoryio.com/`

den Einsatz weiterer Deskriptoren ihre softwaretechnische Spezifikation in Form von XML-Dateien, die darauffolgende Implementierung der korrespondierenden Klassen zur Deserialisierung sowie Ergänzungen der Komponenten *Descriptor Factory* und *Ticket Builder*. Eine Lösungsstrategie bietet den Einsatz der Schnittstelle `IDescriptor` an. Eine Abstraktion der Deskriptoren durch diese Schnittstelle würde in Zusammenhang mit dem Konzept der Polymorphie die dynamische Integration weiterer Testbeschreibungen ermöglichen. Die automatisierte Ableitung der Klassen aus den XML-Dateien lässt sich bspw. durch das mehrmals erwähnte *XML Schema Definition Tool* automatisieren.

6.3 Ausblick

Die synthetisierte Softwarearchitektur wurde prototypisch und zum Zweck der Überprüfung der technischen Umsetzbarkeit des Konzeptes implementiert. Während der Bearbeitung der Themenstellung sind mehrere Ideen für weitere Arbeiten entstanden.

Im Rahmen zukünftiger Arbeiten können Möglichkeiten zur Durchführung eines Testprozesses seitens des Testobjekts nach Zuordnung eines Test-Skriptes untersucht werden. Der Vorteil eines solchen Testansatzes besteht in der Zeiteffizienz, welche durch die lokale Durchführung einzelner Teststufen angeboten wird. Hierfür ist die Nutzung oder Festlegung eines Submodells erforderlich, das alle Systemcharakteristiken beschreibt, die zum Zweck der Bereitstellung eines vom Testobjekt ausführbaren Test-Skriptes in Betracht gezogen werden müssen. Zum Zweck der Initiierung des Testprozesses ist die Definition eines weiteren Submodells notwendig. Dieses soll die Operationen zum Ausführen der Test-Skripte bereitstellen. Ein Komponentendiagramm, das die Testumgebung visualisiert, befindet sich unter A.4.

Die Erstellung eines relationalen Abbilds des Metamodells der Verwaltungsschale lässt sich als ein weiteres potenzielles Thema definieren. Dieses würde als Treiber für die Entwicklung komplexer Web-Anwendungen und Microservices zum Koordinieren und Orchestrieren von Ressourcen inklusive Visualisierung aller von den Submodellen repräsentierten Eigenschaften dienen. Ein relationales Datenmodell würde weiterhin als logische Ergänzung der Eclipse BaSyx Plattform gelten und einen Impuls im Bereich der industriellen Webtechnologien darstellen. In Zusammenhang mit der entwickelten Testlösung kann ein solches Modell für die Realisierung einer Web-basierten grafischen Oberfläche auf Basis des MVC-Entwurfsmusters[2], .NET Core und Entity Framework Core eingesetzt werden, durch welche Testprozesse initiiert und Tickets vom menschlichen Endnutzer erstellt, definiert, bearbeitet und verwaltet werden können.

Ein weiteres mögliches Ziel, für welches im Rahmen dieser Abschlussarbeit die Grundlagen gelegt wurden, entspricht der dynamischen Definition der als Teil der Test-Pipeline definierten Testklassen von Anwendern ohne Programmiererfahrung. Hierfür kann eine Semantik in Zusammenhang mit den Gherkin-Editoren und dem Metamodell der Verwaltungsschale spezifiziert werden, welche auf Basis bestimmter, aus Schlüsselwörtern und Identifikationsdaten zusammengesetzter, Anforderungsformulierungsmuster Testszenarien definiert. Zum Zweck einer solchen Entwicklung durch den Einsatz der Softwareumgebung .NET kann das Entwicklungswerkzeug *SpecFlow*[3] in Betracht gezogen werden. Eine tiefgehende Analyse der Softwarekomponenten dieses quelloffenen und BDD-orientierten Testplattforms würde die Extraktion, Anpassung und ggf. Reimplementierung dieser Bestandteile ermöglichen, welche für die Interpretation der Semantik verantwortlich sind. Darauf basierend lässt sich eine Bibliothek aus einfachen und möglichst generischen Test-Routinen implementieren, welche die Interpretation und Überprüfung eines Spektrums aus textuellen, regelgerechten, feingranularen und technologieorientierten Anforderungen ermöglichen.

Andere wichtige Aspekte, die im Rahmen weiterer Ausarbeitungen zu adressieren wären, entsprechen der Durchführung eines Simulationsprozesses zum Zweck der Verifikation des Verhaltens eines Testobjekts und der Realisierung von Komponententests für jede im Rahmen der VWS definierte Operation. Da die Simulation und die virtuelle Inbetriebnahme als sehr umfassende Themen gelten, eignet sich für eine zukünftige Ausarbeitung dieses Themas an erster Stelle eine Auseinandersetzung mit der *Functional Mock-up Interface*[4] (FMI) Spezifikation zur Definition und Bereitstellung von Simulationsmodellen in einer

[2] `https://learn.microsoft.com/de-de/aspnet/core/mvc/overview?view=aspnetcore-6.0`
[3] `https://specflow.org/`
[4] `https://fmi-standard.org/`

standardisierten Form. Zum Zweck der automatisierten Generierung FMI-basierter Simulationsmodelle können Werkzeuge wie *Open Modellica*[5], *Eclipse Papyrus*, *Modelio*[6], *FMPy*[7] u.a. verwendet werden. Eine weitere Möglichkeit bietet die manuelle Erstellung von Simulationseinheiten durch die Implementierung des entsprechenden Simulationsmodells mit der Programmiersprache C an. Auf Basis einer Softwarekomponente, welche die im Rahmen der FMI-Spezifikation definierten Funktionalitäten implementiert und als Simulationsmaster[8] gilt, lassen sich komplexe Simulationseinheiten sowohl einzeln als auch in Verbindung miteinander ausführen. Im Kontext der Industrie 4.0 Vision kann dieser Simulationsmaster als ein Softwaregegenstand betrachtet werden. Eine Abstraktion dieses Assets durch den Einsatz der Verwaltungsschale würde der im Rahmen dieser Arbeit vorgeschlagenen und prototypisch entwickelten Testlösung die Möglichkeit bieten eine virtuelle Inbetriebnahme für das entsprechende Testobjekt als Teil der Test-Pipeline durchzuführen. Wie die Ergebnisse aus der Simulation protokolliert werden sollen, bleibt ein Thema zukünftiger Arbeiten.

[5] `https://openmodelica.org/`
[6] `https://www.modelio.org/`
[7] `https://github.com/CATIA-Systems/FMPy`
[8] Beispielhafter Simulationsmaster in C: `https://github.com/qtronic/fmusdk/blob/master/fmu20/src/co_simulation/main.c`

Abbildungsverzeichnis

2.1 Allgemeine Struktur der Verwaltungsschale [HDM21] [Bad20] 18

3.1 Generierungsumgebung nach [Sch21] 22
3.2 Ablauf nach [Sch21] (vereinfacht) 22
3.3 Engineering-Prozess als Voraussetzung für eine Testlösung nach [Sch22b] 23
3.4 Testlösungsarchitektur nach [Sch22b] 24

4.1 *BaSyx* - Kernkomponenten und ihre Interaktionen 28
4.2 Allgemeine Anwendungsszenarien 29
4.3 Kernfunktionalitäten 31
4.4 Strukturorientierte Testdurchführung - Notwendigkeit und Vorgehensweise 34
4.5 Registry-orientierte Testumgebung [Sch22a] 36
4.6 Registry-orientierte Testdurchführung - Ablauf 38
4.7 Ticket-orientierte Testumgebung 39
4.8 Ticket-orientierte Testduchführung - Ablauf 41
4.9 Szenario 1: Kapselung der Funktionalitäten durch eine Hauptklasse 46
4.10 Szenario 2: Direkte Instanziierung der Kernkomponenten 47
4.11 Szenario 3: Bereitstellung der Dienste durch Service-Klassen 47
4.12 Ticket-Repository Dienste 48
4.13 Test-Orchestrator Dienste 50
4.14 Test-Object Provider Dienste 52
4.15 Test-Runner Dienste 53
4.16 Klassendiagramm Ticket-Deskriptoren 55
4.17 Klassendiagramm des Strukturdeskriptors 57
4.18 Klassendiagramm Descriptor Factory 59
4.19 Klassendiagramm Test-Ticket 60
4.20 Klassendiagramm Ticket-Builder 61
4.21 Klassendiagramm Ticket-Repository 62
4.22 Klassendiagramm Ticket-Repository Service 63
4.23 Klassendiagramm Test Object 64
4.24 Klassendiagramm Test Object Provider 65
4.25 Klassendiagramm Test Runner 66
4.26 Klassendiagramm Test Orchestrator 67
4.27 Klassendiagramm Test Coordination Service 68
4.28 Klassendiagramm Test Management Shell 69
4.29 Test Management Shell - Struktur 70
4.30 Überblick 71

5.1 TDD-Makrozyklus [Gre11] 73
5.2 Softwarekomponenten und Abhängigkeiten 74
5.3 Testergebnisse - *Descriptor Factory* 83
5.4 Testergebnisse - *Test Ticket* 85
5.5 Testergebnisse - *Ticket Builder* 96
5.6 Aktivitätsdiagramm - Instanziierung der Klasse `TestObject` 99
5.7 Testergebnisse - *Test Object* 99
5.8 Eingesetzte Komponenten der *NUnit Test Engine API* 102
5.9 Testergebnisse - *Test Runner* 102

5.10 Testergebnisse - *Test Runner* TC.TR01 103
5.11 Klassendiagramm: *Ticket Database Repository* 110
5.12 Testergebnisse - *Ticket (Database) Repository* 111
5.13 Testergebnisse - *Test Object Provider* 118
5.14 Prinzipielle Funktionsweise der Komponente *Test Orchestrator* 128
5.15 Testergebnisse - *Test Orchestrator* 129
5.16 Testergebnisse - *Ticket Repository Service* 142
5.17 Testergebnisse - *Test Orchestration Service* 147
5.18 Submodellelement `TicketsCount` - Anfragebearbeitung 149
5.19 Submodellelement `UploadTicket` - Anfragebearbeitung 150
5.20 Submodellelement `UploadTicket` - Anfragebearbeitung 152
5.21 Sequenzdiagramm: Strukturverifikation 156
5.22 Traversierung des Zustandsautomaten anhand der Informationen im Zustandsautomatendeskriptor 159
5.23 Inhalt des Ticket-Verzeichnisses nach einem Upload von 26 Testrezepten 161
5.24 Inhalt des Ticket-Verzeichnisses nach der Durchführung aller Test-Pipelines 162
5.25 Softwarearchitektur 163

A.1 Generische Testautomatisierungsarchitektur nach [Bak16] XIX
A.2 Umsetzung von Skills im Kontext der VWS [Bay22] XX
A.3 Ontologie des PPR-Modells [Bay22] XXI
A.4 Testumgebung für reflektive Testobjekte XXII
A.5 Sequenzdiagramm: Initiierung eines reflektiven Testprozesses XXII
A.6 Sequenzdiagramm: Instanziierung der Softwarekomponenten XXIII
A.7 Sequenzdiagramm: Submodellelement `ExecuteTestPipelines` - Anfragebearbeitung . . . XXIV
A.8 Sequenzdiagramm: Allgemeine Verifikation eines Zustandsautomaten XXV
A.9 Festlegung der gültigen Testobjekte XXVI
A.10 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 1) XXVII
A.11 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 2) XXVIII
A.12 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 3) XXIX
A.13 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 4) XXX
A.14 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 5) XXXI
A.15 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 6) XXXII
A.16 Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 7) XXXIII
A.17 Test Demonstrator VWS - Struktur (Aussschnitt Nr. 1) XXXIV
A.18 Test Demonstrator VWS - Struktur (Aussschnitt Nr. 2) XXXV
A.19 Test Demonstrator VWS - Struktur (Aussschnitt Nr. 3) XXXVI
A.20 Test Demonstrator VWS - Struktur (Aussschnitt Nr. 4) XXXVII
A.21 Test Demonstrator VWS - Struktur (Aussschnitt Nr. 5) XXXVIII
A.22 Test Demonstrator VWS - Struktur (Aussschnitt Nr. 6) XXXIX
A.23 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 1) XL
A.24 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 2) XLI
A.25 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 3) XLII
A.26 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 4) XLIII
A.27 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 5) XLIV
A.28 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 6) XLV
A.29 Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 7) XLVI
A.30 VWS-Struktur einer als Testobjekt geltenden Führungskomponente XLVII
A.31 Anzahl von Tickets im Rahmen der Inbetriebnahmephase XLVIII

Tabellenverzeichnis

5.1 Funktionsbeschreibung: *Descriptor Factory* - `CreatePassiveShell` 75
5.2 Funktionsbeschreibung: *Descriptor Factory* - `CreatePassiveShell` (überladen) 75
5.3 Funktionsbeschreibung: *Descriptor Factory* - `CreateStructureDescriptor` 75
5.4 Funktionsbeschreibung: *Descriptor Factory* - `CreateStructureDescriptor` (überladen) . 75
5.5 Funktionsbeschreibung: *Descriptor Factory* - `CreateStateMachineDescriptor` 76
5.6 Funktionsbeschreibung: *Descriptor Factory* - `CreateStateMachineDescriptor` (überladen) 76
5.7 Teststrategie: *Descriptor Factory* - TC.DF01 76
5.8 Teststrategie: *Descriptor Factory* - TC.DF02 77
5.9 Teststrategie: *Descriptor Factory* - TC.DF03 77
5.10 Teststrategie: *Descriptor Factory* - TC.DF04 77
5.11 Teststrategie: *Descriptor Factory* - TC.DF05 78
5.12 Teststrategie: *Descriptor Factory* - TC.DF06 78
5.13 Teststrategie: *Descriptor Factory* - TC.DF07 78
5.14 Teststrategie: *Descriptor Factory* - TC.DF08 79
5.15 Teststrategie: *Descriptor Factory* - TC.DF09 79
5.16 Teststrategie: *Descriptor Factory* - TC.DF10 79
5.17 Teststrategie: *Descriptor Factory* - TC.DF11 80
5.18 Teststrategie: *Descriptor Factory* - TC.DF12 80
5.19 Teststrategie: *Descriptor Factory* - TC.DF13 80
5.20 Teststrategie: *Descriptor Factory* - TC.DF14 81
5.21 Teststrategie: *Descriptor Factory* - TC.DF15 81
5.22 Datenmodell: *TestTicket* 84
5.23 Funktionsbeschreibung: *Test Ticket* - Konstruktor 84
5.24 Teststrategie: *Test Ticket* - TC.TT01 85
5.25 Funktionsbeschreibung: *Ticket Bulder* - `WithId` 86
5.26 Funktionsbeschreibung: *Ticket Bulder* - `WithType` 86
5.27 Funktionsbeschreibung: *Ticket Bulder* - `WithUri` 86
5.28 Funktionsbeschreibung: *Ticket Bulder* - `WithPassiveAdministrationShell` 87
5.29 Funktionsbeschreibung: *Ticket Bulder* - `WithStructureDescriptor` 87
5.30 Funktionsbeschreibung: *Ticket Bulder* - `WithStateMachineDescriptor` 87
5.31 Funktionsbeschreibung: *Ticket Bulder* - `Build` 87
5.32 Teststrategie: *Ticket Builder* - TC.TB01 88
5.33 Teststrategie: *Ticket Builder* - TC.TB02 89
5.34 Teststrategie: *Ticket Builder* - TC.TB03 90
5.35 Teststrategie: *Ticket Builder* - TC.TB04 90
5.36 Teststrategie: *Ticket Builder* - TC.TB05 91
5.37 Teststrategie: *Ticket Builder* - TC.TB06 91
5.38 Teststrategie: *Ticket Builder* - TC.TB07 92
5.39 Teststrategie: *Ticket Builder* - TC.TB08 92
5.40 Teststrategie: *Ticket Builder* - TC.TB09 93
5.41 Teststrategie: *Ticket Builder* - TC.TB10 93
5.42 Teststrategie: *Ticket Builder* - TC.TB11 94
5.43 Teststrategie: *Ticket Builder* - TC.TB12 94
5.44 Teststrategie: *Ticket Builder* - TC.TB13 95
5.45 Datenmodell: *Test Object* 97
5.46 Funktionsbeschreibung: *Test Object* - Konstruktor 97

5.47 Teststrategie: *Test Object* - TC.TO01 98
5.48 Teststrategie: *Test Object* - TC.TO02 98
5.49 Teststrategie: *Test Object* - TC.TO03 98
5.50 Funktionsbeschreibung: *Test Runner* - `RunTests` 100
5.51 Teststrategie: *Test Runner* - TC.TR01 100
5.52 Teststrategie: *Test Runner* - TC.TR02 101
5.53 Teststrategie: *Test Runner* - TC.TR03 101
5.54 Funktionsbeschreibung: *Ticket Repository* - `Upload` 103
5.55 Funktionsbeschreibung: *Ticket Repository* - `Delete` 103
5.56 Funktionsbeschreibung: *Ticket Repository* - `Update` 104
5.57 Funktionsbeschreibung: *Ticket Repository* - `DeleteTicketById` 104
5.58 Funktionsbeschreibung: *Ticket Repository* - `RetrieveTicketById` 104
5.59 Funktionsbeschreibung: *Ticket Repository* - `RetrieveTicketByUrl` 104
5.60 Funktionsbeschreibung: *Ticket Repository* - `RetrieveAllTickets` 105
5.61 Teststrategie: *Ticket Repository* - TC.RE01 105
5.62 Teststrategie: *Ticket Repository* - TC.RE02 105
5.63 Teststrategie: *Ticket Repository* - TC.RE03 106
5.64 Teststrategie: *Ticket Repository* - TC.RE04 106
5.65 Teststrategie: *Ticket Repository* - TC.RE05 106
5.66 Teststrategie: *Ticket Repository* - TC.RE06 107
5.67 Teststrategie: *Ticket Repository* - TC.RE07 107
5.68 Teststrategie: *Ticket Repository* - TC.RE08 107
5.69 Teststrategie: *Ticket Repository* - TC.RE09 108
5.70 Teststrategie: *Ticket Repository* - TC.RE10 108
5.71 Teststrategie: *Ticket Repository* - TC.RE11 108
5.72 Teststrategie: *Ticket Repository* - TC.RE12 109
5.73 Funktionsbeschreibung: *Test Object Provider* - Konstruktor 111
5.74 Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObject` 112
5.75 Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObject` 112
5.76 Funktionsbeschreibung: *Test Object Provider* - `RetrievePassiveTestObjects` 112
5.77 Funktionsbeschreibung: *Test Object Provider* - `RetrieveActiveTestObjects` 112
5.78 Funktionsbeschreibung: *Test Object Provider* - `RetrieveProcessedTestObjects` 113
5.79 Funktionsbeschreibung: *Test Object Provider* - `RetrieveUnprocessedTestObjects` 113
5.80 Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObjectById` 113
5.81 Funktionsbeschreibung: *Test Object Provider* - `RetrieveTestObjectByUrl` 113
5.82 Teststrategie: *Test Object Provider* - TC.TOP01 114
5.83 Teststrategie: *Test Object Provider* - TC.TOP02 114
5.84 Teststrategie: *Test Object Provider* - TC.TOP03 115
5.85 Teststrategie: *Test Object Provider* - TC.TOP04 115
5.86 Teststrategie: *Test Object Provider* - TC.TOP05 115
5.87 Teststrategie: *Test Object Provider* - TC.TOP06 116
5.88 Teststrategie: *Test Object Provider* - TC.TOP07 116
5.89 Teststrategie: *Test Object Provider* - TC.TOP08 116
5.90 Teststrategie: *Test Object Provider* - TC.TOP09 117
5.91 Teststrategie: *Test Object Provider* - TC.TOP10 117
5.92 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipeline` 119
5.93 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipelineById` 119
5.94 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipelineByUri` 120
5.95 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteTestPipelines` 120
5.96 Funktionsbeschreibung: *Test Orchestrator* - `ExecutePassiveTestPipelines` 120
5.97 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteActiveTestPipelines` 121
5.98 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteProcessedTestPipelines` 121
5.99 Funktionsbeschreibung: *Test Orchestrator* - `ExecuteUnprocessedTestPipelines` 122
5.100 Teststrategie: *Test Orchestrator* - TC.OR01 122

5.101 Teststrategie: *Test Orchestrator* - TC.OR02 . . . 123
5.102 Teststrategie: *Test Orchestrator* - TC.OR03 . . . 123
5.103 Teststrategie: *Test Orchestrator* - TC.OR04 . . . 124
5.104 Teststrategie: *Test Orchestrator* - TC.OR05 . . . 124
5.105 Teststrategie: *Test Orchestrator* - TC.OR06 . . . 125
5.106 Teststrategie: *Test Orchestrator* - TC.OR07 . . . 125
5.107 Teststrategie: *Test Orchestrator* - TC.OR08 . . . 126
5.108 Teststrategie: *Test Orchestrator* - TC.OR09 . . . 126
5.109 Teststrategie: *Test Orchestrator* - TC.OR10 . . . 127
5.110 Teststrategie: *Test Orchestrator* - TC.OR11 . . . 127
5.111 Funktionsbeschreibung: *Ticket Repository Service* - `UploadActiveTicket` . . . 130
5.112 Funktionsbeschreibung: *Ticket Repository Service* - `UploadPassiveTicket` . . . 130
5.113 Funktionsbeschreibung: *Ticket Repository Service* - `UpdateActiveTicket` . . . 130
5.114 Funktionsbeschreibung: *Ticket Repository Service* - `UpdatePassiveTicket` . . . 131
5.115 Funktionsbeschreibung: *Ticket Repository Service* - `DeleteTicket` . . . 131
5.116 Funktionsbeschreibung: *Ticket Repository Service* - `RetrieveTicketById` . . . 131
5.117 Funktionsbeschreibung: *Ticket Repository Service* - `RetrieveTicketByUri` . . . 132
5.118 Funktionsbeschreibung: *Ticket Repository Service* - `RetrieveTicketIds` . . . 132
5.119 Teststrategie: *Ticket Repository Service* - TC.TRS01 . . . 132
5.120 Teststrategie: *Ticket Repository Service* - TC.TRS02 . . . 133
5.121 Teststrategie: *Ticket Repository Service* - TC.TRS03 . . . 133
5.122 Teststrategie: *Ticket Repository Service* - TC.TRS04 . . . 133
5.123 Teststrategie: *Ticket Repository Service* - TC.TRS05 . . . 134
5.124 Teststrategie: *Ticket Repository Service* - TC.TRS06 . . . 134
5.125 Teststrategie: *Ticket Repository Service* - TC.TRS07 . . . 134
5.126 Teststrategie: *Ticket Repository Service* - TC.TRS08 . . . 135
5.127 Teststrategie: *Ticket Repository Service* - TC.TRS09 . . . 135
5.128 Teststrategie: *Ticket Repository Service* - TC.TRS10 . . . 135
5.129 Teststrategie: *Ticket Repository Service* - TC.TRS11 . . . 136
5.130 Teststrategie: *Ticket Repository Service* - TC.TRS12 . . . 136
5.131 Teststrategie: *Ticket Repository Service* - TC.TRS13 . . . 136
5.132 Teststrategie: *Ticket Repository Service* - TC.TRS14 . . . 137
5.133 Teststrategie: *Ticket Repository Service* - TC.TRS15 . . . 137
5.134 Teststrategie: *Ticket Repository Service* - TC.TRS16 . . . 137
5.135 Teststrategie: *Ticket Repository Service* - TC.TRS17 . . . 138
5.136 Teststrategie: *Ticket Repository Service* - TC.TRS18 . . . 138
5.137 Teststrategie: *Ticket Repository Service* - TC.TRS19 . . . 138
5.138 Teststrategie: *Ticket Repository Service* - TC.TRS20 . . . 139
5.139 Teststrategie: *Ticket Repository Service* - TC.TRS21 . . . 139
5.140 Teststrategie: *Ticket Repository Service* - TC.TRS22 . . . 139
5.141 Teststrategie: *Ticket Repository Service* - TC.TRS23 . . . 140
5.142 Teststrategie: *Ticket Repository Service* - TC.TRS24 . . . 140
5.143 Teststrategie: *Ticket Repository Service* - TC.TRS25 . . . 140
5.144 Teststrategie: *Ticket Repository Service* - TC.TRS26 . . . 141
5.145 Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteTestPipelineById` . . . 142
5.146 Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteTestPipelineByUri` . . . 143
5.147 Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteActiveTestPipelines` . . . 143
5.148 Funktionsbeschreibung: *Test Orchestration Service* - `ExecutePassiveTestPipelines` . . . 143
5.149 Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteProcessedTestPipelines` . . . 144
5.150 Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteUnprocessedTestPipelines` 144
5.151 Funktionsbeschreibung: *Test Orchestration Service* - `ExecuteTestPipelines` . . . 144
5.152 Teststrategie: *Test Orchestration Service* - TC.TOS01 . . . 145
5.153 Teststrategie: *Test Orchestration Service* - TC.TOS02 . . . 145
5.154 Teststrategie: *Test Orchestration Service* - TC.TOS03 . . . 145

5.155 Teststrategie: *Test Orchestration Service* - TC.TOS04 146
5.156 Teststrategie: *Test Orchestration Service* - TC.TOS05 146
5.157 Teststrategie: *Test Orchestration Service* - TC.TOS06 146
5.158 Submodellelement: *Ticket Management SM* - `TicketsCount` 148
5.159 Submodellelement: *Ticket Management SM* - `PassiveTicketsCount` 148
5.160 Submodellelement: *Ticket Management SM* - `ActiveTicketsCount` 148
5.161 Submodellelement: *Ticket Management SM* - `ProcessedTicketsCount` 148
5.162 Submodellelement: *Ticket Management SM* - `UnprocessedTicketsCount` 149
5.163 Submodellelement: *Ticket Management SM* - `UploadActiveTicket` 150
5.164 Submodellelement: *Ticket Management SM* - `UploadPassiveTicket` 150
5.165 Submodellelement: *Ticket Management SM* - `UpdateActiveTicket` 151
5.166 Submodellelement: *Ticket Management SM* - `UpdatePassiveTicket` 151
5.167 Submodellelement: *Ticket Management SM* - `DeleteTicket` 152
5.168 Submodellelement: *Ticket Management SM* - `RetrieveTicketById` 153
5.169 Submodellelement: *Ticket Management SM* - `RetrieveTicketByUri` 153
5.170 Submodellelement: *Ticket Management SM* - `RetrieveTicketIds` 153
5.171 Submodellelement: *Test Execution SM* - `ExecuteTestPipelineById` 154
5.172 Submodellelement: *Test Execution SM* - `ExecuteTestPipelineById` 154
5.173 Submodellelement: *Test Execution SM* - `ExecuteActiveTestPipelines` 154
5.174 Submodellelement: *Test Execution SM* - `ExecutePassiveTestPipelines` 154
5.175 Submodellelement: *Test Execution SM* - `ExecuteTestPipelines` 155

Source Code Content

4.1 JSON-Objekt einer Typ 1 VWS 56
4.2 XML-Struktur einer Typ 1 VWS 56
4.3 Beispiel für die Spezifikation zustandsorientierter Testfälle 58

5.1 Instanziierung aktiver Test-Tickets 95
5.2 Allgemeine Struktur eines Zustandsautomatendeskriptors 157
5.3 Allgemeine Struktur einer Aktion im Kontext des Zustandsautomatendeskriptors 157
5.4 Allgemeine Struktur eines Pfades im Kontext des Zustandsautomatendeskriptors 158
5.5 Instanziierung der implementierten Softwarekomponenten 160
5.6 Allgemeine Struktur eines Zustandsautomatendeskriptors 161

A.1 Vorlage für die Erstellung von Test-Klassen XLIX
A.2 Beispielhafte Verhaltensbeschreibung als einfache Basis für die Durchführung von Komponententests L
A.3 XML-Schema für Zustandsautomatendeskriptoren LI
A.4 *PackML*-basierter Zustandsautomatendeskriptor LIII
A.5 Beispielhafter Protokoll als Ergebnis einer Strukturtestung LX
A.6 Beispielhafter Protokoll als Ergebnis einer zustandsbezogenen Testung LXXXIX

Glossar

Abstraction "*A view of an object that focuses on the information relevant to a particular purpose and ignores the remainder of the information.* " [IEE90].

Adaptation Parameter "*A variable that is given a specific value to adapt a program to a given installation site or to given conditions in its operational environment.* " [IEE90].

Asset "*Entity which is owned by or under the custodial duties of an organization, having either a perceived or actual value to the organization.*
Anmerkung 1: Jede Verwaltungsschale kann ein oder mehrere Teilmodelle enthalten. " [Bos22].

Aufzeichnung "*Dokument, das erreichte Ergebnisse angibt oder einen Nachweis ausgeführter Tätigkeiten bereitstellt.* " [Sta15].

Builder Design Pattern A design pattern, which is used to separete "*the construction of a complex object from its representation so that the same construction process can create different representations.* " [Gam94].

Capability Checking "*A formal procedure to assess the fulfillment of a required capability against the provided capabilities of a resource* " [Bay22].

Dependency Injection "*Dependency Injection is a set of software design principles and patterns that enable us to develop loosely coupled code..* " [See11].

Fähigkeit "*The implementation-independent description of the function of a resource to achieve a certain effect in the physical or virtual world.* " [Bay22].

Feasibility Checking "*A formal procedure to assess the possibility to achieve the desired effect of a skill execution in a concrete context* " [Bay22].

Funktionsblock "*A stateful functional element in a system that interacts with other elements via input and output variables and performs a computation or an action and thus one way of realizing a skill* " [Bay22].

Konformität "*Erfüllung einer Anforderung* " [Sta15].

Objektiver Nachweis "*Daten, welche die Existenz oder Wahrheit von etwas bestätigen. Objektive Nachweise können durch Beobachtung, Messung, Test oder anderen Mitteln erbracht werden.* " [Sta15].

Ontologie "*A formal, explicit specification of a shared conceptualization* " [Bos22].

Process "*Entirety of procedures in a system by means of which the material, energy or information is transformed, transported or stored* " [Bos22].

Product "*The intermediate or final merchandise that is created as a result of a process step* " [Bos22].

Prüfung "*Bestimmung der Konformität mit festgelegten Anforderungen.* " [Sta15].

Ressource "*An asset that is used in a process to perform the procedures* " [Bos22].

Skill "*The asset-dependent implementation of the function of a resource to achieve a certain effect in the physical or virtual world* " [Bay22].

Test "*Bestimmung entsprechend den Anforderungen an einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung.* " [Sta15].

Überprüfung "*Bestimmung der Eignung, Angemessenheit und Wirksamkeit eines Objekts, festgelegte Ziele zu erreichen.* " [Sta15].

Validierung "*Bestätigung durch Bereitstellung eines objektiven Nachweises, dass die Anforderungen für einen spezifischen beabsichtigten Gebrauch oder eine spezifische beabsichtigte Anwendung erfüllt worden sind.*
Anmerkung zum Begriff:
Der für eine Validierung erforderliche objektive Nachweis ist das Ergebnis eines Tests oder einer anderen Form der Bestimmung, z. B. Durchführen alternativer Berechnungen oder Überprüfen von Dokumenten. " [Sta15].

Verifizierung "*Bestätigung durch Bereitstellung eines objektiven Nachweises, dass festgelegte Anforderungen erfüllt worden sind.*
Anmerkung 1 zum Begriff: Der für eine Verifizierung erforderliche objektive Nachweis kann das Ergebnis einer Prüfung oder anderer Formen der Bestimmung sein, z. B. Durchführen alternativer Berechnungen oder Überprüfen von Dokumenten. " [Sta15].

Verwaltungsschale "*digitale Repräsentation des Assets* " [Bos22].

Wandlungsfähigkeit "*Eigenschaft eines Systems, es mit geringem Arbeitsaufwand an Änderungen, die zum Planungszeitpunkt nicht bekannt waren, anzupassen.*
Anmerkung: Änderungen im Kontext von Produktionssystemen können z. B. neue Produktvarianten, veränderte Normen sein. " [WZ09] [Bos22].

Abkürzungsverzeichnis

3A Arrange-Act-Assert.

AAS Asset Administration Shell.

API Application Programming Interface.

AS Automatisierungssystem.

BDD Behavior-Driven Development.

BMBF Bundesministerium für Bildung und Forschung.

CPPS Cyber Physical Production Systems.

CPS Cyber Physical Systems.

ECU Electronic Control Unit.

EVA Eingabe-Verarbeitung-Ausgabe.

FA Fachausschuss.

FAAAST Fraunhofer Advanced Asset Administration Shell Tools.

FMI Functional Mock-up Interface.

GMA Gesellschaft Mess- und Automatisierungstechnik.

GUID Globally Unique Identifier.

HIL Hardware-in-the-Loop.

HTTP Hypertext Transfer Protocol.

I4.0 Industrie 4.0.

IDTA Industriel Digital Twin Association.

IEC International Electrotechnical Commission.

IFAK Institute for Automation and Communication.

ISTQB International Software Testing Qualifications Board.

JSON JavaScript Object Notation.

MBT Modellbasiertes Testen.

MIL Model-in-the-Loop.

MVC Model-View-Controller.

MVP Minimum Viable Product.

OMAC Organization for Machine Automation and Control.

OPC UA Open Platform Communications Unified Architecture.

ORM Object-Relational Mapping.

PackML Packaging Machine Language.

PPR Product, Process, Resource.

RAMI 4.0 Referenzarchitekturmodell Industrie 4.0.

REST Representational state transfer.

SDK Software Development Kit.

SIL Software-in-the-Loop.

SOLID Single responsibility, Open–closed, Liskov substitution, Interface segregation, Dependency inversion.

SPS Speicherprogrammierbare Steuerung.

SQL Structured Query Language.

SUT Software under Test.

TAA Test Automation Architecture.

TAS Test Automation Solution.

TDD Test-Driven Development.

UI User Interface.

UML Unified Modeling Language.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VDE Verband der Elektrotechnik, Elektronik und Informationstechnik.

VDI Verein Deutscher Ingenieure.

VIBN Virtuelle Inbetriebnahme.

VWS Verwaltungsschale.

XML Extensible Markup Language.

Literaturverzeichnis

[4018] Plattform Industrie 4.0. „The Structure of the Administration Shell: Trilateral Perspectives from France, Italy and Germany“. In: (März 2018). URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.pdf?__blob=publicationFile&v=4 (besucht am 04. 05. 2022).

[Aru19] A. K. Arumugam. „Software Testing Techniques & New Trends“. In: *International Journal of Engineering Research and Technology (IJERT)* V8.12 (2019). DOI: 10.17577/IJERTV8IS120318. URL: https://www.researchgate.net/publication/342116266_Software_Testing_Techniques_New_Trends (besucht am 14. 04. 2022).

[Bad20] Sebastian Bader, Erich Barnstedt, Heinz Bedenbender, Bernd Berres, Meik Billmann, Birgit Boss, Andre Braunmandl, Erich Clauer, Christian Diedrich, Björn Flubacher, Wolfgang Fritsche, Kai Garrels, Andreas Gatterburg, Martin Hankel, Oliver Hillermeier, Michael Hoffmeister, Michael Jochem, Alexander Köpke, Yevgen Kogan und Constantin Ziesche. „Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0 (Version 3.0RC01)“. In: (Nov. 2020).

[Bar15] Mike Barth, Philipp Puntel Schmidt, Mario Hoernicke, Mathias Oppelt, Gerrit Wolf, Lorenz Hundt und Oliver Stern. „Methoden und Modelle der Virtuellen Inbetriebnahme - Eine Übersicht der Richtlinienarbeit des GMA FA 6.11“. In: Juni 2015.

[Bau21] M. Baumgartner, S. Gwihs, R. Seidl, T. Steirer und M.F. Wendland. *Basiswissen Testautomatisierung: Aus- und Weiterbildung zum ISTQB® Advanced Level Specialist – Certified Test Automation Engineer*. dpunkt.verlag, 2021. ISBN: 9783960887874. URL: https://books.google.de/books?id=mAcYEAAAQBAJ.

[Bay22] Andreas Bayha, Jürgen Bock, Birgit Boss, Christian Diedrich und Somayeh Malakuti. „Describing Capabilities of Industrie 4.0 Components. Joint White Paper between Plattform Industrie 4.0, VDI GMA 7.20, BaSys 4.2“. In: (2022).

[Bec02] Kent Beck. *Test Driven Development. By Example (Addison-Wesley Signature)*. Addison-Wesley Longman, Amsterdam, 2002. ISBN: 0321146530.

[BGW18] Uwe Bracht, Dieter Geckler und Sigrid Wenzel. *Digitale Fabrik: Methoden und Praxisbeispiele*. Springer, 2018. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-662-55783-9.pdf.

[Bor17] Vaishali Borse. „Seven Fundamental Principles of Testing“. In: *International Journal of Science and Research (IJSR)* 6.12 (2017), S. 1–3. DOI: 10.21275/ART20178462.

[Bos22] B. Boss, J. Grothoff, M. Lieske, D. Schel, D. Stock, C. Bauer und H. et. al. Bedenbender. „Industrie 4.0 Begriffe / Terms and definitions“. In: *VDI status report Industrie 4.0* (Juni 2022), S. 36.

[Bou19] N. Boulila. „Cyber-Physical Systems and Industry 4.0: Properties, Structure, Communication, and Behavior“. In: (Apr. 2019). DOI: 10.13140/RG.2.2.27890.76485. URL: https://www.researchgate.net/publication/332420221_Cyber-Physical_Systems_and_Industry_40_Properties_Structure_Communication_and_Behavior (besucht am 04. 05. 2022).

[Chr06] G. Chryssolouris. *Manufacturing Systems: Theory and Practice*. SpringerLink Bücher. New York, NY: Springer New York, 2006. ISBN: 0387256830. DOI: 10.1007/0-387-28431-1.

[Cla13] J. Clark. „Dependency Injection: A Practical Introduction. An overview of the dependency injection pattern by JeremyBytes.com“. In: (2013). URL: http://www.jeremybytes.com/downloads/dependencyinjection.pdf (besucht am 15. 04. 2022).

[Dij70] Edsger W. Dijkstra. „Notes on Structured Programming“. 1970. URL: http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF (besucht am 14. 04. 2022).

[DW16] D. Dr. Gorecky und S. Weyer. „Systemarchitektur für Industrie 4.0-Produktionsanlagen“. In: (2016).

[Gam94] Erich Gamma, Richard Helm, Ralph Johnson und John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. Aufl. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1.

[Glo18] Thomas Glock, Björn Sillman, Max Kobold, Sebatian Rebmann und Eric Sax. „Model-based validation and testing of industry 4.0 plants“. In: *2018 Annual IEEE International Systems Conference (SysCon)*. 2018, S. 1–8. DOI: 10.1109/SYSCON.2018.8369563.

[Gre11] J.W. Grenning. *Test Driven Development for Embedded C*. Pragmatic Bookshelf Series. Pragmatic Bookshelf, 2011. ISBN: 9781934356623. URL: https://books.google.de/books?id=QuUBRQAACAAJ.

[Grü13] Stephan Grünfelder. *Software-Test für Embedded Systems: Ein Praxishandbuch für Entwickler, Tester und technische Projektleiter*. 1. Aufl. Heidelberg: dpunkt.verl., 2013. ISBN: 3864900484.

[GTD21] Jonas Gampig, Tarik Terzimehić und Kirill Dorofeev. „PLC Integration into Industry 4.0 Middleware: Function Block Library for the Interaction with REST and OPC UA Asset Administration Shells“. In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2021, S. 1–8. DOI: 10.1109/ETFA45728.2021.9613267.

[Ham11] Reinhard Hametner, Benjamin Kormann, Birgit Vogel-Heuser, Dietmar Winkler und Alois Zoitl. „Test case generation approach for industrial automation systems“. In: *The 5th International Conference on Automation, Robotics and Applications*. 2011, S. 57–62. DOI: 10.1109/ICARA.2011.6144856.

[HDM21] Yining Huang, Saadia Dhouib und Jacques Malenfant. „An AAS Modeling Tool for Capability-Based Engineering of Flexible Production Lines“. In: *IECON 2021 - 47th Annual Conference of the IEEE Industrial Electronics Society*. 2021, S. 1–6. DOI: 10.1109/IECON48115.2021.9589329.

[HF06] Tanvir Hussain und Georg Frey. „UML-based Development Process for IEC 61499 with Automatic Test-case Generation“. In: *2006 IEEE Conference on Emerging Technologies and Factory Automation*. 2006, S. 1277–1284. DOI: 10.1109/ETFA.2006.355407.

[HK19] M. Hanefi und U. Köse. „Planning Activities in Software Testing Process: A Literature Review and Suggestions for Future Research“. In: (2019). URL: https://www.researchgate.net/publication/331519510_Planning_Activities_in_Software_Testing_Process_A_Literature_Review_and_Suggestions_for_Future_Research (besucht am 14. 04. 2022).

[IEE90] IEEE. „IEEE Standard Glossary of Software Engineering Terminology“. In: *IEEE Std 610.12-1990* (1990), S. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[III17] III. „ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary“. In: *ISO/IEC/IEEE 24765:2017(E)* (2017), S. 1–541. DOI: 10.1109/IEEESTD.2017.8016712.

[III22] III. „ISO/IEC/IEEE 29119-1:2022(en) Software and systems engineering — Software testing — Part 1: General concepts“. In: (2022). URL: https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29119:-1:ed-2:v1:en (besucht am 16. 04. 2022).

[Jor13] Paul C. Jorgensen. *Software Testing: a Craftsman's Approach*. Fourth. Auerbach Publications, 2013.

[Kam20] Achim Kampker, Saskia Wessel, Nicolas Lutz, Michaela Reibetanz und Martin Hehl. „Virtual Commissioning for Scalable Production Systems in the Automotive Industry: Model for evaluating benefit and effort of virtual commissioning“. In: *2020 9th International Conference on Industrial Technology and Management (ICITM)*. 2020, S. 107–111. DOI: 10.1109/ICITM48982.2020.9080348.

[KB10] Dimitry Korotkiy und Klaus Bender. „Universelle Architektur zur Testautomatisierung“. In: *atp edition* 5 (Mai 2010), S. 26–31. DOI: 10.17560/atp.v52i05.395. URL: https://ojs.di-verlag.de/index.php/atp_edition/article/download/2005/2298 (besucht am 16. 04. 2022).

[Kho20] Vladimir Khorikov. *Unit testing: Principles, practices, and patterns*. Shelter Island NY: Manning, 2020. ISBN: 1617296279.

[Kle19] Stephan Kleuker. *Qualitätssicherung durch Softwaretests: Vorgehensweisen und Werkzeuge zum Testen von Java-Programmen*. 2., erweiterte und aktualisierte Auflage. Springer eBook Collection. Wiesbaden: Springer Vieweg, 2019. ISBN: 978-3-658-24885-7. DOI: 10.1007/978-3-658-24886-4.

[Kre15] Willibald Krenn, Rupert Schlick, Stefan Tiran, Bernhard Aichernig, Elisabeth Jobstl und Harald Brandl. „MoMut::UML Model-Based Mutation Testing for UML“. In: *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. 2015, S. 1–8. DOI: 10.1109/ICST.2015.7102627.

[KSZ19] T. Kuhn, F. Schnicke und C. Ziesche. „Tutorial, Teil 1: BaSys und Eclipse BaSyx - Middleware für die wandelbare Produktion. In Bewegung“. In: *iX* 11 (2019), S. 56–63.

[KT18] Itir Karac und Burak Turhan. „What Do We (Really) Know about Test-Driven Development?“ In: *IEEE Software* 35 (Juli 2018), S. 81–85. DOI: 10.1109/MS.2018.2801554.

[Lig09] Peter Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. 2. Aufl. Heidelberg: Spektrum, 2009. ISBN: 978-3-8274-2056-5. DOI: 10.1007/978-3-8274-2203-3.

[Löc21] Andreas Löcklin, Christoph Kotsch, Kai Krüning, Markus Rentschler, Christof Ebert, Manuel Müller und Michael Weyrich. „Testen 4.0 in der Automatisierungstechnik: Agiles modellbasiertes Testen vernetzter Systeme und Komponenten“. In: *Automation 2021, VDI Verlag GmbH Düsseldorf, pp. 335-352, 29. und 30 June 2021* (2021). ISSN: 0083-5560. DOI: 10.51202/9783181023921-335. URL: https://doi.org/10.51202/9783181023921-335.

[Mal10] Qaisar A. Malik, Antti Jääskeläinen, Heikki Virtanen, Mika Katara, Fredrik Abbors, Dragos Truscan und Johan Lilius. „Model-Based Testing Using System vs. Test Models - What Is the Difference?“ In: *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*. 2010, S. 291–299. DOI: 10.1109/ECBS.2010.41.

[MRK16] Stephan Magnus, Tim Russ und Jan Krause. „Anforderungs- und modellbasierte Testfallgenerierung und Testdurchführung unter Nutzung von Methoden zur Netzwerkanalyse“. In: Mai 2016.

[MS19] Carsten Matysczok und Marco Seewaldt. „VDI Fachausschuss Digitale Fabrik - Darstellung der aktuellen Aktivitäten und Richtlinien“. In: *Simulation in Produktion und Logistik 2019*. Hrsg. von Matthias Putz und Andreas Schlegel. Auerbach: Verlag Wissenschaftliche Scripten, 2019, S. 39–48. ISBN: 978-3-95735-114-2. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-662-55783-9.pdf.

[Mul10] T. Muller, A. Beer, M. Klonk und V. Rahul. „Certified Tester Foundation Level Syllabus“. In: (2010). URL: https://www.german-testing-board.info/wp-content/uploads/2016/08/ISTQB_FL_Syll_2010.pdf (besucht am 14. 04. 2022).

[Osh13] R. Osherove. *The Art of Unit Testing: with examples in C#*. Manning, 2013. ISBN: 9781617290893. URL: https://books.google.de/books?id=2GRRmgEACAAJ.

[OU14] Mathias Oppelt und Leon Urbas. „Integrated virtual commissioning an essential activity in the automation engineering process: From virtual commissioning to simulation supported engineering“. In: *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*. 2014, S. 2564–2570. DOI: 10.1109/IECON.2014.7048867.

[Per19] Alexander Perzylo, Julian Grothoff, Levi Lucio, Michael Weser, Somayeh Malakuti, Pierre Venet, Vincent Aravantinos und Torben Deppe. „Capability-based semantic interoperability of manufacturing resources: A BaSys 4.0 perspective“. In: *IFAC-PapersOnLine* 52.13 (2019). 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019, S. 1590–1596. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.11.427. URL: https://www.sciencedirect.com/science/article/pii/S2405896319314089.

[PF06] Seno Panjaitan und Georg Frey. „Combination of UML Modeling and the IEC 61499 Function Block Concept for the Development of Distributed Automation Systems“. In: *2006 IEEE Conference on Emerging Technologies and Factory Automation*. 2006, S. 766–773. DOI: 10.1109/ETFA.2006.355405.

[PSB13] Julius Pfrommer, Miriam Schleipen und Jürgen Beyerer. „Fähigkeiten adaptiver Produktionsanlagen“. In: *atp Edition* 55 (Nov. 2013), S. 42–49. DOI: 10.17560/atp.v55i11.273.

[Rie20] O. Riedel, A. Verl, K. Kübler, F. Jaensch, R. Richter, F. Nöhring und T. Borggräfe. „Leitfaden Virtuelle Inbetriebnahme. Handlungsempfehlungen zum wirtschaftlichen Einstieg“. In: (2020). URL: https://www.vdma.org/c/document_library/get_file?uuid=66118d8d-68fc-55a7-a25b-a90ffe89f1f2&groupId=34570 (besucht am 24. 04. 2022).

[Ros20] Roland Rosen, Jens Jaekel, Mike Barth, Oliver Stern, Ronald Schmidt-Vollus, Till Heinzerling, Peter Hoffmann, Christoph Richter, Philipp Puntel Schmidt, Christian Scheifele, Karl Kübler und Marcus Röhler. „Simulation und digitaler Zwilling im Anlagenlebenszyklus“. In: Feb. 2020.

[Sch09] Michael Schenk, Hrsg. *Industrielle Dienstleistungen und Internationalisierung: One-Stop Services als erfolgreiches Konzept*. 1. Aufl. Wiesbaden: Gabler, 2009. ISBN: 9783834913593.

[Sch21] Stephan Schäfer, Dirk Schöttke, Thomas Kämpfe, Kiril Ralinovski, Bernd Tauber und Ralf Lehmann. „Design and Deployment of Digital Twins for Programmable Logic Controllers in Existing Plants“. In: *Proceedings of the 2nd International Conference on Innovative Intelligent Industrial Production and Logistics, IN4PL 2021, October 25-27, 2021*. Hrsg. von Hervé Panetto, Marco Macchi und Kurosh Madani. SCITEPRESS, 2021, S. 145–150. DOI: 10.5220/0010711000003062. URL: https://doi.org/10.5220/0010711000003062.

[Sch22a] S. Schäfer, D. Schöttke, T. Kämpfe, V. Denkov und B. Tauber. „Verwaltungsschale: Einsatz in wandelbaren Produktionsanlagen“. In: *atp magazin 8/2022: Flexibilität und Effizienz in der Prozessindustrie*. Aug. 2022.

[Sch22b] S. Schäfer, D. Schöttke, T. Kämpfe, V. Denkov und B. Tauber. „Verwaltungsschalen zum anlagenweiten Testen wandelbarer Produktionsanlagen“. In: Jan. 2022, S. 59–74. ISBN: 9783181023990. DOI: 10.51202/9783181023990-59.

[Sch22c] Stephan Schäfer, Dirk Schöttke, Thomas Kämpfe und Oliver Lachmann. „Industriesteuerungen und Verwaltungsschalen, Voraussetzungen zur Synchronisation von Anlagensegmenten“. In: 2022.

[See11] M. Seemann. *Dependency Injection In .NET*. Wiley India Pvt. Limited, 2011. ISBN: 9789350042373. URL: https://books.google.de/books?id=4tk0qAAACAAJ.

[SIG17] VDA QMC Working Group 13 / Automotive SIG. „Automotive SPICE Process Assessment / Reference Model“. In: (2017). URL: https://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf (besucht am 14. 04. 2022).

[SK12] R. Singh und A.K. Khan. „An Approach for Integration Testing in Online Retail Applications“. In: *International Journal of Computer Science and Information Technology* 4.3 (2012), S. 141–158. ISSN: 09754660. DOI: 10.5121/ijcsit.2012.4312. URL: https://arxiv.org/ftp/arxiv/papers/1207/1207.2718.pdf (besucht am 16. 04. 2022).

[Sta15] International Organization for Standardization. „Qualitätsmanagementsysteme – Grundlagen und Begriffe (ISO 9000:2015); Deutsche und Englische Fassung EN ISO 9000:2015“. In: (2015).

[UA19] A. Uddin und A. Anand. „Importance of Software Testing in the Process of Software Development“. In: (2019). URL: https://www.researchgate.net/publication/331223692_Importance_of_Software_Testing_in_the_Process_of_Software_Development (besucht am 14. 04. 2022).

[UL07] Utting und Legeard. *Practical Model-Based Testing: A Tools Approach*. 1. Aufl. Morgan Kaufmann, 2007. ISBN: 0123725011. URL: http://www.amazon.com/gp/redirect.html%3FASIN=0123725011%26tag=ws%26lcode=xm2%26cID=2025%26ccmID=165953%26location=/Practical-Model-Based-Testing-Tools-Approach/dp/0123725011%253FSubscriptionId=13CT5CVB80YFWJEPWS02.

[V14] o. V. „Softwaretest“. In: (2014). URL: https://wwwlehre.dhbw-stuttgart.de/~hoyer/ePapers/Wikipedia/SWE1-SoftwareTest_Wikipedia.pdf (besucht am 14. 04. 2022).

[Wes20] Michael Weser, Jürgen Bock, Siwara Schmitt, Alexander Perzylo und Kathrin Evers. „An Ontology-based Metamodel for Capability Descriptions“. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Bd. 1. 2020, S. 1679–1686. DOI: 10.1109/ETFA46521.2020.9212104.

[Wey15] M. Weyrich. „Herausforderung Test verteilter Systeme - Wie Industrie 4.0 das Testen verändert“. In: Jan. 2015.

[Wey18] Michael Weyrich, Andreas Zeller, Benjamin Maschler, Tobias Jung, Jan Krause, Kai Krüning, Christian Köllner und Daniel Siegrist. „Testen vernetzter Systeme für die Industrie 4.0“. In: *atp edition* 60 (Apr. 2018), S. 26. DOI: 10.17560/atp.v58i03.2343.

[Wie07] H.-P. Wiendahl, H. A. ElMaraghy, P. Nyhuis, M. F. Zäh, H.-H. Wiendahl, N. Duffie und M. Brieke. „Changeable Manufacturing - Classification, Design and Operation“. In: *CIRP Annals* 56.2 (2007), S. 783–809. ISSN: 00078506. DOI: 10.1016/j.cirp.2007.10.003.

[Wit18] F. Witte. *Metriken für das Testreporting*. Jan. 2018. ISBN: 978-3-658-19844-2. DOI: 10.1007/978-3-658-19845-9.

[Wit19] F. Witte. *Testmanagement und Softwaretest: Theoretische Grundlagen und praktische Umsetzung*. Jan. 2019. ISBN: 978-3-658-25086-7. DOI: 10.1007/978-3-658-25087-4.

[WZ09] Engelbert Westkämper und Erich Zahn. *Wandlungsfähige Produktionsunternehmen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. ISBN: 978-3-540-21889-0. DOI: 10.1007/978-3-540-68890-7.

Onlinereferenzen

[4018] Plattform Industrie 4.0. „The Structure of the Administration Shell: Trilateral Perspectives from France, Italy and Germany“. In: (März 2018). URL: https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.pdf?__blob=publicationFile&v=4 (besucht am 04. 05. 2022).

[Aru19] A. K. Arumugam. „Software Testing Techniques & New Trends“. In: *International Journal of Engineering Research and Technology (IJERT)* V8.12 (2019). DOI: 10.17577/IJERTV8IS120318. URL: https://www.researchgate.net/publication/342116266_Software_Testing_Techniques_New_Trends (besucht am 14. 04. 2022).

[Bak16] B. Bakker, G. Bath, A. Born, M. Fewster, J. Haukinen, J. McKay, A. Pollner, R. Popescu und I. Schieferdecker. *Certified Tester Advanced Level Syllabus Test Automation Engineer*. 2016. (Besucht am 16. 04. 2022).

[Boa16] International Software Testing Qualifications Board. *ISTQB Glossary*. 2016. URL: https://glossary.istqb.org/en/search/ (besucht am 16. 04. 2022).

[Bou19] N. Boulila. „Cyber-Physical Systems and Industry 4.0: Properties, Structure, Communication, and Behavior“. In: (Apr. 2019). DOI: 10.13140/RG.2.2.27890.76485. URL: https://www.researchgate.net/publication/332420221_Cyber-Physical_Systems_and_Industry_40_Properties_Structure_Communication_and_Behavior (besucht am 04. 05. 2022).

[Cla13] J. Clark. „Dependency Injection: A Practical Introduction. An overview of the dependency injection pattern by JeremyBytes.com“. In: (2013). URL: http://www.jeremybytes.com/downloads/dependencyinjection.pdf (besucht am 15. 04. 2022).

[Dij70] Edsger W. Dijkstra. „Notes on Structured Programming“. 1970. URL: http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF (besucht am 14. 04. 2022).

[Fou22] Eclipse Foundation. 2022. URL: https://wiki.eclipse.org/BaSyx (besucht am 04. 05. 2022).

[HK19] M. Hanefi und U. Köse. „Planning Activities in Software Testing Process: A Literature Review and Suggestions for Future Research“. In: (2019). URL: https://www.researchgate.net/publication/331519510_Planning_Activities_in_Software_Testing_Process_A_Literature_Review_and_Suggestions_for_Future_Research (besucht am 14. 04. 2022).

[IES22] Fraunhofer IESE. *BaSys 4.0, The Middleware for Industrie 4.0*. 2022. URL: https://www.iese.fraunhofer.de/content/dam/iese/dokumente/innovationsthemen/basys_40-en-fraunhofer_iese.pdf (besucht am 04. 05. 2022).

[KB10] Dimitry Korotkiy und Klaus Bender. „Universelle Architektur zur Testautomatisierung“. In: *atp edition* 5 (Mai 2010), S. 26–31. DOI: 10.17560/atp.v52i05.395. URL: https://ojs.di-verlag.de/index.php/atp_edition/article/download/2005/2298 (besucht am 16. 04. 2022).

[Mar22] Marco Sprenger, Prof. Dr. Martin Ruskoswki, Magnus Volkmann und Jörn Peschke. *Skill based Production – Wie sehen zukünftig Produktionsstrukturen aus: Podcast der Technologie-Initiative SmartFactory KL e.V.* 2022. URL: https://smartfactory.de/podcast/skill-based-production-die-produktionsarchitektur-der-zukunft/ (besucht am 14. 04. 2022).

[Mul10] T. Muller, A. Beer, M. Klonk und V. Rahul. „Certified Tester Foundation Level Syllabus“. In: (2010). URL: https://www.german-testing-board.info/wp-content/uploads/2016/08/ISTQB_FL_Syll_2010.pdf (besucht am 14. 04. 2022).

[Raz18] Razorcat Development GmbH. *TESSY 4.1 UserManual*. 2018. URL: https://www.razorcat.com/files/files/tessy/legacy/TESSY_4.1.x/4.1.23/TESSY_UserManual_41.pdf (besucht am 14. 04. 2022).

[Rie20] O. Riedel, A. Verl, K. Kübler, F. Jaensch, R. Richter, F. Nöhring und T. Borggräfe. „Leitfaden Virtuelle Inbetriebnahme. Handlungsempfehlungen zum wirtschaftlichen Einstieg“. In: (2020). URL: https://www.vdma.org/c/document_library/get_file?uuid=66118d8d-68fc-55a7-a25b-a90ffe89f1f2&groupId=34570 (besucht am 24. 04. 2022).

[SIG17] VDA QMC Working Group 13 / Automotive SIG. „Automotive SPICE Process Assessment / Reference Model“. In: (2017). URL: https://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf (besucht am 14. 04. 2022).

[V14] o. V. „Softwaretest“. In: (2014). URL: https://wwwlehre.dhbw-stuttgart.de/~hoyer/ePapers/Wikipedia/SWE1-SoftwareTest_Wikipedia.pdf (besucht am 14. 04. 2022).

[WZ16] M. Weyrich und A. Zeller. *Testen von Industrie 4.0 Systemen. Wie vernetzte Systeme und Industrie 4.0 das Verständnis von Systemtest und Qualitätssicherung ändern*. 2016. URL: https://www.ias.uni-stuttgart.de/dokumente/vortraege/2016-01-26_Industrie40_Duesseldorf_v12final.pdf (besucht am 16. 04. 2022).

[Zie22] C. Ziesche. *Industrie neu gedacht - ein Tech-Podcast von Bosch Rexroth: Was leistet die Verwaltungsschale?* 2022. URL: https://industrie.podigee.io/10-verwaltungsschale (besucht am 14. 04. 2022).

Anhang A

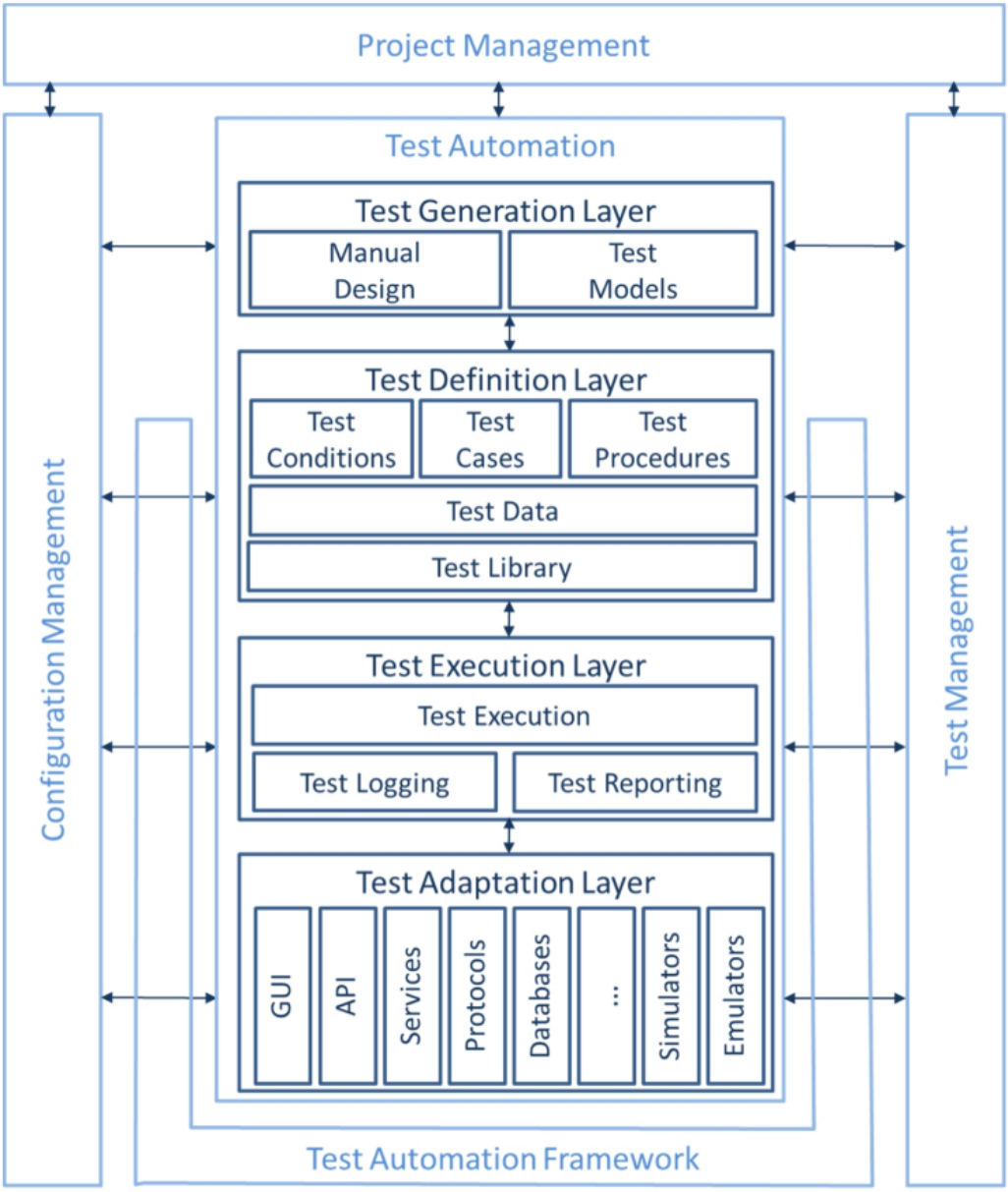


Abbildung A.1: Generische Testautomatisierungsarchitektur nach [Bak16]

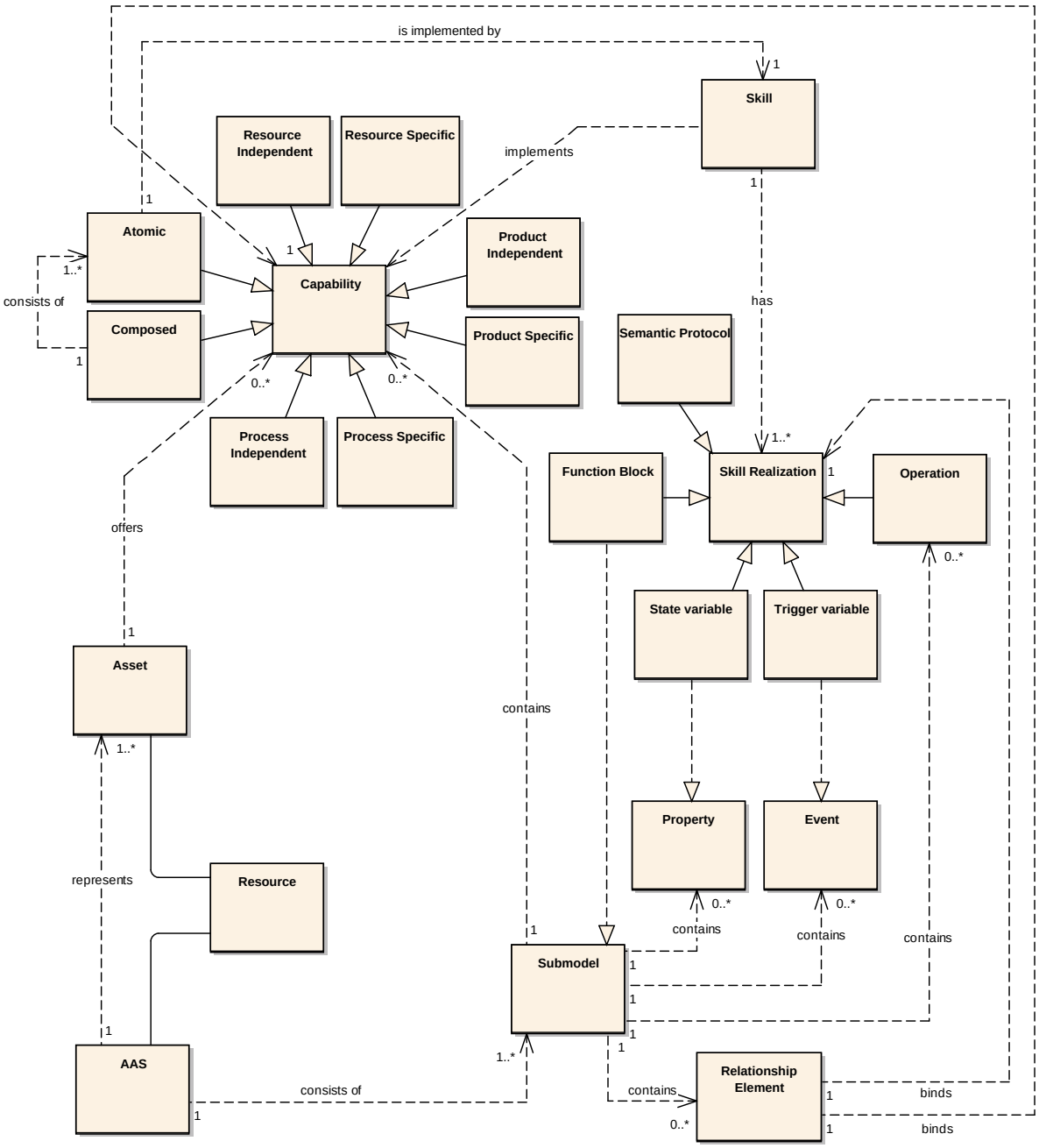


Abbildung A.2: Umsetzung von Skills im Kontext der VWS [Bay22]

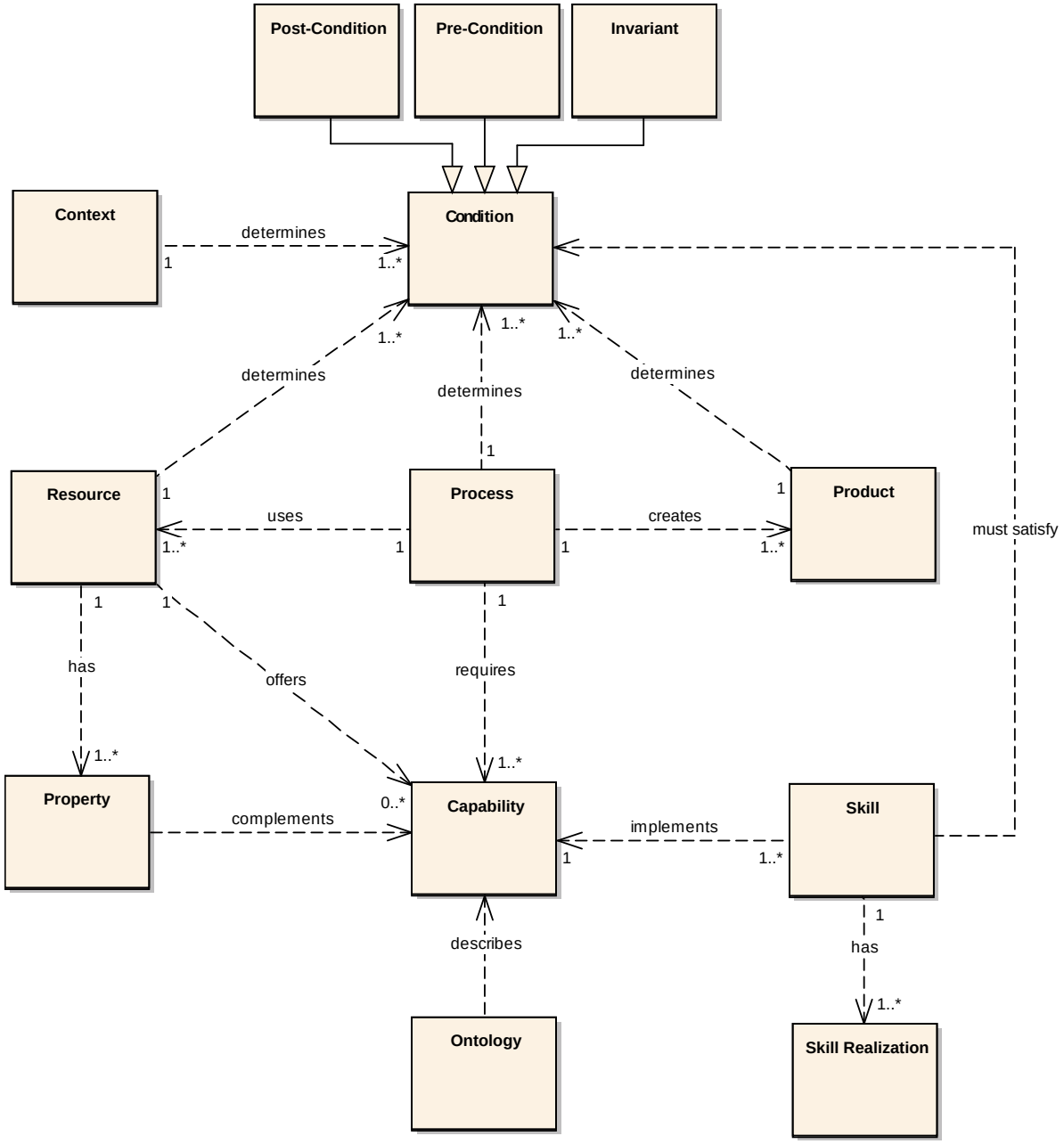


Abbildung A.3: Ontologie des PPR-Modells [Bay22]

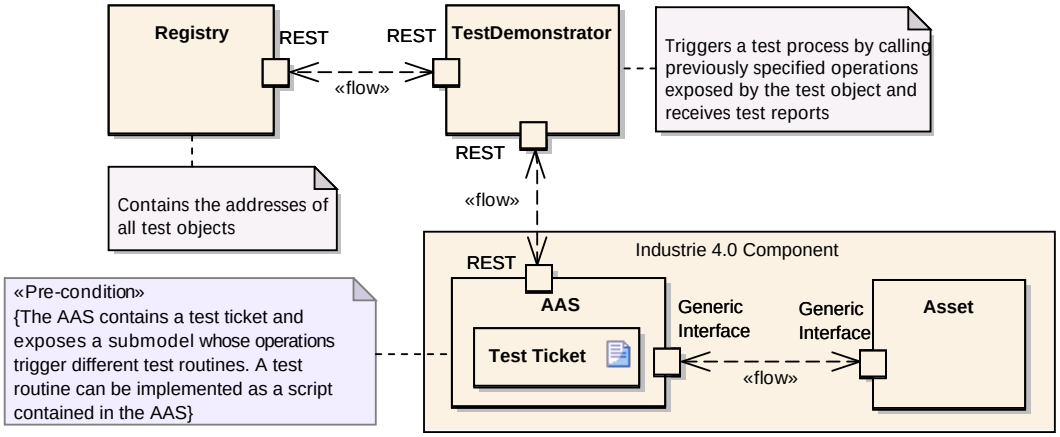


Abbildung A.4: Testumgebung für reflektive Testobjekte

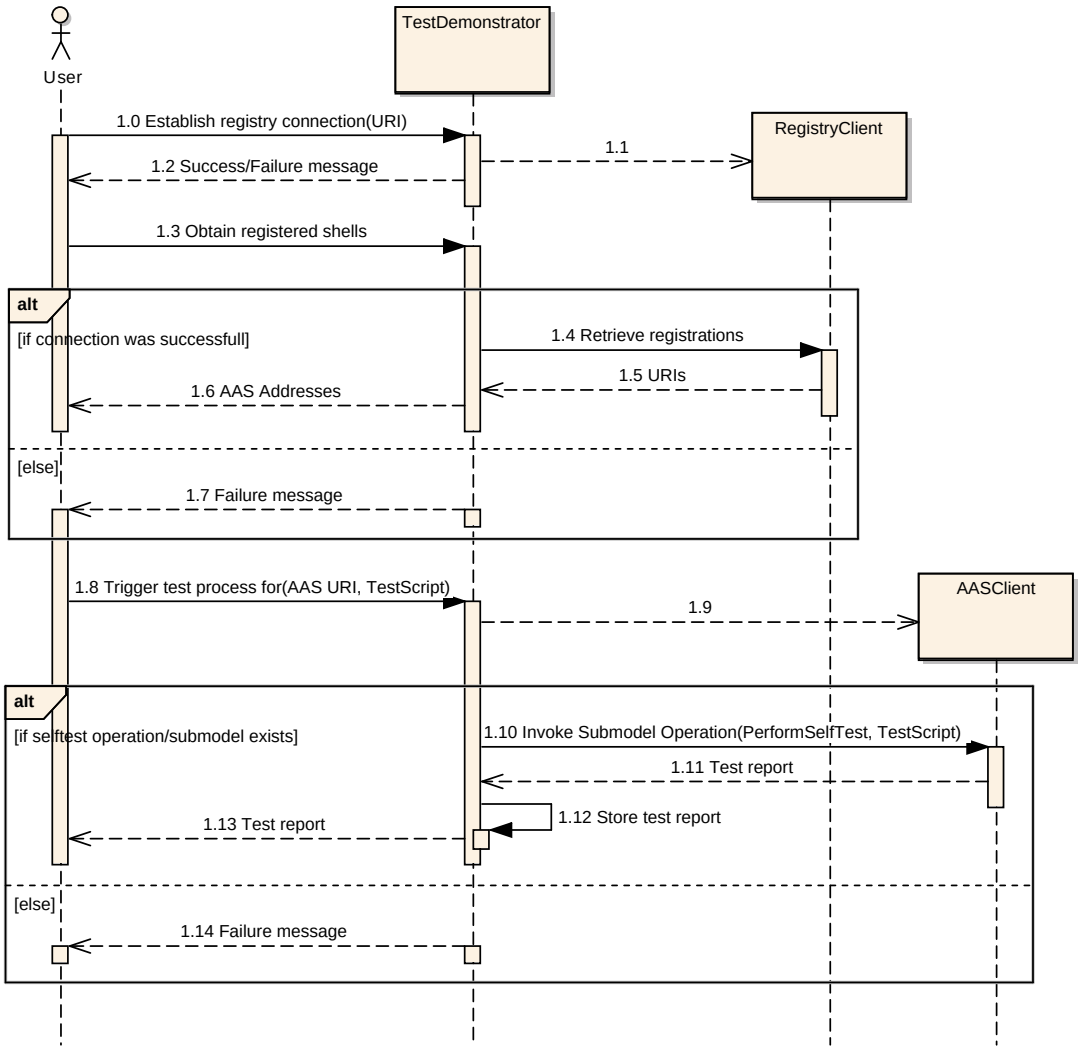


Abbildung A.5: Sequenzdiagramm: Initiierung eines reflektiven Testprozesses

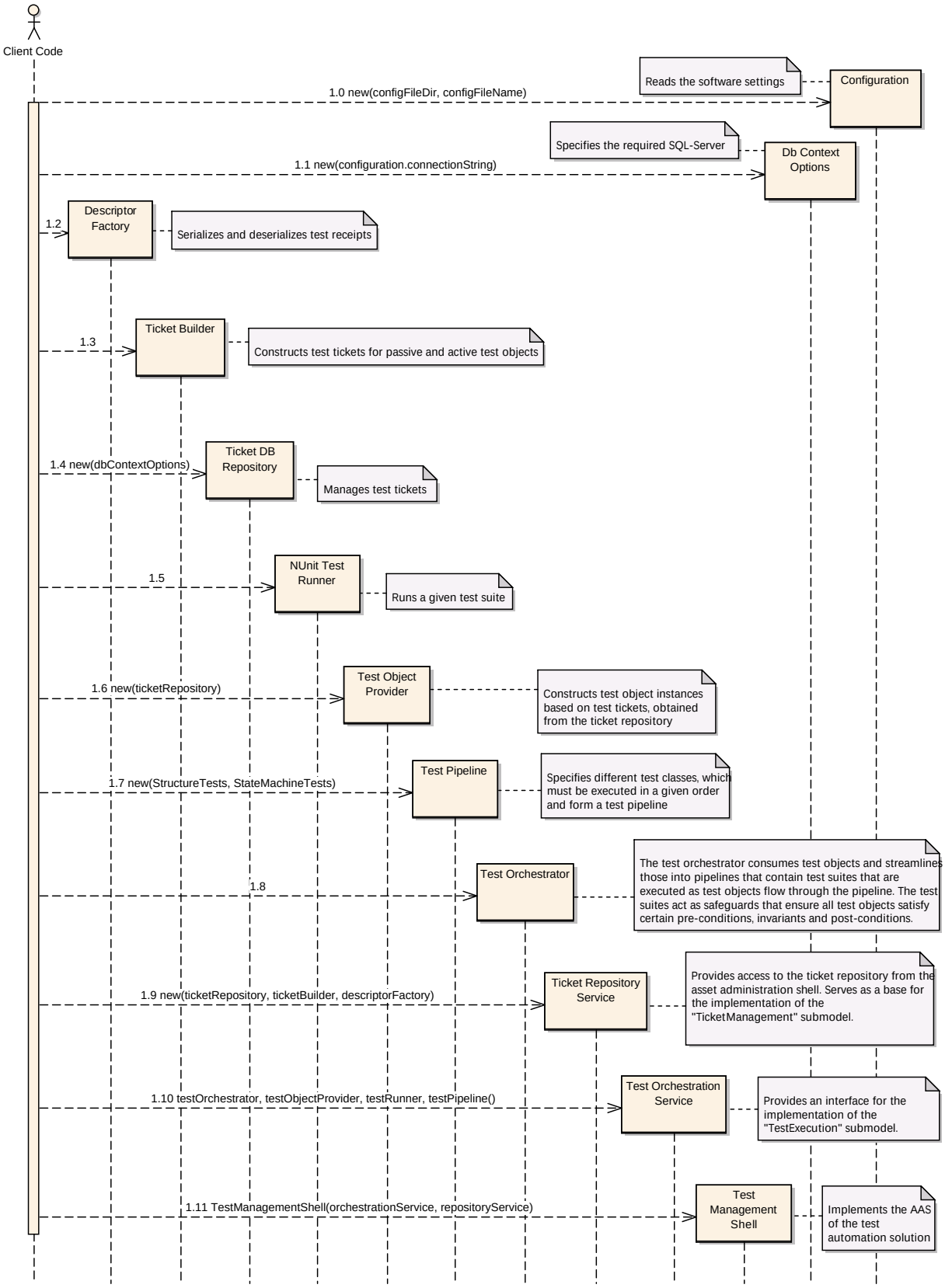


Abbildung A.6: Sequenzdiagramm: Instanziierung der Softwarekomponenten

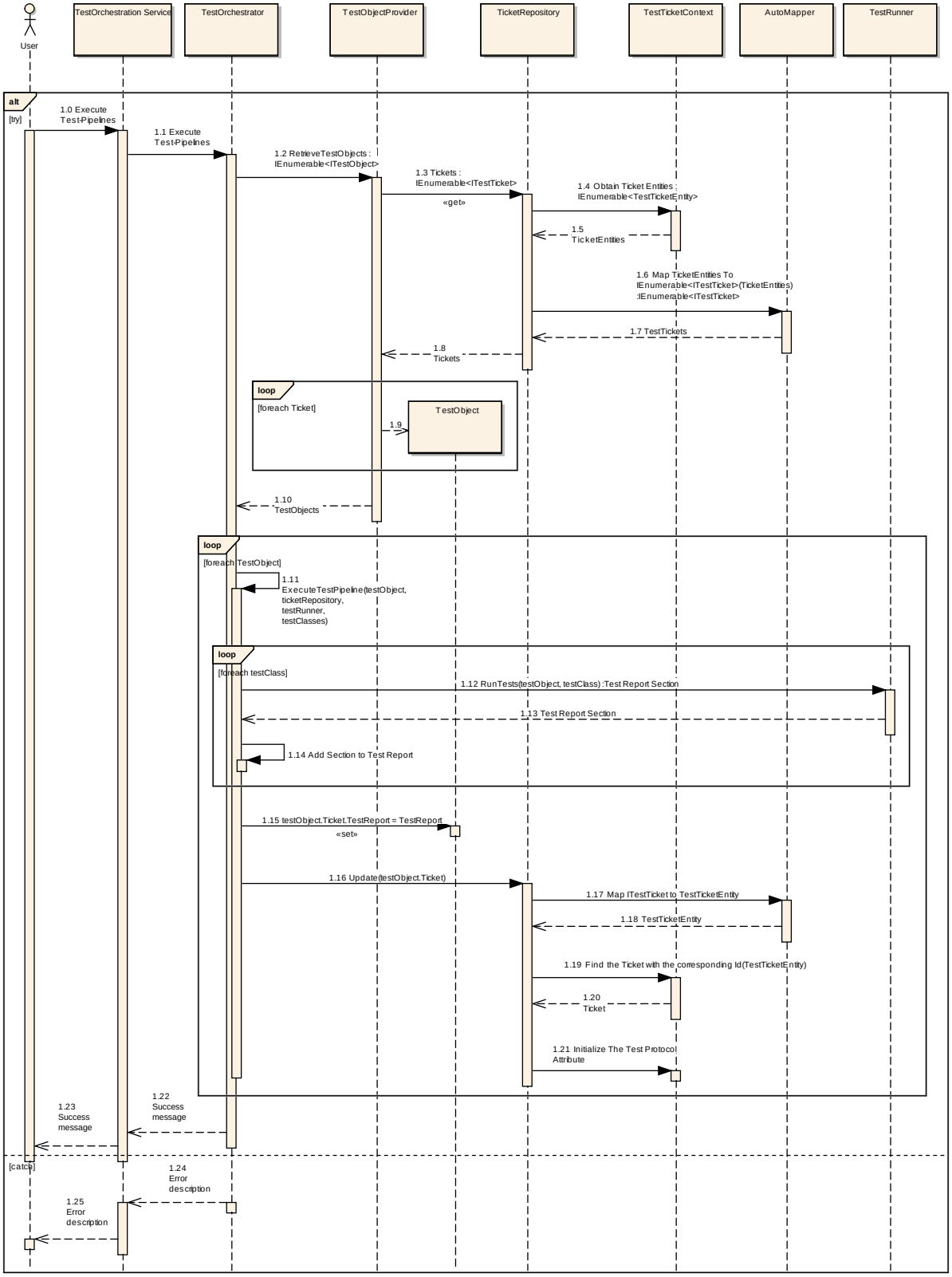


Abbildung A.7: Sequenzdiagramm: Submodellelement `ExecuteTestPipelines` - Anfragebearbeitung

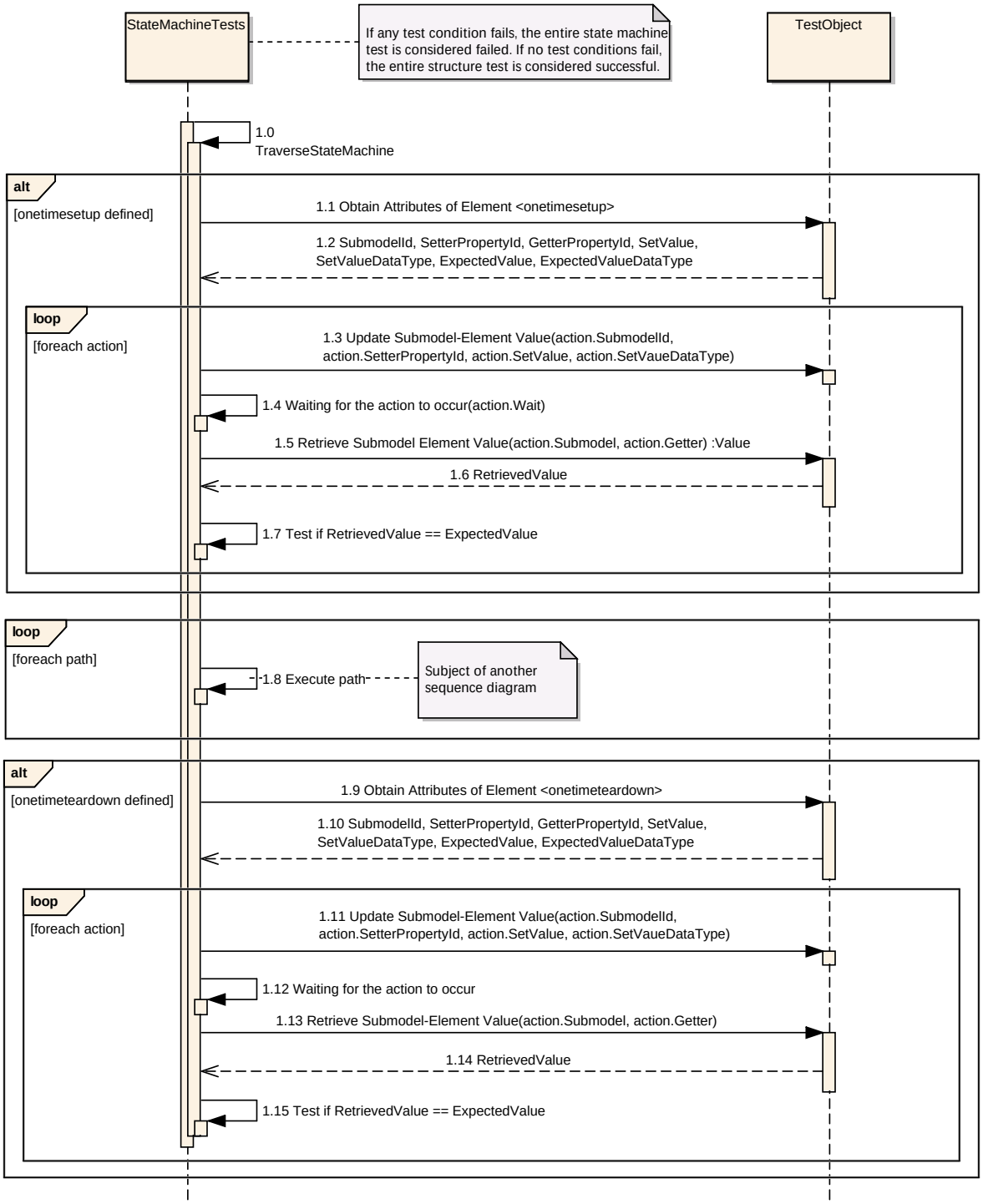


Abbildung A.8: Sequenzdiagramm: Allgemeine Verifikation eines Zustandsautomaten

```
AdministrationShell_ShouldBeAcceptedFromSolution (37)
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/01_Demo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/02_WorkspaceExtension_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/03_BucklingArmRobot_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/04_Sorting_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/05_SupplyUnit_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/01_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/02_Bosch_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/03_Bosch_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/04_Bosch_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/05_Bosch_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/06_Bosch_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/07_PhoenixContact_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/08_SchneiderElectric_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/09_SchneiderElectric_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/10_SchneiderElectric_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/11_SchneiderElectric_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/12_Pepperl+Fuchs_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/13_DKE_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/14_Siemens_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/15_Siemens_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/16_Lenze_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/17_ABB_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/18_Hitachi_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/19_Balluff_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/20_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/21_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/22_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/23_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/24_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/25_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/26_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/27_Festo_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/30_Wittenstein_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/31_SICK_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/32_SICK_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/33_SICK_AAS.xml")
  AdministrationShell_ShouldBeAcceptedFromSolution("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/34_Festo_AAS.xml")
```

Abbildung A.9: Festlegung der gültigen Testobjekte

TestDemonstrator.Tests (186)	47.3 sec
TestDemonstrator.Tests.DescriptorFactoryTests (66)	2.7 sec
DescriptorFactoryTests (66)	2.7 sec
CreatePassiveShell_WithDeserializableContent_ReturnsSameContentAfterDeserialization("../../../Descriptor...	608 ms
CreatePassiveShell_WithSerializableInstance_ReturnsSameContentAfterSerialization("../../../DescriptorFacto...	21 ms
CreateStateMachineDescriptor_WithDeserializableContent_ReturnsSameContentAfterDeserialization("../../.....	47 ms
CreateStateMachineDescriptor_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryT...	15 ms
CreateStateMachineDescriptor_WithSerializableInstance_ReturnsSameContentAfterSerialization("../../../Des...	13 ms
CreateStructureDescriptor_WithDeserializableContent_ReturnsSameContentAfterDeserialization("../../../Des...	20 ms
CreateStructureDescriptor_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/...	7 ms
CreateStructureDescriptor_WithSerializableInstance_ReturnsSameConentAfterSerialization("../../../Descript...	14 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler (30)	1.4 sec
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	17 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	80 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	37 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	71 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	37 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	63 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	58 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	38 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	40 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	41 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	25 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	29 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	31 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	26 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	31 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	32 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	33 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	31 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	29 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	96 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	80 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	77 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	58 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	66 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	81 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	68 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	85 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	23 ms

Abbildung A.10: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 1)

CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	23 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	19 ms
CreatePassiveShell_WithDeserializableContent_ReturnsValidHandler("../../../DescriptorFactoryTests/TestD...	19 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException (7)	197 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	23 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	18 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	34 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	25 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	29 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	34 ms
CreatePassiveShell_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests/TestDa...	34 ms
CreatePassiveShell_WithUnserializableInstance_ThrowsException (2)	< 1 ms
CreatePassiveShell_WithUnserializableInstance_ThrowsException(BaSyx.Models.Core.AssetAdministratio...	< 1 ms
CreatePassiveShell_WithUnserializableInstance_ThrowsException(null)	< 1 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException (8)	158 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	19 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	21 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	17 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	17 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	21 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	23 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	21 ms
CreateStateMachineDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryT...	19 ms
CreateStateMachineDescriptor_WithUnserializableInstance_ThrowsException (2)	< 1 ms
CreateStateMachineDescriptor_WithUnserializableInstance_ThrowsException(null)	< 1 ms
CreateStateMachineDescriptor_WithUnserializableInstance_ThrowsException(TestDemonstrator.Ticket.D...	< 1 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException (7)	130 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	8 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	11 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	22 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	14 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	12 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	37 ms
CreateStructureDescriptor_WithUndeserializableContent_ThrowsException("../../../DescriptorFactoryTests...	26 ms
CreateStructureDescriptor_WithUnserializableInstance_ThrowsException (2)	< 1 ms
CreateStructureDescriptor_WithUnserializableInstance_ThrowsException(BaSyx.Models.Core.AssetAdmini...	< 1 ms
CreateStructureDescriptor_WithUnserializableInstance_ThrowsException(null)	< 1 ms
TestDemonstrator.Tests.TestObjectProviderTests (10)	12.2 sec
TestObjectProviderTests (10)	12.2 sec
RetrieveActiveTestObjects_TestObjectsCountEqualsUploadedActiveTicketsCount	3.3 sec
RetrievePassiveTestObjects_TestObjectsCountEqualsUploadedPassiveTicketsCount	944 ms
RetrieveProcessedTestObjects_TestObjectsCountEqualsProcessedTicketsCount	1 sec

Abbildung A.11: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 2)

RetrieveProcessedTestObjects_TestObjectsCountEqualsProcessedTicketsCount	1 sec
RetrieveTestObject_RetrievesValidTestObject	800 ms
RetrieveTestObjectById_RetrievesTestObjectOnUploadedTicketWithKnownId	1.1 sec
RetrieveTestObjectById_UnknownTicketId_ThrowsException	951 ms
RetrieveTestObjectByUri_RetrievesTestObjectOnUploadedTicketWithKnownUri	1.1 sec
RetrieveTestObjectByUri_UnknownTicketUri_ThrowsException	945 ms
RetrieveTestObjects_TestObjectsCountEqualsUploadedTicketsCount	1.1 sec
RetrieveUnprocessedTestObjects_TestObjectsCountEqualsUnprocessedTicketsCount	863 ms
TestDemonstrator.Tests.TestObjectTests (32)	5.6 sec
TestObjectTests (32)	5.6 sec
InstantiateTestObject_FromActiveTicketWithNonExistentAAS_ThrowsException("http://localhost:65535","../...	4.1 sec
InstantiateTestObject_FromValidActiveTicket_ReturnsValidHandler("http://localhost:4999","../../../Descriptor...	68 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler (30)	1.5 sec
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	16 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	77 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	44 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	92 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	49 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	26 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	50 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	29 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	36 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	44 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	13 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	27 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	24 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	27 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	27 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	21 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	32 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	45 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	17 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	95 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	70 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	115 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	74 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	90 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	83 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	100 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	95 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	15 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	15 ms

Abbildung A.12: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 3)

InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	15 ms
InstantiateTestObject_FromValidPassiveTicket_ReturnsValidHandler("../../../DescriptorFactoryTests/TestDat...	13 ms
TestDemonstrator.Tests.TestOrchestratorTests (12)	15.9 sec
OrchestratorTests (12)	15.9 sec
ExecuteActiveTestPipelines_ExecutesAndProtocolsTestSuitesForEachRegisteredTicket	1.5 sec
ExecutePassiveTestPipelines_ExecutesAndProtocolsTestSuitesForEachRegisteredTicket	1.2 sec
ExecuteProcessedTestPipelines_ExecutesAndProtocolsTestSuitesForEachRegisteredTicket	1.1 sec
ExecuteTestPipeline_ExecutesAndProtocolsInvalidTestSuite_ThrowsException	922 ms
ExecuteTestPipeline_ExecutesAndProtocolsMultipleTestSuites	1.1 sec
ExecuteTestPipeline_ExecutesAndProtocolsSingleTestSuite	1.2 sec
ExecuteTestPipelineById_ExecutesAndProtocolsTestSuite	1.6 sec
ExecuteTestPipelineById_OnUnknownTicketId_ThrowsException	769 ms
ExecuteTestPipelineByUri_ExecutesAndProtocolsTestSuite	1.9 sec
ExecuteTestPipelineByUri_OnUnknownUri_ThrowsException	1.9 sec
ExecuteTestPipelines_ExecutesAndProtocolsTestSuitesForEachRegisteredTicket	1.3 sec
ExecuteUnprocessedTestPipelines_ExecutesAndProtocolsTestSuitesForEachRegisteredTicket	1.4 sec
TestDemonstrator.Tests.TestOrchestratorTests.TestSuites (6)	< 1 ms
ExamplePipelineLevel01 (2)	< 1 ms
FailingTest	< 1 ms
PassingTest	< 1 ms
ExamplePipelineLevel02 (2)	< 1 ms
FailingTest	< 1 ms
PassingTest	< 1 ms
ExamplePipelineLevel03 (2)	< 1 ms
FailingTest	< 1 ms
PassingTest	< 1 ms
TestDemonstrator.Tests.TestRunnerTests (3)	77 ms
NUnitRunnerTests (3)	77 ms
RunTestSuite_ExecutesTestsAndGeneratesTestReport	25 ms
RunTestSuite_WithInvalidTestObject_ThrowsException	21 ms
RunTestSuite_WithUnsuitableOrInvalidTestClass_ThrowsException	31 ms
TestDemonstrator.Tests.TestRunnerTests.TestSuites (1)	< 1 ms
ExampleTestSuite (1)	< 1 ms
PassingTest	< 1 ms
TestDemonstrator.Tests.TicketBuilderTests (43)	1.6 sec
BuilderTests (43)	1.6 sec
BuildActiveTicket_WithoutStateMachineDescriptor_ReturnsValidActiveTicket("../../../DescriptorFactoryTests/...	9 ms
BuildActiveTicket_WithoutStructureDescriptor_ThrowsException("http://localhost:4999","../../../DescriptorF...	13 ms
BuildActiveTicket_WithoutUri_ThrowsException("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW...	19 ms
BuildActiveTicket_WithPassiveShell_ThrowsException("http://localhost:4999","../../../DescriptorFactoryTests/...	24 ms
BuildActiveTicket_WithPassiveType_ThrowsException(Passive,"http://localhost:4999","../../../DescriptorFacto...	21 ms
BuildActiveTicket_WithStateMachineDescriptor_ReturnsValidActiveTicket("../../../DescriptorFactoryTests/Tes...	20 ms

Abbildung A.13: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 4)

BuildActiveTicket_WithStateMachineDescriptor_ReturnsValidActiveTicket("../../../DescriptorFactoryTests/Tes...	20 ms
BuildActiveTicket_WithUndefinedType_ThrowsException(Undefined,"http://localhost:4999","../../../Descript...	17 ms
BuildPassiveTicket_WithActiveType_ThrowsException(Active,"../../../DescriptorFactoryTests/TestData/Passive...	16 ms
BuildPassiveTicket_WithoutPassiveShell_ThrowsException("../../../DescriptorFactoryTests/TestData/PassiveS...	6 ms
BuildPassiveTicket_WithoutStructureDescriptor_ThrowsException("../../../DescriptorFactoryTests/TestData/P...	8 ms
BuildPassiveTicket_WithStateMachineDescriptor_ThrowsException("../../../DescriptorFactoryTests/TestData/...	29 ms
BuildPassiveTicket_WithUndefinedType_ThrowsException(Undefined,"../../../DescriptorFactoryTests/TestDat...	17 ms
BuildPassiveTicket_WithUri_ThrowsException("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/0...	14 ms
BuildPassiveTicket_ReturnsValidPassiveTicket (30)	1.4 sec
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/...	15 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/...	49 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/...	40 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/HTW/...	102 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	41 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	25 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	51 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	28 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	30 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	35 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	19 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	24 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	25 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	37 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	21 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	17 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	24 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	51 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	21 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	79 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	77 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	109 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	91 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	81 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	83 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	84 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	86 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	24 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	12 ms
BuildPassiveTicket_ReturnsValidPassiveTicket("../../../DescriptorFactoryTests/TestData/PassiveShells/IDTA/...	13 ms
TestDemonstrator.Tests.TicketRepositoryTests (12)	9.2 sec
TicketRepositoryTests (12)	9.2 sec
DeleteTicket_DeletesUploadedTicketSuccessfully	631 ms

Abbildung A.14: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 5)

TestDemonstrator.Tests.TicketRepositoryTests (12)	9.2 sec
TicketRepositoryTests (12)	9.2 sec
DeleteTicket_DeletesUploadedTicketSuccessfully	631 ms
DeleteTicketById_DeletesUploadedTicketSuccessfully	776 ms
DeleteTicketById_OnUnknownTicket_ThrowsException	729 ms
RetrieveAllTickets_ObtainsAllTicketsSuccessfully	1 sec
RetrieveTicketById_OnUnknownTicket_ThrowsException	581 ms
RetrieveTicketById_RetrievesUploadedTicketSuccessfully	1.2 sec
RetrieveTicketByUri_OnUnknownTicket_ThrowsException	950 ms
RetrieveTicketByUri_RetrievesUploadedTicketSuccessfully	743 ms
UpdateTicket_OnUnknownTicket_ThrowsException	697 ms
UpdateTicket_UpdatesUploadedTicketSuccessfully	756 ms
UploadTicket_UploadsActiveTicketSuccessfully	608 ms
UploadTicket_UploadsPassiveTicketSuccessfully	579 ms
TestDemonstrator.Tests.TicketTests (1)	< 1 ms
TicketTests (1)	< 1 ms
InstantiateTestTicket_ReturnsTicketWithGuidAndCreationTimestamp	< 1 ms
TestDemonstratorAAS.Tests (41)	27.3 sec
TestDemonstratorAAS.Tests.OrchestrationServiceTests (8)	7.5 sec
OrchestrationServiceTests (8)	7.5 sec
ExecuteTestPipelineById_RunsTestsCreatesProtocolAndReturnsSuccessMessage("<?xml version=\"1.0\" en...	2.8 sec
ExecuteTestPipelineById_WithUnknownId_ThrowsException	542 ms
ExecuteTestPipelineByUri_RunsTestsCreatesProtocolAndReturnsSuccessMessage("http://localhost:4999/","...	1.5 sec
ExecuteTestPipelineByUri_WithUnknownUri_ThrowsException("http://localhost:4999")	559 ms
ExecuteTestPipelineById_WithInvalidId_ThrowsException (2)	975 ms
ExecuteTestPipelineById_WithInvalidId_ThrowsException(" ")	466 ms
ExecuteTestPipelineById_WithInvalidId_ThrowsException(null)	509 ms
ExecuteTestPipelineByUri_WithInvalidUri_ThrowsException (2)	1.2 sec
ExecuteTestPipelineByUri_WithInvalidUri_ThrowsException(" ")	514 ms
ExecuteTestPipelineByUri_WithInvalidUri_ThrowsException(null)	642 ms
TestDemonstratorAAS.Tests.RepositoryServiceTests (33)	19.8 sec
RepositoryServiceTests (33)	19.8 sec
DeleteTicket_DeletesTicket("<?xml version=\"1.0\" encoding=\"utf-16\"?>\r\n<aas:aasenv xmlns:xsi=\"htt...	603 ms
DeleteTicket_WithInvalidId_ThrowsException("")	683 ms
DeleteTicket_WithUnknownId_ThrowsException	774 ms
RetrieveTicketById_RetrievesTicket("<?xml version=\"1.0\" encoding=\"utf-16\"?>\r\n<aas:aasenv xmlns:x...	637 ms
RetrieveTicketById_WithInvalidId_ThrowsException("")	483 ms
RetrieveTicketById_WithUnknownId_ThrowsException	359 ms
RetrieveTicketByUri_RetrievesTicket("http://localhost:4999","<?xml version=\"1.0\" encoding=\"utf-16\"?>...	649 ms
RetrieveTicketByUri_WithInvalidUri_ThrowsException("")	646 ms
RetrieveTicketByUri_WithUnknownUri_ThrowsException("http://localhost:4999")	719 ms
UpdateActiveTicket_WithInvalidUri_ThrowsException (2)	1.2 sec

Abbildung A.15: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 6)

TestDemonstratorAAS.Tests.RepositoryServiceTests (33)	19.8 sec
RepositoryServiceTests (33)	19.8 sec
DeleteTicket_DeletesTicket("<?xml version=\"1.0\" encoding=\"utf-16\"?>\r\n<aas:aasenv xmlns:xsi=\"htt...	603 ms
DeleteTicket_WithInvalidId_ThrowsException("")	683 ms
DeleteTicket_WithUnknownId_ThrowsException	774 ms
RetrieveTicketById_RetrievesTicket("<?xml version=\"1.0\" encoding=\"utf-16\"?>\r\n<aas:aasenv xmlns:x...	637 ms
RetrieveTicketById_WithInvalidId_ThrowsException("")	483 ms
RetrieveTicketById_WithUnknownId_ThrowsException	359 ms
RetrieveTicketByUri_RetrievesTicket("http://localhost:4999","<?xml version=\"1.0\" encoding=\"utf-16\"?>...	649 ms
RetrieveTicketByUri_WithInvalidUri_ThrowsException("")	646 ms
RetrieveTicketByUri_WithUnknownUri_ThrowsException("http://localhost:4999")	719 ms
UpdateActiveTicket_WithInvalidUri_ThrowsException (2)	1.2 sec
UpdateActiveTicket_WithInvalidUri_ThrowsException("00000000-0000-0000-0000-000000000000","http:...	489 ms
UpdateActiveTicket_WithInvalidUri_ThrowsException("00000000-0000-0000-0000-000000000000","http:...	675 ms
UpdateActiveTicket_WithNullOrEmptyArgument_ThrowsException (4)	2.2 sec
UpdateActiveTicket_WithNullOrEmptyArgument_ThrowsException("","http://localhost:4999","<?xml versi...	542 ms
UpdateActiveTicket_WithNullOrEmptyArgument_ThrowsException("00000000-0000-0000-0000-0000000...	658 ms
UpdateActiveTicket_WithNullOrEmptyArgument_ThrowsException("00000000-0000-0000-0000-0000000...	560 ms
UpdateActiveTicket_WithNullOrEmptyArgument_ThrowsException("00000000-0000-0000-0000-0000000...	482 ms
UploadActiveTicket_WithInvalidUri_ThrowsException (2)	1.1 sec
UploadActiveTicket_WithInvalidUri_ThrowsException("http://localhost:65536","<?xml version=\"1.0\" en...	474 ms
UploadActiveTicket_WithInvalidUri_ThrowsException("http:/invalid:65533","<?xml version=\"1.0\" encodi...	592 ms
UploadActiveTicket_WithNullOrEmptyArgument_ThrowsException (3)	1.7 sec
UploadActiveTicket_WithNullOrEmptyArgument_ThrowsException("","<?xml version=\"1.0\" encoding=\...	540 ms
UploadActiveTicket_WithNullOrEmptyArgument_ThrowsException("<?xml version=\"1.0\" encoding=\"u...	486 ms
UploadActiveTicket_WithNullOrEmptyArgument_ThrowsException("http://localhost:4999","","<?xml versi...	680 ms
UpdateActiveTicket_InvalidStateMachineDescriptor_ThrowsException("00000000-0000-0000-0000-000000...	576 ms
UpdateActiveTicket_WithInvalidId_ThrowsException("00000000-0000-0000-0000-000000000000 0","http:/...	782 ms
UpdateActiveTicket_WithInvalidStructureDescriptor_ThrowsException("00000000-0000-0000-0000-000000...	658 ms
UpdateActiveTicket_WithValidArguments_UpdatesTicket("http://localhost:4999","<?xml version=\"1.0\" en...	710 ms
UpdatePassiveTicket_InvalidPassiveShell_ThrowsException("00000000-0000-0000-0000-000000000000","<...	597 ms
UpdatePassiveTicket_InvalidStructureContent_ThrowsException("00000000-0000-0000-0000-00000000000...	565 ms
UpdatePassiveTicket_WithValidArguments_UploadsTicket("<?xml version=\"1.0\" encoding=\"utf-16\"?>\r...	712 ms
UploadActiveTicket_InvalidStateMachineDescriptor_ThrowsException("http://localhost:4999","<?xml versio...	567 ms
UploadActiveTicket_WithInvalidStructureDescriptor_ThrowsException("http://localhost:4999","<?xml versi...	455 ms
UploadActiveTicket_WithValidArguments_UploadsTicket("http://localhost:4999","<?xml version=\"1.0\" en...	610 ms
UploadPassiveTicket_InvalidPassiveShell_ThrowsException("<?xml version=\"1.0\"?>\r\n<aas:aasenv xmlns...	739 ms
UploadPassiveTicket_InvalidStructureContent_ThrowsException("<?xml version=\"1.0\" encoding=\"utf-16...	486 ms
UploadPassiveTicket_WithValidArguments_UploadsTicket("<?xml version=\"1.0\" encoding=\"utf-16\"?>\r...	597 ms

Abbildung A.16: Ergebnisse aus der Testung der Softwarearchitektur (Aussschnitt Nr. 7)

10/19/22, 4:44 PM TestManagementShell

TestManagementShell - Asset Administration Shell UI

Generic UI to discover the Asset Administration Shell

Asset Administration Shell

IdShort	TestManagementShell
Identifier (IRI)	http://htw-berlin.de/shells/TestManagementShell/4c10347a-ec59-4bfb-9d13-afae1a0dd49a
Description[de-DE]	Verwaltungsschale zum Testen von Industrie 4.0 Komponenten (Prototyp)
Description[en-US]	Asset administration shell for verification of Industrie 4.0 components (prototype)

Asset

IdShort	TestDemonstrator
Identifier (IRI)	http://htw-berlin.de/assets/TestDemonstrator/e4b9640c-8461-4954-8e53-64c9b97bbad1
Description[de-DE]	Softwarekomponente Test Demonstrator
Description[en-US]	Software component test demonstrator

TicketManagement

IdShort	TicketManagement
Identifier (IRI)	TicketManagement

Submodel-Elements

TicketsCount

Description[EN]	Obtains the number of tickets in the ticket repository
Description[DE]	Gibt die Anzahl von Test-Rezepten im Ticketverzeichnis an
ValueType	int

16 Retrieve Update Clear

PassiveTicketsCount

Description[EN]	Obtains the number of passive tickets in the ticket repository

localhost:8080/ui 1/7

Abbildung A.17: Test Demonstrator VWS - Struktur (Aussschnitt Nr. 1)

10/19/22, 4:44 PM TestManagementShell

Description[DE]	Gibt die Anzahl von passiven Test-Rezepten im Ticketverzeichnis an
ValueType	int

1 Retrieve Update Clear

ActiveTicketsCount

Description[EN]	Obtains the number of active tickets in the ticket repository
Description[DE]	Gibt die Anzahl von aktiven Test-Rezepten im Ticketverzeichnis an
ValueType	int

15 Retrieve Update Clear

ProcessedTicketsCount

Description[EN]	Obtains the number of processed tickets in the ticket repository
Description[DE]	Gibt die Anzahl von bearbeiteten Test-Rezepten im Ticketverzeichnis an
ValueType	int

14 Retrieve Update Clear

UnprocessedTicketsCount

Description[EN]	Obtains the number of unprocessed tickets in the ticket repository
Description[DE]	Gibt die Anzahl von nicht bearbeiteten Test-Rezepten im Ticketverzeichnis an
ValueType	int

2 Retrieve Update Clear

UploadActiveTicket

Description[EN]	Uploads an active ticket to the ticket repository	
Description[DE]	Lädt aktives Test-Rezept im Ticketverzeichnis hoch	
Uri	string	Input Argument Value
Structure	string	Input Argument Value

localhost:8080/ui 2/7

Abbildung A.18: Test Demonstrator VWS - Struktur (Aussschnitt Nr. 2)

10/19/22, 4:44 PM TestManagementShell

StateMachine	string	Input Argument Value

Execute Clear

UploadPassiveTicket

Description[EN]	Uploads a passive ticket to the ticket repository	
Description[DE]	Lädt passives Test-Rezept im Ticketverzeichnis hoch	
Content	string	Input Argument Value
Structure	string	Input Argument Value

Execute Clear

UpdateActiveTicket

Description[EN]	Updates an active ticket based on a ticket identificator	
Description[DE]	Aktualisiert aktives Test-Rezept nach Eingabe des entsprechenden Identifikators	
TicketId	string	Input Argument Value
Uri	string	Input Argument Value
Structure	string	Input Argument Value
StateMachine	string	Input Argument Value

Execute Clear

UpdatePassiveTicket

Description[EN]	Updates a passive ticket based on a ticket identificator	
Description[DE]	Aktualisiert passives Test-Rezept nach Eingabe des entsprechenden Identifikators	
TicketId	string	Input Argument Value
Content	string	Input Argument Value
Structure	string	Input Argument Value

localhost:8080/ui 3/7

Abbildung A.19: Test Demonstrator VWS - Struktur (Aussschnitt Nr. 3)

10/19/22, 4:44 PM TestManagementShell

Execute Clear

DeleteTicket

Description[EN]	Deletes a ticket based on a ticket identificator	
Description[DE]	Löscht ein Test-Rezept nach Eingabe des entsprechenden Ticket-Identifikators	
TicketId	string	Input Argument Value

Execute Clear

RetrieveTicketById

Description[EN]	Retrieves a ticket based on a ticket identificator	
Description[DE]	Ruft Test-Rezept nach Eingabe des entsprechenden Identifikators ab	
TicketId	string	Input Argument Value
CreatedAt	string	Output Argument Value
UpdatedAt	string	Output Argument Value
Type	string	Output Argument Value
Uri	string	Output Argument Value
PassiveShellContent	string	Output Argument Value
StructureDescriptor	string	Output Argument Value
StateMachineDescriptor	string	Output Argument Value
TestReport	string	Output Argument Value

Execute Clear

RetrieveTicketByUri

Description[EN]	Retrieves an active ticket based on the registered URI address
Description[DE]	Ruft aktives Test-Rezept nach Eingabe der entsprechenden URI-Adresse ab

localhost:8080/ui 4/7

Abbildung A.20: Test Demonstrator VWS - Struktur (Aussschnitt Nr. 4)

TicketUri	string	
CreatedAt	string	Output Argument Value
UpdatedAt	string	Output Argument Value
Type	string	Output Argument Value
TicketId	string	Output Argument Value
ShellContent	string	Output Argument Value
StructureDescriptor	string	Output Argument Value
StateMachineDescriptor	string	Output Argument Value
TestReport	string	Output Argument Value

Execute Clear

RetrieveTicketIdentificators

Description[EN]	Retrieves all ticket identificators as a comma separated string
Description[DE]	Ruft alle registrierten Ticket-Identifikatoren ab
TicketIDs	string Output Argument Value

Execute Clear

TestExecution

IdShort	TestExecution
Identifier (IRI)	TestExecution

Submodel-Elements

ExecuteTestPipelineById

Description[EN]	Executes a test pipeline for a single ticket based on the given ticket identificator

Abbildung A.21: Test Demonstrator VWS - Struktur (Aussschnitt Nr. 5)

Description[DE]	Führt eine Testpipeline für ein einzelnes Ticket basierend auf der angegebenen Ticket-ID aus	
TicketId	string	Input Argument Value

Execute Clear

ExecuteTestPipelineByUri

Description[EN]	Executes a test pipeline for an active test object based on the URI address specified by the corresponding ticket
Description[DE]	Führt eine Testpipeline für ein aktives Testobjekt basierend auf der durch das entsprechende Ticket angegebenen URI-Adresse aus
AASUri	string Input Argument Value

Execute Clear

ExecuteActiveTestPipelines

Description[EN]	Executes a test pipeline for each active ticket in the repository
Description[DE]	Führt eine Testpipeline für jedes aktive Ticket im Repository aus

Execute Clear

ExecutePassiveTestPipelines

Description[EN]	Executes a test pipeline for each passive ticket in the repository
Description[DE]	Führt eine Testpipeline für jedes passive Ticket im Repository aus

Execute Clear

ExecuteTestPipelines

Description[EN]	Executes a test pipeline for each ticket in the repository
Description[DE]	Führt eine Testpipeline für jedes Ticket im Repository aus

Execute Clear

Abbildung A.22: Test Demonstrator VWS - Struktur (Aussschnitt Nr. 6)

Update README.md
Vasil Denkov authored 22 hours ago

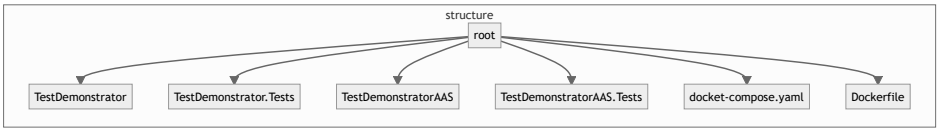
Name	Last commit	Last update
..		
TestDemonstrator.Tests	pipeline steps included, dockerized app, added readme etc.	23 hours ago
TestDemonstrator	pipeline steps included, dockerized app, added readme etc.	23 hours ago
TestDemonstratorAAS.Tests	pipeline steps included, dockerized app, added readme etc.	23 hours ago
TestDemonstratorAAS	pipeline steps included, dockerized app, added readme etc.	23 hours ago
Dockerfile	pipeline steps included, dockerized app, added readme etc.	23 hours ago
README.md	Update README.md	22 hours ago
appsettings.json	pipeline steps included, dockerized app, added readme etc.	23 hours ago
docker-compose.yaml	pipeline steps included, dockerized app, added readme etc.	23 hours ago

README.md

Test Demonstrator

Structural overview

The following flowchart provides an overview of the general structure of this repository. The synthesised software architecture is composed of two projects - **TestDemonstrator** and **TestDemonstratorAAS**, each of which was created as a result of a test driven development process. The corresponding test projects are **TestDemonstrator.Tests** and **TestDemonstratorAAS.Tests**, respectively.



The **TestDemonstrator** implements the core components and a demo to show their usage. The **TestDemonstratorAAS**-project contains high-level modules and provides components that define the structure of the AAS. A docker-compose and a docker file are provided to run the administration shell in an isolated environment. The container runtime starts an SQL database used as a ticket repository. This serves as a prerequisite for the usage of the ticket repository submodel. The user can access the database through port 3306. The connection string is defined by the docker-compose file. In order to start the AAS the same connection string must be specified in the appsettings.json file of the **TestDemonstratorAAS**-Project. The default *User Id* and server identification must be taken into consideration. See also MS SQL Server 2019. The AAS is started on HTTP port 8080 and HTTPS port 5000. Changing the ports requires adaptations of the docker-compose file.

Dependencies

The following diagram illustrates the dependencies between the four projects. As shown, the **TestDemonstratorAAS**-project cannot be compiled without the core components implemented by the **TestDemonstrator**.

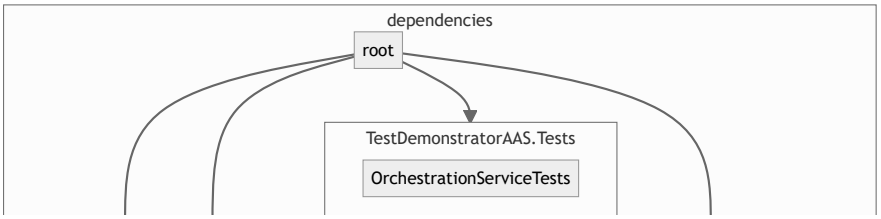


Abbildung A.23: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 1)

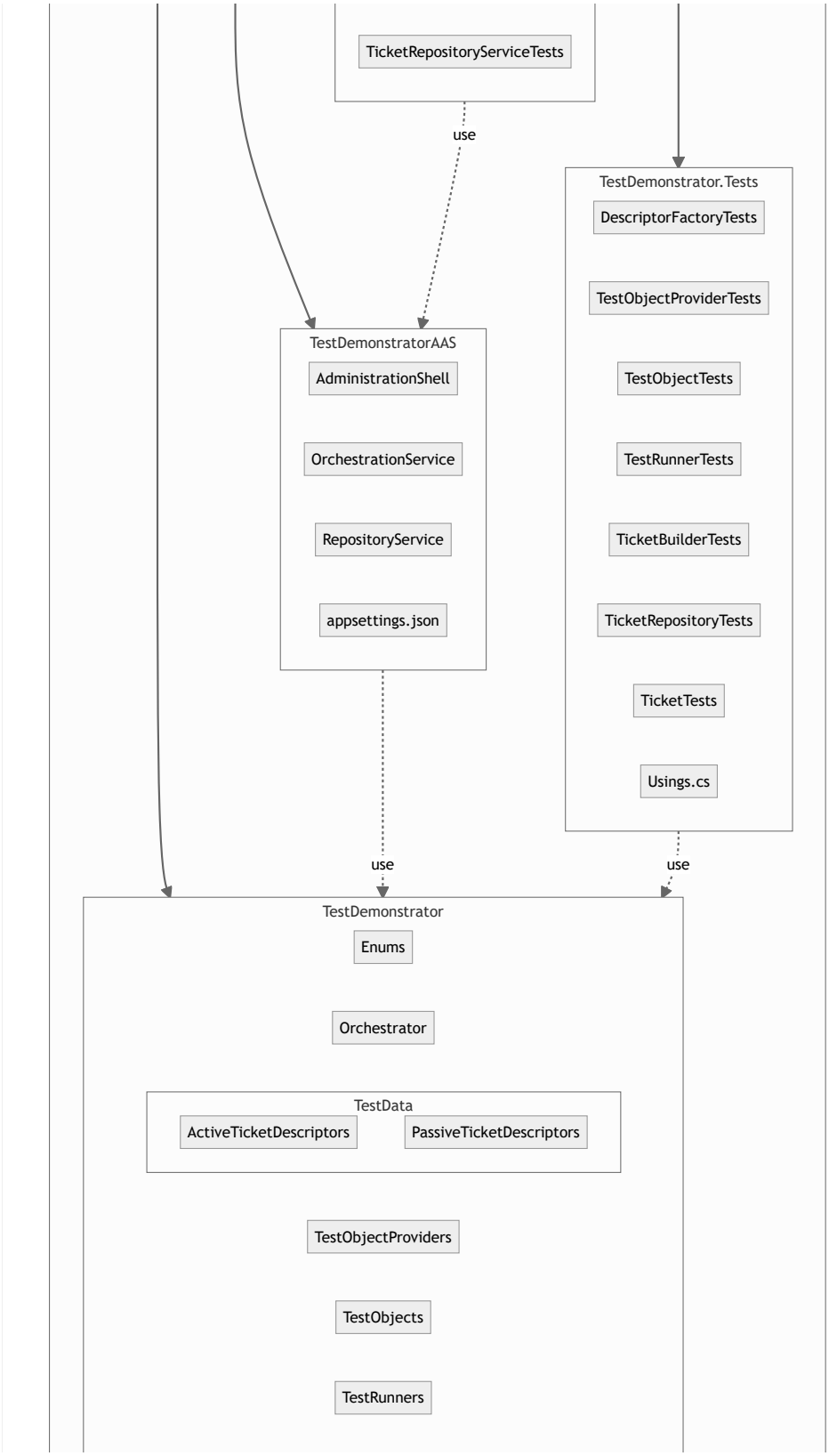
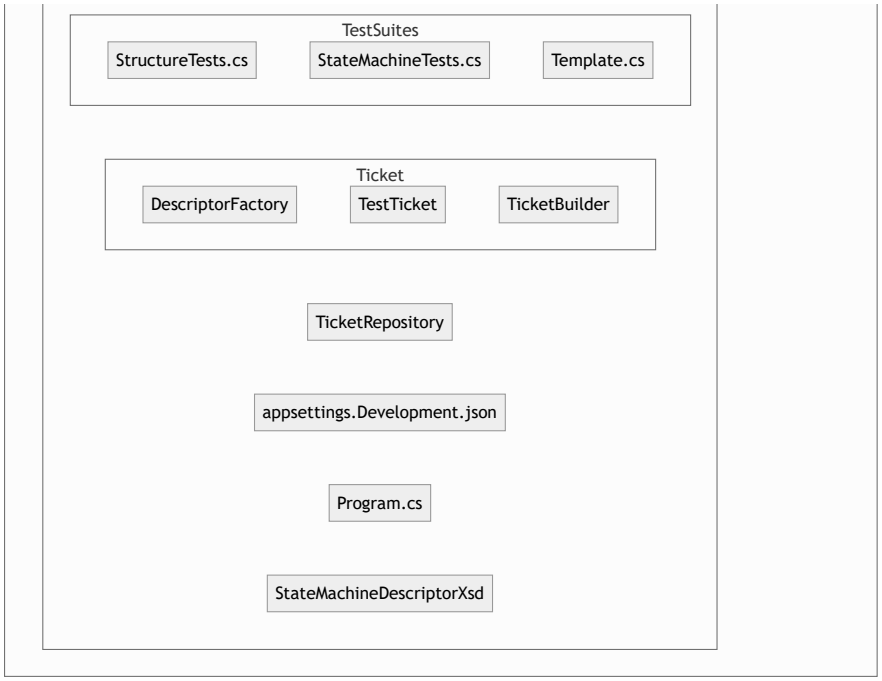


Abbildung A.24: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 2)



The local execution of each test project requires a connection string to a test database, which must be specified in the *appsettings.Development.json*-file.

Utilization

In order to run the application, the following command must be executed from the root directory.

```
docker-compose up -d
```

The database must be created and the demonstrator will be build. Potential warnings can be associated with the generated TestDemonstratorAAS/StateMachineDescriptorXsd.Designer.cs file. For more information read the thesis.

The AAS should be available under http://localhost:8080.

Handling

In order to perform a test pipeline, a ticket must be loaded through the **TicketManagement** submodel.

A passive test pipeline consists only of structure tests. A passive ticket can be uploaded through the **UploadPassiveTicket** operation. First, an XML file must be provided that represents the current test object. Second, the expected structure must be provided as an XML file as well. Both the AAS and the structure descriptor can be created via the AASX Package Explorer.

The operation can be called through swagger in the following manner.

1. Go to Swagger.
2. Find the following POST service

```
/aas/submodels/{submodelIdShort}/submodel/submodelElements/{idShortPathToOperation}/invoke
```

3. Type "**TicketManagement**" into the "**submodelIdShort**" field.
4. Type "**UploadPassiveTicket**" into the "**idShortPathToOperation**" field.
5. Give the following JSON object in the "**Request body**" field.

▶ Request body

The request must look like this:

Abbildung A.25: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 3)

POST /aas/submodels/{submodelIdShort}/submodel/submodelElements/{idShortPathToOperation}/invoke Invokes a specific operation from the Submodel synchronously or asynchronously

Parameters — Cancel — Reset

Name	Description
submodelIdShort * required string *(path)*	Submodel's short id TicketManagement
idShortPathToOperation * required string *(path)*	The IdShort path to the Operation UploadPassiveTicket
async boolean *(query)*	Determines whether the execution of the operation is asynchronous (true) or not (false) --

Request body — application/json

The parameterized request object for the invocation


```
{
    "requestId": "644f60f5-96b9-41aa-a0dc-2b9b95e912ec",
    "inputArguments": [
        {
            "modelType": {
                "name": "OperationVariable"
            },
            "value": {
                "idShort": "Content",
                "modelType": {
                    "name": "Property"
                },
                "valueType": "string",
                "value": "<?xml version=\"1.0\" encoding=\"utf-16\"?> <aas:aasenv xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:IEC61360=\"http://www.admin-shell.io/IEC61360/2/0\"
xmlns:aas=\"http://www.admin-shell.io/aas/2/0\">   <aas:assetAdministrationShells>     <aas:assetAdministrationShell>       <aas:idShort>Demo_VWS</aas:idShort>       <aas:description>
<aas:langString lang=\"de\">1. VWS</aas:langString>       <aas:langString lang=\"en\">1. AAS</aas:langString>       </aas:description>       <aas:identification
idType=\"IRI\">Demo_VWS</aas:identification>       <aas:administration>       <aas:version>1.0</aas:version>       <aas:revision>120</aas:revision>       </aas:administration>       <aas:assetRef>
<aas:keys>       <aas:key type=\"Asset\" idType=\"IRI\" local=\"true\">urn:org.eclipse.basyx:assets:demo-asset:1.0.0</aas:key>       </aas:keys>       </aas:assetRef>       <aas:submodelRefs>
```


The response body contains the following message (if the shell content and the structure descriptor were valid):

Request URL

```
http://localhost:8080/aas/submodels/TicketManagement/submodel/submodelElements/UploadPassiveTicket/invoke
```

Server response

Code	Details
200	

Response body


```
{
  "requestId": "644f60f5-96b9-41aa-a0dc-2b9b95e912ec",
  "inoutputArguments": [],
  "outputArguments": [],
  "operationResult": {
    "success": true,
    "messages": [
      {
        "messageType": "Information",
        "text": "Successfully uploaded ticket with Id: 0574e56d-7019-4bc9-8783-b790ca89ad7c"
      }
    ]
  },
  "executionState": "Completed"
}
```


Response headers

```
access-control-allow-credentials: true
access-control-allow-methods: *
access-control-allow-origin: *
content-length: 287
content-type: application/json; charset=utf-8
date: Tue,18 Oct 2022 12:19:51 GMT
server: Kestrel
```

7. After the ticket was successfully uploaded, we want to retrieve all registered ticket identificators. Execute the **RetrieveTicketIdentificators** operation of the **TicketManagement** submodel as showed in the following image. Use the following request body.

▶ Request body

Abbildung A.26: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 4)

POST /aas/submodels/{submodelIdShort}/submodel/submodelElements/{idShortPathToOperation}/invoke

Parameters

Name	Description
submodelIdShort * required string *(path)*	Submodel's short id TicketManagement
idShortPathToOperation * required string *(path)*	The IdShort path to the Operation RetrieveTicketIdentificators
async boolean *(query)*	Determines whether the execution of the operation is asynchronous (true) or not (false) --

Request body

The parameterized request object for the invocation


```
{
    "requestId": "46ce82f6-4270-4872-80e9-7759efab04fb",
    "timeout": 60000
}
```


Since there are many tickets in the repository, which would not be the case on the first run, the demonstrator responds with the following message. The operation returns a string of comma separated ticket identificators.

Response body


```
{
  "requestId": "46ce82f6-4270-4872-80e9-7759efab04fb",
  "inoutputArguments": [],
  "outputArguments": [
    {
      "modelType": {
        "name": "OperationVariable"
      },
      "value": {
        "idShort": "TicketIDs",
        "modelType": {
          "name": "Property"
        },
        "valueType": "string",
        "value": "20f021df-6e58-4932-8913-10710e4c9255,dffa3209-0694-4699-85cc-21af2eca133e,2ec8f3a3-ce15-4613-ae2f-3292fefd9f45,2d7bf1f0-6cb9-4f6f-ae72-3576ac320301,7670cc06-76fc-4e88-853b-4
bd765fdb951,426cc9f0-09d3-471e-9e4a-4fb364f6dcc0,df0fc4f7-2593-4d7b-8d45-555bd9ab8a67,d7f69e36-2c20-4b2c-a852-73176b04ddad,8dd25484-478c-4efd-bf30-7c1d635897ca,fe932dee-cbb2-457f-92b9-7d33060
cb133,54dca9fe-a415-44a8-b352-9f28e5b31048,0574e56d-7019-4bc9-8783-b790ca09ad7c,d26901f5-5f28-4ce7-a21e-d5a57940c84d,215876e5-db5e-4458-9664-de7332cd2baa,8845b734-1c1d-4e54-83ef-e7a0300b7b99,
65b1dad1-8adb-4248-ac7b-fae2e3bf116a"
      }
    }
  ],
  "operationResult": {
    "success": true,
    "messages": [
      {
        "messageType": "Information",
        "text": "Successfully obtained identificators!"
      }
    ]
```


Download

8. Copy the identificator and clear the request body.

9. Execute the **ExetuteTestPipelineById** operation of the **TestExecution** submodel. Use the following request body. Do not forget to specify the ID of the uploaded ticket.

▶ Request body

Abbildung A.27: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 5)

Name	Description
submodelIdShort * required string *(path)*	Submodel's short id TestExecution
idShortPathToOperation * required string *(path)*	The IdShort path to the Operation ExecuteTestPipelineById
async boolean *(query)*	Determines whether the execution of the operation is asynchronous (true) or not (false) --

Request body

The parameterized request object for the invocation

```
{
    "requestId": "46ce82f6-4270-4872-80e9-7759efab04fb",
    "inputArguments": [
        {
            "modelType": {
                "name": "OperationVariable"
            },
            "value": {
                "idShort": "TicketId",
                "modelType": {
                    "name": "Property"
                },
                "valueType": "string",
                "value": "0574e56d-7019-4bc9-8783-b790ca89ad7c"
            }
        }
    ],
    "timeout": 60000
}
```

After a while, the following response body is returned:

Response body

```
{
  "requestId": "46ce82f6-4270-4872-80e9-7759efab04fb",
  "inoutputArguments": [],
  "outputArguments": [],
  "operationResult": {
    "success": true,
    "messages": [
      {
        "messageType": "Information",
        "text": "Successfully executed test pipeline for ticket with Id: 0574e56d-7019-4bc9-8783-b790ca89ad7c"
      }
    ]
  },
  "executionState": "Completed"
}
```

Download

The test protocol is stored in the ticket now. Clear the request body.

10. Execute the **RetrieveTicketById** operation of the **TicketManagement** submodel. Use the following request body. Do not forget to specify the ID of the uploaded ticket.

▶ Request body

The response body contains the whole ticket as well as the generated test protocol as an XML string. The XML element **output** contains the following information.

```
2022-10-18 12:42:22.350 Expected submodels count:  8
2022-10-18 12:42:22.361 Actual submodels count:  8
2022-10-18 12:42:22.370 Expected IdShort of submodel number 0: SM_Process1
2022-10-18 12:42:22.380 Actual IdShort of submodel number 0:   SM_Process1
2022-10-18 12:42:22.389 Expected submodel elements count: 3
2022-10-18 12:42:22.399 Actual submodel elements count:   3
2022-10-18 12:42:22.409   Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-18 12:42:22.419   Actual IdShort of submodel element 0:   SMC_Statemachine
2022-10-18 12:42:22.428   Submodel element SMC_Statemachine is an element collection
2022-10-18 12:42:22.437   SMC_Statemachine: Expected elements count: 2
2022-10-18 12:42:22.447   SMC_Statemachine: Actual submodel elements count: 2
2022-10-18 12:42:22.456           Expected IdShort of submodel element 0: Prop_State
2022-10-18 12:42:22.466           Actual IdShort of submodel element 0:   Prop_State
```

Abbildung A.28: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 6)

```
2022-10-18 12:42:22.476          Expected IdShort of submodel element 1: Prop_Mode
2022-10-18 12:42:22.486          Actual IdShort of submodel element 1:   Prop_Mode
2022-10-18 12:42:22.495   Expected IdShort of submodel element 1: SMC_ProcessInfo
2022-10-18 12:42:22.504   Actual IdShort of submodel element 1:   SMC_ProcessInfo
2022-10-18 12:42:22.514   Submodel element SMC_ProcessInfo is an element collection
2022-10-18 12:42:22.523   SMC_ProcessInfo: Expected elements count: 5
2022-10-18 12:42:22.534   SMC_ProcessInfo: Actual submodel elements count: 5
2022-10-18 12:42:22.544          Expected IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-18 12:42:22.555          Actual IdShort of submodel element 0:   SMC_ElementOfProcessInfo1
2022-10-18 12:42:22.565          Submodel element SMC_ElementOfProcessInfo1 is an element collection
2022-10-18 12:42:22.575          SMC_ElementOfProcessInfo1: Expected elements count: 9
2022-10-18 12:42:22.584          SMC_ElementOfProcessInfo1: Actual submodel elements count: 9
2022-10-18 12:42:22.594                  Expected IdShort of submodel element 0: ProcessVariables1
2022-10-18 12:42:22.604                  Actual IdShort of submodel element 0:   ProcessVariables1
2022-10-18 12:42:22.614                  Expected IdShort of submodel element 1: ProcessVariables2
2022-10-18 12:42:22.623                  Actual IdShort of submodel element 1:   ProcessVariables2

...

2022-10-18 12:42:38.609 Test passed!
```

In order to upload and execute a test pipeline for an active test object with a PackML state machine, such as the one provided by the PackML example and the AAS generator, follow the steps described in the following README to generate an active AAS with PackML interface. Then upload a ticket through the **UpdateActiveTicket** operation of the **TicketManagement** submodel. A state machine descriptor can be found in the TestDemonstrator project.

Abbildung A.29: Test Demonstrator VWS - Inbetriebnahmeanleitung (Aussschnitt Nr. 7)

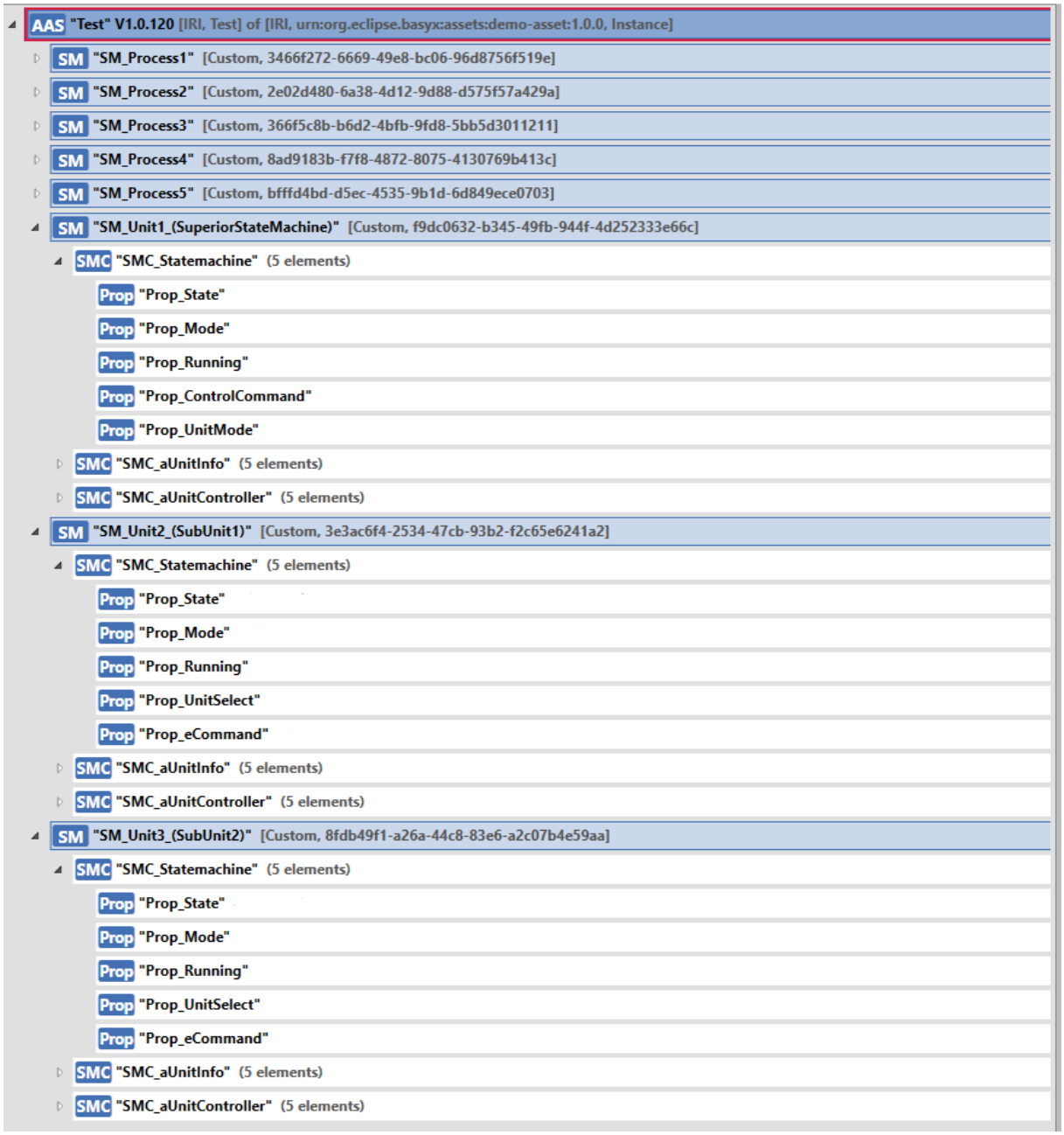


Abbildung A.30: VWS-Struktur einer als Testobjekt geltenden Führungskomponente

TicketManagement

IdShort	TicketManagement
Identifier (IRI)	TicketManagement

Submodel-Elements

TicketsCount

Description[EN]	Obtains the number of tickets in the ticket repository
Description[DE]	Gibt die Anzahl von Test-Rezepten im Ticketverzeichnis an
ValueType	int

26 Retrieve Update Clear

PassiveTicketsCount

Description[EN]	Obtains the number of passive tickets in the ticket repository
Description[DE]	Gibt die Anzahl von passiven Test-Rezepten im Ticketverzeichnis an
ValueType	int

25 Retrieve Update Clear

ActiveTicketsCount

Description[EN]	Obtains the number of active tickets in the ticket repository
Description[DE]	Gibt die Anzahl von aktiven Test-Rezepten im Ticketverzeichnis an
ValueType	int

1 Retrieve Update Clear

Abbildung A.31: Anzahl von Tickets im Rahmen der Inbetriebnahmephase

```
1  using System.Diagnostics;
2  using System.Globalization;
3
4  using NUnit.Framework;
5
6  using TestDemonstrator.TestObjects.Contracts;
7  using TestDemonstrator.TestRunners;
8
9  namespace PipelineTestSuites
10 {
11   public class Template
12   {
13     private ITestObject testObject = null!;
14
15     [OneTimeSetUp]
16     public void OneTimeSetUp()
17     {
18       using var listener = new ConsoleTraceListener()
19       {
20         Name = $"{GetType().Name}TraceListener"
21       };
22
23       Trace.Listeners.Add(listener);
24       testObject = TestObjectFetcher.CurrentTestObject;
25     }
26
27     [SetUp]
28     public void SetUp()
29     {
30
31     }
32
33     [Test]
34     public void Test()
35     {
36       Trace.WriteLine($"{DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss.fff",
           CultureInfo.InvariantCulture)}");
37     }
38
39     [TearDown]
40     public void TearDown()
41     {
42
43     }
44
45     [OneTimeTearDown]
46     public void OneTimeTearDown()
47     {
48       Trace.Flush();
49       Trace.Listeners.Remove($"{GetType().Name}TraceListener");
50     }
51   }
52 }
```

Listing A.1: Vorlage für die Erstellung von Test-Klassen

```xml
<?xml version="1.0" encoding="utf-16"?>
<descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <onetimesetup>
  </onetimesetup>
  <testCases>
  <testCase description="Operation ClearRepository sets ticket number to zero">
    <testSteps>
      <action submodel="TicketManagement" property="ClearRepository" wait="100"/>
      <assert submodel="TicketManagement" property="TicketsCount" valueType="System.Int32"
          value="0"/>
      <assert submodel="TicketManagement" property="ProcessedTicketsCount"
          valueType="System.Int32" value="0"/>
      <assert submodel="TicketManagement" property="UnprocessedTicketsCount"
          valueType="System.Int32" value="0"/>
      <assert submodel="TicketManagement" property="ActiveTicketsCount"
          valueType="System.Int32" value="0"/>
      <assert submodel="TicketManagement" property="PassiveTicketsCount"
          valueType="System.Int32" value="0"/>
    </testSteps>
  </testCase>
  <testCase description="Operation UploadTicket uploads ticket successfully">
    <setup>
      <action submodel="TicketManagement" property="ClearRepository" wait="100"/>
    </setup>
    <testSteps>
      <action submodel="TicketManagement" property="UploadPassiveTicket" wait="100">
        <inputArguments>
          <argument idShort="PassiveShellContent" valueType="System.String"
              value="../path/to/descriptor"/>
          <argument idShort="StructureDescriptorContent" valueType="System.String"
              value="../path/to/descriptor"/>
        </inputArguments>
      </action>
      <assert submodel="TicketManagement" property="TicketsCount" valueType="System.Int32"
          value="1"/>
      <assert submodel="TicketManagement" property="ProcessedTicketsCount"
          valueType="System.Int32" value="0"/>
      <assert submodel="TicketManagement" property="UnprocessedTicketsCount"
          valueType="System.Int32" value="1"/>
      <assert submodel="TicketManagement" property="ActiveTicketsCount"
          valueType="System.Int32" value="0"/>
      <assert submodel="TicketManagement" property="PassiveTicketsCount"
          valueType="System.Int32" value="1"/>
    </testSteps>
  </testCase>
  <testCase description="Operation ExecuteTestPipelineById processes a test receipt"/>
    <setup>
    <action submodel="TicketManagement" property="ClearRepository" wait="100"/>
    <action submodel="TicketManagement" property="UploadPassiveTicket" wait="100">
      <inputArguments>
        <argument idShort="PassiveShellContent" valueType="System.String"
            value="../path/to/descriptor"/>
        <argument idShort="StructureDescriptorContent" valueType="System.String"
            value="../path/to/descriptor"/>
      </inputArguments>
    </action>
    </setup>
```

```
44        <testSteps>
45        <action submodel="TestExecution" property="ExecuteTestPipelineById" wait="100">
46          <inputArguments>
47            <argument idShort="TicketId" valueType="System.String"
                value="46ce82f6-4270-4872-80e9-7759efab04fb"/>
48          </inputArguments>
49        </action>
50        <assert submodel="TicketManagement" property="ProcessedTicketsCount"
            valueType="System.String" value="1"/>
51        </testSteps>
52        <teardown>
53        </teardown>
54      </testCase>
55      </testCases>
56      <onetimeteardown>
57      </onetimeteardown>
58    </descriptor>
```

Listing A.2: Beispielhafte Verhaltensbeschreibung als einfache Basis für die Durchführung von Komponententests

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <xs:schema id="descriptor" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
3     <xs:simpleType name="dataType">
4       <xs:restriction base="xs:string">
5       <xs:enumeration value="System.Int32" />
6       <xs:enumeration value="System.Boolean" />
7       <xs:enumeration value="System.String" />
8       </xs:restriction>
9     </xs:simpleType>
10    <xs:element name="action">
11    <xs:complexType>
12      <xs:attribute name="submodel" type="xs:string" use="required"/>
13      <xs:attribute name="setter" type="xs:string" use="required"/>
14      <xs:attribute name="getter" type="xs:string" use="required"/>
15      <xs:attribute name="set" type="xs:string" use="required"/>
16      <xs:attribute name="setType" type="dataType" use="required"/>
17      <xs:attribute name="expect" type="xs:string" use="required"/>
18      <xs:attribute name="expectType" type="dataType" use="required"/>
19      <xs:attribute name="wait" type="xs:integer" use="required"/>
20    </xs:complexType>
21    </xs:element>
22    <xs:element name="descriptor" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
23    <xs:complexType>
24      <xs:choice minOccurs="1" maxOccurs="unbounded">
25      <xs:element ref="action" />
26      <xs:element name="onetimesetup" minOccurs="0" maxOccurs="unbounded">
27        <xs:complexType>
28        <xs:sequence>
29          <xs:element ref="action" minOccurs="1" maxOccurs="unbounded" />
30        </xs:sequence>
31        </xs:complexType>
32      </xs:element>
33      <xs:element name="paths" minOccurs="1" maxOccurs="1">
34        <xs:complexType>
35        <xs:sequence>
36          <xs:element name="path" minOccurs="1" maxOccurs="unbounded">
37          <xs:complexType>
```

```
38            <xs:sequence>
39            <xs:element name="setup" minOccurs="0" maxOccurs="1">
40              <xs:complexType>
41              <xs:sequence>
42                <xs:element ref="action" minOccurs="1" maxOccurs="unbounded" />
43              </xs:sequence>
44              </xs:complexType>
45            </xs:element>
46            <xs:element name="properties" minOccurs="1" maxOccurs="1">
47              <xs:complexType>
48              <xs:attribute name="submodel" type="xs:string" use="required"/>
49              <xs:attribute name="setter" type="xs:string" use="required"/>
50              <xs:attribute name="getter" type="xs:string" use="required"/>
51              </xs:complexType>
52            </xs:element>
53            <xs:element name="state" minOccurs="1" maxOccurs="unbounded">
54              <xs:complexType>
55              <xs:sequence>
56                <xs:element name="transition" minOccurs="1" maxOccurs="1">
57                <xs:complexType>
58                  <xs:attribute name="event" type="xs:string" use="required"/>
59                  <xs:attribute name="target" type="xs:string" use="required"/>
60                  <xs:attribute name="wait" type="xs:integer" use="required"/>
61                </xs:complexType>
62                </xs:element>
63              </xs:sequence>
64              <xs:attribute name="id" type="xs:string" use="required"/>
65              </xs:complexType>
66            </xs:element>
67            <xs:element name="teardown" minOccurs="0" maxOccurs="1">
68              <xs:complexType>
69              <xs:sequence>
70                <xs:element ref="action" minOccurs="1" maxOccurs="unbounded" />
71              </xs:sequence>
72              </xs:complexType>
73            </xs:element>
74            </xs:sequence>
75          </xs:complexType>
76          </xs:element>
77        </xs:sequence>
78        </xs:complexType>
79      </xs:element>
80      <xs:element name="onetimeteardown" minOccurs="0" maxOccurs="1">
81        <xs:complexType>
82        <xs:sequence>
83          <xs:element ref="action" minOccurs="1" maxOccurs="unbounded" />
84        </xs:sequence>
85        </xs:complexType>
86      </xs:element>
87      </xs:choice>
88    </xs:complexType>
89    </xs:element>
90  </xs:schema>
```

Listing A.3: XML-Schema für Zustandsautomatendeskriptoren

```xml
<?xml version="1.0" encoding="utf-16" ?>
<descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="../../../StateMachineDescriptorXsd.xsd">
  <onetimesetup>
    <action submodel="SM_Unit1_(SuperiorStateMachine)"
        setter="SMC_Statemachine/Prop_UnitMode" getter="SMC_Statemachine/Prop_Mode"
        set="2" setType="System.Int32" expect="2" expectType="System.Int32" wait="100"/>
  </onetimesetup>
  <paths>
    <path>
      <setup>
        <action submodel="SM_Unit1_(SuperiorStateMachine)"
            setter="SMC_Statemachine/Prop_ControlCommand"
            getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
            expect="Aborted" expectType="System.String" wait="100"/>
        <action submodel="SM_Unit2_(SubUnit1)" setter="SMC_Statemachine/Prop_UnitSelect"
            getter="SMC_Statemachine/Prop_UnitSelect" set="1" setType="System.Int32"
            expect="True" expectType="System.Boolean" wait="100"/>
        <action submodel="SM_Unit3_(SubUnit2)" setter="SMC_Statemachine/Prop_UnitSelect"
            getter="SMC_Statemachine/Prop_UnitSelect" set="1" setType="System.Int32"
            expect="True" expectType="System.Boolean" wait="100"/>
      </setup>
      <properties submodel="SM_Unit1_(SuperiorStateMachine)"
          setter="SMC_Statemachine/Prop_ControlCommand"
          getter="SMC_Statemachine/Prop_State"/>
      <state id="Aborted">
        <transition event="Clear" target="Stopped" wait="100"/>
      </state>
      <teardown>
        <action submodel="SM_Unit2_(SubUnit1)" setter="SMC_Statemachine/Prop_UnitSelect"
            getter="SMC_Statemachine/Prop_UnitSelect" set="0" setType="System.Int32"
            expect="False" expectType="System.Boolean" wait="100"/>
        <action submodel="SM_Unit3_(SubUnit2)" setter="SMC_Statemachine/Prop_UnitSelect"
            getter="SMC_Statemachine/Prop_UnitSelect" set="0" setType="System.Int32"
            expect="False" expectType="System.Boolean" wait="100"/>
      </teardown>
    </path>
    <path>
      <setup>
        <action submodel="SM_Unit1_(SuperiorStateMachine)"
            setter="SMC_Statemachine/Prop_ControlCommand"
            getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
            expect="Aborted" expectType="System.String" wait="100"/>
      </setup>
      <properties submodel="SM_Unit1_(SuperiorStateMachine)"
          setter="SMC_Statemachine/Prop_ControlCommand"
          getter="SMC_Statemachine/Prop_State"/>
      <state id="Aborted">
        <transition event="Clear" target="Stopped" wait="100"/>
      </state>
      <state id="Stopped">
        <transition event="Abort" target="Aborted" wait="100"/>
      </state>
    </path>
    <path>
      <setup>
        <action submodel="SM_Unit1_(SuperiorStateMachine)"
            setter="SMC_Statemachine/Prop_ControlCommand"
```

```
            getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
            expect="Aborted" expectType="System.String" wait="100"/>
      </setup>
      <properties submodel="SM_Unit1_(SuperiorStateMachine)"
          setter="SMC_Statemachine/Prop_ControlCommand"
          getter="SMC_Statemachine/Prop_State"/>
      <state id="Aborted">
        <transition event="Clear" target="Stopped" wait="100"/>
      </state>
      <state id="Stopped">
        <transition event="Reset" target="Idle" wait="100"/>
      </state>
    </path>
    <path>
      <setup>
        <action submodel="SM_Unit1_(SuperiorStateMachine)"
            setter="SMC_Statemachine/Prop_ControlCommand"
            getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
            expect="Aborted" expectType="System.String" wait="100"/>
      </setup>
      <properties submodel="SM_Unit1_(SuperiorStateMachine)"
          setter="SMC_Statemachine/Prop_ControlCommand"
          getter="SMC_Statemachine/Prop_State"/>
      <state id="Aborted">
        <transition event="Clear" target="Stopped" wait="100"/>
      </state>
      <state id="Stopped">
        <transition event="Reset" target="Idle" wait="100"/>
      </state>
      <state id="Idle">
        <transition event="Abort" target="Aborted" wait="100"/>
      </state>
    </path>
    <path>
      <setup>
        <action submodel="SM_Unit1_(SuperiorStateMachine)"
            setter="SMC_Statemachine/Prop_ControlCommand"
            getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
            expect="Aborted" expectType="System.String" wait="100"/>
      </setup>
      <properties submodel="SM_Unit1_(SuperiorStateMachine)"
          setter="SMC_Statemachine/Prop_ControlCommand"
          getter="SMC_Statemachine/Prop_State"/>
      <state id="Aborted">
        <transition event="Clear" target="Stopped" wait="100"/>
      </state>
      <state id="Stopped">
        <transition event="Reset" target="Idle" wait="100"/>
      </state>
      <state id="Idle">
        <transition event="Stop" target="Stopped" wait="100"/>
      </state>
    </path>
    <path>
      <setup>
        <action submodel="SM_Unit1_(SuperiorStateMachine)"
            setter="SMC_Statemachine/Prop_ControlCommand"
            getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
            expect="Aborted" expectType="System.String" wait="100"/>
```

```
79          </setup>
80          <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
81          <state id="Aborted">
82            <transition event="Clear" target="Stopped" wait="100"/>
83          </state>
84          <state id="Stopped">
85            <transition event="Reset" target="Idle" wait="100"/>
86          </state>
87          <state id="Idle">
88            <transition event="Start" target="Execute" wait="100"/>
89          </state>
90        </path>
91        <path>
92          <setup>
93            <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
94          </setup>
95          <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
96          <state id="Aborted">
97            <transition event="Clear" target="Stopped" wait="100"/>
98          </state>
99          <state id="Stopped">
100           <transition event="Reset" target="Idle" wait="100"/>
101         </state>
102         <state id="Idle">
103           <transition event="Start" target="Execute" wait="100"/>
104         </state>
105         <state id="Execute">
106           <transition event="Abort" target="Aborted" wait="100"/>
107         </state>
108       </path>
109       <path>
110         <setup>
111           <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
112         </setup>
113         <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
114         <state id="Aborted">
115           <transition event="Clear" target="Stopped" wait="100"/>
116         </state>
117         <state id="Stopped">
118           <transition event="Reset" target="Idle" wait="100"/>
119         </state>
120         <state id="Idle">
121           <transition event="Start" target="Execute" wait="100"/>
122         </state>
123         <state id="Execute">
124           <transition event="Stop" target="Stopped" wait="100"/>
125         </state>
```

```
126        </path>
127        <path>
128          <setup>
129            <action submodel="SM_Unit1_(SuperiorStateMachine)"
                   setter="SMC_Statemachine/Prop_ControlCommand"
                   getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                   expect="Aborted" expectType="System.String" wait="100"/>
130          </setup>
131          <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                 setter="SMC_Statemachine/Prop_ControlCommand"
                 getter="SMC_Statemachine/Prop_State"/>
132          <state id="Aborted">
133            <transition event="Clear" target="Stopped" wait="100"/>
134          </state>
135          <state id="Stopped">
136            <transition event="Reset" target="Idle" wait="100"/>
137          </state>
138          <state id="Idle">
139            <transition event="Start" target="Execute" wait="100"/>
140          </state>
141          <state id="Execute">
142            <transition event="Hold" target="Held" wait="100"/>
143          </state>
144        </path>
145        <path>
146          <setup>
147            <action submodel="SM_Unit1_(SuperiorStateMachine)"
                   setter="SMC_Statemachine/Prop_ControlCommand"
                   getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                   expect="Aborted" expectType="System.String" wait="100"/>
148          </setup>
149          <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                 setter="SMC_Statemachine/Prop_ControlCommand"
                 getter="SMC_Statemachine/Prop_State"/>
150          <state id="Aborted">
151            <transition event="Clear" target="Stopped" wait="100"/>
152          </state>
153          <state id="Stopped">
154            <transition event="Reset" target="Idle" wait="100"/>
155          </state>
156          <state id="Idle">
157            <transition event="Start" target="Execute" wait="100"/>
158          </state>
159          <state id="Execute">
160            <transition event="Suspend" target="Suspended" wait="100"/>
161          </state>
162        </path>
163        <path>
164          <setup>
165            <action submodel="SM_Unit1_(SuperiorStateMachine)"
                   setter="SMC_Statemachine/Prop_ControlCommand"
                   getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                   expect="Aborted" expectType="System.String" wait="100"/>
166          </setup>
167          <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                 setter="SMC_Statemachine/Prop_ControlCommand"
                 getter="SMC_Statemachine/Prop_State"/>
168          <state id="Aborted">
169            <transition event="Clear" target="Stopped" wait="100"/>
```

```
170         </state>
171         <state id="Stopped">
172           <transition event="Reset" target="Idle" wait="100"/>
173         </state>
174         <state id="Idle">
175           <transition event="Start" target="Execute" wait="100"/>
176         </state>
177         <state id="Execute">
178           <transition event="Hold" target="Held" wait="100"/>
179         </state>
180         <state id="Held">
181           <transition event="Abort" target="Aborted" wait="100"/>
182         </state>
183       </path>
184       <path>
185         <setup>
186           <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
187         </setup>
188         <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
189         <state id="Aborted">
190           <transition event="Clear" target="Stopped" wait="100"/>
191         </state>
192         <state id="Stopped">
193           <transition event="Reset" target="Idle" wait="100"/>
194         </state>
195         <state id="Idle">
196           <transition event="Start" target="Execute" wait="100"/>
197         </state>
198         <state id="Execute">
199           <transition event="Hold" target="Held" wait="100"/>
200         </state>
201         <state id="Held">
202           <transition event="Stop" target="Stopped" wait="100"/>
203         </state>
204       </path>
205       <path>
206         <setup>
207           <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
208         </setup>
209         <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
210         <state id="Aborted">
211           <transition event="Clear" target="Stopped" wait="100"/>
212         </state>
213         <state id="Stopped">
214           <transition event="Reset" target="Idle" wait="100"/>
215         </state>
216         <state id="Idle">
217           <transition event="Start" target="Execute" wait="100"/>
218         </state>
```

```
219         <state id="Execute">
220           <transition event="Hold" target="Held" wait="100"/>
221         </state>
222         <state id="Held">
223           <transition event="Unhold" target="Execute" wait="100"/>
224         </state>
225       </path>
226       <path>
227         <setup>
228           <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
229         </setup>
230         <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
231         <state id="Aborted">
232           <transition event="Clear" target="Stopped" wait="100"/>
233         </state>
234         <state id="Stopped">
235           <transition event="Reset" target="Idle" wait="100"/>
236         </state>
237         <state id="Idle">
238           <transition event="Start" target="Execute" wait="100"/>
239         </state>
240         <state id="Execute">
241           <transition event="Suspend" target="Suspended" wait="100"/>
242         </state>
243         <state id="Suspended">
244           <transition event="Abort" target="Aborted" wait="100"/>
245         </state>
246       </path>
247       <path>
248         <setup>
249           <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
250         </setup>
251         <properties submodel="SM_Unit1_(SuperiorStateMachine)"
                setter="SMC_Statemachine/Prop_ControlCommand"
                getter="SMC_Statemachine/Prop_State"/>
252         <state id="Aborted">
253           <transition event="Clear" target="Stopped" wait="100"/>
254         </state>
255         <state id="Stopped">
256           <transition event="Reset" target="Idle" wait="100"/>
257         </state>
258         <state id="Idle">
259           <transition event="Start" target="Execute" wait="100"/>
260         </state>
261         <state id="Execute">
262           <transition event="Suspend" target="Suspended" wait="100"/>
263         </state>
264         <state id="Suspended">
265           <transition event="Stop" target="Stopped" wait="100"/>
266         </state>
267       </path>
```

```
268       <path>
269         <setup>
270           <action submodel="SM_Unit1_(SuperiorStateMachine)"
                  setter="SMC_Statemachine/Prop_ControlCommand"
                  getter="SMC_Statemachine/Prop_State" set="8" setType="System.Int32"
                  expect="Aborted" expectType="System.String" wait="100"/>
271         </setup>
272         <properties submodel="SM_Unit1_(SuperiorStateMachine)"
              setter="SMC_Statemachine/Prop_ControlCommand"
              getter="SMC_Statemachine/Prop_State"/>
273         <state id="Aborted">
274           <transition event="Clear" target="Stopped" wait="100"/>
275         </state>
276         <state id="Stopped">
277           <transition event="Reset" target="Idle" wait="100"/>
278         </state>
279         <state id="Idle">
280           <transition event="Start" target="Execute" wait="100"/>
281         </state>
282         <state id="Execute">
283           <transition event="Suspend" target="Suspended" wait="100"/>
284         </state>
285         <state id="Suspended">
286           <transition event="Unsuspend" target="Execute" wait="100"/>
287         </state>
288       </path>
289     </paths>
290     <onetimeteardown>
291       <action submodel="SM_Unit1_(SuperiorStateMachine)"
              setter="SMC_Statemachine/Prop_ControlCommand" getter="SMC_Statemachine/Prop_State"
              set="8" setType="System.Int32" expect="Aborted" expectType="System.String"
              wait="100"/>
292     </onetimeteardown>
293   </descriptor>
```

Listing A.4: *PackML*-basierter Zustandsautomatendeskriptor

```
2022-10-15 00:07:14.522 Expected submodels count: 8
2022-10-15 00:07:14.532 Actual submodels count: 8
2022-10-15 00:07:14.542 Expected IdShort of submodel number 0: SM_Process1
2022-10-15 00:07:14.551 Actual IdShort of submodel number 0: SM_Process1
2022-10-15 00:07:14.560 Expected submodel elements count: 3
2022-10-15 00:07:14.570 Actual submodel elements count: 3
2022-10-15 00:07:14.579 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:14.588 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:14.597 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:14.607 SMC_Statemachine: Expected elements count: 3
2022-10-15 00:07:14.617 SMC_Statemachine: Actual submodel elements count: 3
2022-10-15 00:07:14.626 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:14.635 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:14.644 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:14.653 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:14.663 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:14.671 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:14.680 Expected IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:14.689 Actual IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:14.697 Submodel element SMC_ProcessInfo is an element collection
2022-10-15 00:07:14.706 SMC_ProcessInfo: Expected elements count: 5
2022-10-15 00:07:14.714 SMC_ProcessInfo: Actual submodel elements count: 5
2022-10-15 00:07:14.723 Expected IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:14.733 Actual IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:14.741 Submodel element SMC_ElementOfProcessInfo1 is an element collection
2022-10-15 00:07:14.750 SMC_ElementOfProcessInfo1: Expected elements count: 9
2022-10-15 00:07:14.759 SMC_ElementOfProcessInfo1: Actual submodel elements count: 9
2022-10-15 00:07:14.767 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:14.776 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:14.784 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:14.793 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:14.802 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:14.811 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:14.820 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:14.828 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:14.836 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:14.844 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:14.853 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:14.861 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:14.870 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:14.879 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:14.888 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:14.897 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:14.906 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:14.914 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:14.922 Expected IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:14.931 Actual IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:14.939 Submodel element SMC_ElementOfProcessInfo2 is an element collection
2022-10-15 00:07:14.948 SMC_ElementOfProcessInfo2: Expected elements count: 9
2022-10-15 00:07:14.957 SMC_ElementOfProcessInfo2: Actual submodel elements count: 9
2022-10-15 00:07:14.965 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:14.974 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:14.983 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:14.990 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:14.999 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.007 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.015 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.023 Actual IdShort of submodel element 3: ProcessVariables4
```

```
2022-10-15 00:07:15.031 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.039 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.047 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.056 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.065 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.073 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.081 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.089 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.097 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.106 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.114 Expected IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:15.122 Actual IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:15.130 Submodel element SMC_ElementOfProcessInfo3 is an element collection
2022-10-15 00:07:15.139 SMC_ElementOfProcessInfo3: Expected elements count: 9
2022-10-15 00:07:15.148 SMC_ElementOfProcessInfo3: Actual submodel elements count: 9
2022-10-15 00:07:15.156 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:15.164 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:15.173 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:15.182 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:15.190 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.197 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.204 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.213 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.221 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.229 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.240 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.249 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.257 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.269 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.277 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.285 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.293 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.300 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.307 Expected IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:15.314 Actual IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:15.322 Submodel element SMC_ElementOfProcessInfo4 is an element collection
2022-10-15 00:07:15.329 SMC_ElementOfProcessInfo4: Expected elements count: 9
2022-10-15 00:07:15.336 SMC_ElementOfProcessInfo4: Actual submodel elements count: 9
2022-10-15 00:07:15.344 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:15.352 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:15.360 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:15.367 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:15.375 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.383 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.392 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.400 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.408 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.415 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.423 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.430 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.437 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.445 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.453 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.460 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.468 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.476 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.483 Expected IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:15.490 Actual IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:15.498 Submodel element SMC_ElementOfProcessInfo5 is an element collection
```

```
2022-10-15 00:07:15.506 SMC_ElementOfProcessInfo5: Expected elements count: 9
2022-10-15 00:07:15.513 SMC_ElementOfProcessInfo5: Actual submodel elements count: 9
2022-10-15 00:07:15.522 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:15.530 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:15.537 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:15.544 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:15.552 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.560 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:15.567 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.574 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:15.583 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.591 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:15.599 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.607 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:15.614 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.622 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:15.629 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.636 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:15.645 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.652 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:15.659 Expected IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:15.667 Actual IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:15.674 Submodel element SMC_ProcessController is an element collection
2022-10-15 00:07:15.681 SMC_ProcessController: Expected elements count: 5
2022-10-15 00:07:15.688 SMC_ProcessController: Actual submodel elements count: 5
2022-10-15 00:07:15.696 Expected IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:15.704 Actual IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:15.712 Submodel element SMC_ElementOfProcessController1 is an element
    collection
2022-10-15 00:07:15.720 SMC_ElementOfProcessController1: Expected elements count: 5
2022-10-15 00:07:15.728 SMC_ElementOfProcessController1: Actual submodel elements count: 5
2022-10-15 00:07:15.734 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:15.743 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:15.750 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:15.757 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:15.765 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:15.772 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:15.779 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:15.787 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:15.794 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:15.801 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:15.809 Expected IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:15.816 Actual IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:15.824 Submodel element SMC_ElementOfProcessController2 is an element
    collection
2022-10-15 00:07:15.831 SMC_ElementOfProcessController2: Expected elements count: 5
2022-10-15 00:07:15.839 SMC_ElementOfProcessController2: Actual submodel elements count: 5
2022-10-15 00:07:15.845 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:15.852 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:15.859 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:15.866 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:15.873 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:15.880 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:15.888 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:15.895 Actual IdShort of submodel element 3: Parameter4
```

```
2022-10-15 00:07:15.902 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:15.910 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:15.918 Expected IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:15.925 Actual IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:15.932 Submodel element SMC_ElementOfProcessController3 is an element
    collection
2022-10-15 00:07:15.940 SMC_ElementOfProcessController3: Expected elements count: 5
2022-10-15 00:07:15.948 SMC_ElementOfProcessController3: Actual submodel elements count: 5
2022-10-15 00:07:15.957 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:15.964 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:15.972 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:15.980 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:15.987 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:15.994 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:16.001 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:16.008 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:16.017 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:16.024 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:16.031 Expected IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:16.039 Actual IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:16.046 Submodel element SMC_ElementOfProcessController4 is an element
    collection
2022-10-15 00:07:16.053 SMC_ElementOfProcessController4: Expected elements count: 5
2022-10-15 00:07:16.060 SMC_ElementOfProcessController4: Actual submodel elements count: 5
2022-10-15 00:07:16.067 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:16.075 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:16.082 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:16.090 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:16.098 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:16.105 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:16.112 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:16.120 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:16.127 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:16.135 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:16.142 Expected IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:16.150 Actual IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:16.158 Submodel element SMC_ElementOfProcessController5 is an element
    collection
2022-10-15 00:07:16.166 SMC_ElementOfProcessController5: Expected elements count: 5
2022-10-15 00:07:16.174 SMC_ElementOfProcessController5: Actual submodel elements count: 5
2022-10-15 00:07:16.181 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:16.188 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:16.196 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:16.202 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:16.210 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:16.217 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:16.224 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:16.232 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:16.239 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:16.247 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:16.254 Expected IdShort of submodel number 1: SM_Process2
2022-10-15 00:07:16.262 Actual IdShort of submodel number 1: SM_Process2
2022-10-15 00:07:16.269 Expected submodel elements count: 3
```

```
2022-10-15 00:07:16.277 Actual submodel elements count: 3
2022-10-15 00:07:16.284 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:16.292 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:16.299 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:16.306 SMC_Statemachine: Expected elements count: 3
2022-10-15 00:07:16.314 SMC_Statemachine: Actual submodel elements count: 3
2022-10-15 00:07:16.321 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:16.329 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:16.336 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:16.344 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:16.352 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:16.360 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:16.368 Expected IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:16.376 Actual IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:16.383 Submodel element SMC_ProcessInfo is an element collection
2022-10-15 00:07:16.391 SMC_ProcessInfo: Expected elements count: 5
2022-10-15 00:07:16.398 SMC_ProcessInfo: Actual submodel elements count: 5
2022-10-15 00:07:16.405 Expected IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:16.412 Actual IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:16.419 Submodel element SMC_ElementOfProcessInfo1 is an element collection
2022-10-15 00:07:16.426 SMC_ElementOfProcessInfo1: Expected elements count: 9
2022-10-15 00:07:16.435 SMC_ElementOfProcessInfo1: Actual submodel elements count: 9
2022-10-15 00:07:16.443 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.451 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.457 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.465 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.473 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:16.481 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:16.490 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:16.497 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:16.507 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:16.515 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:16.522 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:16.530 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:16.538 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:16.545 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:16.554 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:16.562 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:16.570 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:16.577 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:16.585 Expected IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:16.592 Actual IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:16.599 Submodel element SMC_ElementOfProcessInfo2 is an element collection
2022-10-15 00:07:16.606 SMC_ElementOfProcessInfo2: Expected elements count: 9
2022-10-15 00:07:16.615 SMC_ElementOfProcessInfo2: Actual submodel elements count: 9
2022-10-15 00:07:16.622 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.630 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.637 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.645 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.652 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:16.659 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:16.667 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:16.675 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:16.681 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:16.690 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:16.697 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:16.704 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:16.713 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:16.720 Actual IdShort of submodel element 6: ProcessVariables7
```

```
2022-10-15 00:07:16.727 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:16.734 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:16.743 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:16.751 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:16.759 Expected IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:16.767 Actual IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:16.775 Submodel element SMC_ElementOfProcessInfo3 is an element collection
2022-10-15 00:07:16.781 SMC_ElementOfProcessInfo3: Expected elements count: 9
2022-10-15 00:07:16.789 SMC_ElementOfProcessInfo3: Actual submodel elements count: 9
2022-10-15 00:07:16.797 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.803 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.812 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.819 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.827 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:16.835 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:16.842 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:16.850 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:16.858 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:16.865 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:16.872 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:16.879 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:16.887 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:16.895 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:16.903 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:16.911 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:16.918 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:16.926 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:16.933 Expected IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:16.941 Actual IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:16.949 Submodel element SMC_ElementOfProcessInfo4 is an element collection
2022-10-15 00:07:16.956 SMC_ElementOfProcessInfo4: Expected elements count: 9
2022-10-15 00:07:16.965 SMC_ElementOfProcessInfo4: Actual submodel elements count: 9
2022-10-15 00:07:16.972 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.979 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:16.987 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:16.996 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:17.003 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:17.011 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:17.018 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:17.026 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:17.034 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:17.041 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:17.048 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:17.055 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:17.063 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:17.070 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:17.077 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:17.085 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:17.094 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:17.101 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:17.109 Expected IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:17.116 Actual IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:17.124 Submodel element SMC_ElementOfProcessInfo5 is an element collection
2022-10-15 00:07:17.132 SMC_ElementOfProcessInfo5: Expected elements count: 9
2022-10-15 00:07:17.139 SMC_ElementOfProcessInfo5: Actual submodel elements count: 9
2022-10-15 00:07:17.146 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:17.154 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:17.162 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:17.170 Actual IdShort of submodel element 1: ProcessVariables2
```

```
2022-10-15 00:07:17.177 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:17.185 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:17.192 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:17.201 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:17.208 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:17.216 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:17.224 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:17.231 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:17.239 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:17.247 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:17.254 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:17.262 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:17.269 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:17.277 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:17.284 Expected IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:17.290 Actual IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:17.298 Submodel element SMC_ProcessController is an element collection
2022-10-15 00:07:17.306 SMC_ProcessController: Expected elements count: 5
2022-10-15 00:07:17.314 SMC_ProcessController: Actual submodel elements count: 5
2022-10-15 00:07:17.322 Expected IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:17.329 Actual IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:17.337 Submodel element SMC_ElementOfProcessController1 is an element
    collection
2022-10-15 00:07:17.344 SMC_ElementOfProcessController1: Expected elements count: 5
2022-10-15 00:07:17.350 SMC_ElementOfProcessController1: Actual submodel elements count: 5
2022-10-15 00:07:17.358 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.366 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.374 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.382 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.389 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.397 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.404 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.411 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.419 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.426 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.433 Expected IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:17.442 Actual IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:17.448 Submodel element SMC_ElementOfProcessController2 is an element
    collection
2022-10-15 00:07:17.456 SMC_ElementOfProcessController2: Expected elements count: 5
2022-10-15 00:07:17.464 SMC_ElementOfProcessController2: Actual submodel elements count: 5
2022-10-15 00:07:17.471 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.479 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.486 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.493 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.502 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.512 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.520 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.528 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.535 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.542 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.550 Expected IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:17.558 Actual IdShort of submodel element 2:
    SMC_ElementOfProcessController3
```

```
2022-10-15 00:07:17.566 Submodel element SMC_ElementOfProcessController3 is an element
    collection
2022-10-15 00:07:17.573 SMC_ElementOfProcessController3: Expected elements count: 5
2022-10-15 00:07:17.582 SMC_ElementOfProcessController3: Actual submodel elements count: 5
2022-10-15 00:07:17.591 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.600 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.606 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.615 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.622 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.630 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.637 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.645 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.653 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.660 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.667 Expected IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:17.674 Actual IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:17.682 Submodel element SMC_ElementOfProcessController4 is an element
    collection
2022-10-15 00:07:17.689 SMC_ElementOfProcessController4: Expected elements count: 5
2022-10-15 00:07:17.698 SMC_ElementOfProcessController4: Actual submodel elements count: 5
2022-10-15 00:07:17.705 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.712 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.720 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.727 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.735 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.742 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.750 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.757 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.764 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.772 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.779 Expected IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:17.787 Actual IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:17.795 Submodel element SMC_ElementOfProcessController5 is an element
    collection
2022-10-15 00:07:17.803 SMC_ElementOfProcessController5: Expected elements count: 5
2022-10-15 00:07:17.810 SMC_ElementOfProcessController5: Actual submodel elements count: 5
2022-10-15 00:07:17.818 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.825 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:17.833 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.840 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:17.847 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.854 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:17.862 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.869 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:17.877 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.884 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:17.891 Expected IdShort of submodel number 2: SM_Process3
2022-10-15 00:07:17.898 Actual IdShort of submodel number 2: SM_Process3
2022-10-15 00:07:17.905 Expected submodel elements count: 3
2022-10-15 00:07:17.913 Actual submodel elements count: 3
2022-10-15 00:07:17.921 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:17.928 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:17.936 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:17.943 SMC_Statemachine: Expected elements count: 3
2022-10-15 00:07:17.951 SMC_Statemachine: Actual submodel elements count: 3
```

```
2022-10-15 00:07:17.958 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:17.972 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:17.987 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:18.002 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:18.011 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:18.021 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:18.029 Expected IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:18.038 Actual IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:18.054 Submodel element SMC_ProcessInfo is an element collection
2022-10-15 00:07:18.066 SMC_ProcessInfo: Expected elements count: 5
2022-10-15 00:07:18.079 SMC_ProcessInfo: Actual submodel elements count: 5
2022-10-15 00:07:18.092 Expected IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:18.106 Actual IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:18.120 Submodel element SMC_ElementOfProcessInfo1 is an element collection
2022-10-15 00:07:18.130 SMC_ElementOfProcessInfo1: Expected elements count: 9
2022-10-15 00:07:18.143 SMC_ElementOfProcessInfo1: Actual submodel elements count: 9
2022-10-15 00:07:18.159 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.170 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.187 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.202 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.218 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.239 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.254 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.268 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.314 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.327 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.345 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.365 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.388 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.400 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.408 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:18.422 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:18.441 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:18.454 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:18.469 Expected IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:18.478 Actual IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:18.485 Submodel element SMC_ElementOfProcessInfo2 is an element collection
2022-10-15 00:07:18.493 SMC_ElementOfProcessInfo2: Expected elements count: 9
2022-10-15 00:07:18.502 SMC_ElementOfProcessInfo2: Actual submodel elements count: 9
2022-10-15 00:07:18.509 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.518 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.525 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.533 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.541 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.549 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.558 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.566 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.574 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.582 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.590 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.601 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.608 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.618 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.628 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:18.636 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:18.645 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:18.653 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:18.661 Expected IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:18.669 Actual IdShort of submodel element 2: SMC_ElementOfProcessInfo3
```

```
2022-10-15 00:07:18.677 Submodel element SMC_ElementOfProcessInfo3 is an element collection
2022-10-15 00:07:18.685 SMC_ElementOfProcessInfo3: Expected elements count: 9
2022-10-15 00:07:18.693 SMC_ElementOfProcessInfo3: Actual submodel elements count: 9
2022-10-15 00:07:18.700 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.708 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.716 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.723 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.732 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.739 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.747 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.754 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.763 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.770 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.779 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.787 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.796 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.803 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.811 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:18.819 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:18.828 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:18.836 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:18.844 Expected IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:18.852 Actual IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:18.860 Submodel element SMC_ElementOfProcessInfo4 is an element collection
2022-10-15 00:07:18.867 SMC_ElementOfProcessInfo4: Expected elements count: 9
2022-10-15 00:07:18.875 SMC_ElementOfProcessInfo4: Actual submodel elements count: 9
2022-10-15 00:07:18.883 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.890 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:18.898 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.905 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:18.913 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.920 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:18.928 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.936 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:18.944 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.953 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:18.962 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.969 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:18.978 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.985 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:18.994 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:19.002 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:19.010 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:19.017 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:19.026 Expected IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:19.035 Actual IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:19.047 Submodel element SMC_ElementOfProcessInfo5 is an element collection
2022-10-15 00:07:19.055 SMC_ElementOfProcessInfo5: Expected elements count: 9
2022-10-15 00:07:19.064 SMC_ElementOfProcessInfo5: Actual submodel elements count: 9
2022-10-15 00:07:19.071 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:19.079 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:19.087 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:19.095 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:19.102 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:19.110 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:19.118 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:19.126 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:19.133 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:19.141 Actual IdShort of submodel element 4: ProcessVariables5
```

```
2022-10-15 00:07:19.148 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:19.156 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:19.166 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:19.173 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:19.182 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:19.189 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:19.197 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:19.204 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:19.213 Expected IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:19.222 Actual IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:19.231 Submodel element SMC_ProcessController is an element collection
2022-10-15 00:07:19.240 SMC_ProcessController: Expected elements count: 5
2022-10-15 00:07:19.251 SMC_ProcessController: Actual submodel elements count: 5
2022-10-15 00:07:19.261 Expected IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:19.269 Actual IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:19.278 Submodel element SMC_ElementOfProcessController1 is an element
    collection
2022-10-15 00:07:19.285 SMC_ElementOfProcessController1: Expected elements count: 5
2022-10-15 00:07:19.294 SMC_ElementOfProcessController1: Actual submodel elements count: 5
2022-10-15 00:07:19.302 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.310 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.318 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.326 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.336 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.346 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.354 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.362 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.370 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.377 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.385 Expected IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:19.392 Actual IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:19.399 Submodel element SMC_ElementOfProcessController2 is an element
    collection
2022-10-15 00:07:19.406 SMC_ElementOfProcessController2: Expected elements count: 5
2022-10-15 00:07:19.414 SMC_ElementOfProcessController2: Actual submodel elements count: 5
2022-10-15 00:07:19.421 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.429 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.437 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.446 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.454 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.463 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.470 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.479 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.487 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.495 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.502 Expected IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:19.511 Actual IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:19.518 Submodel element SMC_ElementOfProcessController3 is an element
    collection
2022-10-15 00:07:19.525 SMC_ElementOfProcessController3: Expected elements count: 5
2022-10-15 00:07:19.534 SMC_ElementOfProcessController3: Actual submodel elements count: 5
2022-10-15 00:07:19.541 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.549 Actual IdShort of submodel element 0: Parameter1
```

```
2022-10-15 00:07:19.557 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.566 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.573 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.581 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.589 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.596 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.604 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.612 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.619 Expected IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:19.626 Actual IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:19.634 Submodel element SMC_ElementOfProcessController4 is an element
    collection
2022-10-15 00:07:19.642 SMC_ElementOfProcessController4: Expected elements count: 5
2022-10-15 00:07:19.650 SMC_ElementOfProcessController4: Actual submodel elements count: 5
2022-10-15 00:07:19.657 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.667 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.674 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.682 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.689 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.697 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.704 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.711 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.719 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.726 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.733 Expected IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:19.740 Actual IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:19.748 Submodel element SMC_ElementOfProcessController5 is an element
    collection
2022-10-15 00:07:19.758 SMC_ElementOfProcessController5: Expected elements count: 5
2022-10-15 00:07:19.767 SMC_ElementOfProcessController5: Actual submodel elements count: 5
2022-10-15 00:07:19.775 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.784 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:19.791 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.799 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:19.806 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.814 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:19.822 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.830 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:19.838 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.846 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:19.853 Expected IdShort of submodel number 3: SM_Process4
2022-10-15 00:07:19.862 Actual IdShort of submodel number 3: SM_Process4
2022-10-15 00:07:19.869 Expected submodel elements count: 3
2022-10-15 00:07:19.879 Actual submodel elements count: 3
2022-10-15 00:07:19.886 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:19.896 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:19.904 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:19.912 SMC_Statemachine: Expected elements count: 3
2022-10-15 00:07:19.920 SMC_Statemachine: Actual submodel elements count: 3
2022-10-15 00:07:19.929 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:19.936 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:19.944 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:19.954 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:19.961 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:19.969 Actual IdShort of submodel element 2: Prop_Running
```

```
2022-10-15 00:07:19.975 Expected IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:19.983 Actual IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:19.990 Submodel element SMC_ProcessInfo is an element collection
2022-10-15 00:07:19.997 SMC_ProcessInfo: Expected elements count: 5
2022-10-15 00:07:20.006 SMC_ProcessInfo: Actual submodel elements count: 5
2022-10-15 00:07:20.013 Expected IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:20.020 Actual IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:20.029 Submodel element SMC_ElementOfProcessInfo1 is an element collection
2022-10-15 00:07:20.036 SMC_ElementOfProcessInfo1: Expected elements count: 9
2022-10-15 00:07:20.044 SMC_ElementOfProcessInfo1: Actual submodel elements count: 9
2022-10-15 00:07:20.052 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.058 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.066 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.075 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.084 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.093 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.101 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.109 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.116 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.124 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.132 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.140 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.147 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.155 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.163 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.170 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.177 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.185 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.193 Expected IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:20.200 Actual IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:20.208 Submodel element SMC_ElementOfProcessInfo2 is an element collection
2022-10-15 00:07:20.215 SMC_ElementOfProcessInfo2: Expected elements count: 9
2022-10-15 00:07:20.222 SMC_ElementOfProcessInfo2: Actual submodel elements count: 9
2022-10-15 00:07:20.230 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.239 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.246 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.254 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.295 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.313 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.323 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.332 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.340 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.347 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.354 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.362 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.369 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.377 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.384 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.392 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.399 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.407 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.415 Expected IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:20.422 Actual IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:20.429 Submodel element SMC_ElementOfProcessInfo3 is an element collection
2022-10-15 00:07:20.436 SMC_ElementOfProcessInfo3: Expected elements count: 9
2022-10-15 00:07:20.443 SMC_ElementOfProcessInfo3: Actual submodel elements count: 9
2022-10-15 00:07:20.451 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.459 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.469 Expected IdShort of submodel element 1: ProcessVariables2
```

```
2022-10-15 00:07:20.477 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.484 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.491 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.498 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.505 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.513 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.524 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.531 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.539 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.548 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.555 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.565 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.572 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.580 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.587 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.594 Expected IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:20.601 Actual IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:20.608 Submodel element SMC_ElementOfProcessInfo4 is an element collection
2022-10-15 00:07:20.616 SMC_ElementOfProcessInfo4: Expected elements count: 9
2022-10-15 00:07:20.623 SMC_ElementOfProcessInfo4: Actual submodel elements count: 9
2022-10-15 00:07:20.630 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.638 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.645 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.653 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.660 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.667 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.673 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.681 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.687 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.695 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.702 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.709 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.716 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.725 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.734 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.741 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.749 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.758 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.765 Expected IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:20.774 Actual IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:20.781 Submodel element SMC_ElementOfProcessInfo5 is an element collection
2022-10-15 00:07:20.789 SMC_ElementOfProcessInfo5: Expected elements count: 9
2022-10-15 00:07:20.797 SMC_ElementOfProcessInfo5: Actual submodel elements count: 9
2022-10-15 00:07:20.804 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.811 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:20.819 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.827 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:20.836 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.844 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:20.851 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.858 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:20.866 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.873 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:20.880 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.888 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:20.896 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.903 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:20.910 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:20.917 Actual IdShort of submodel element 7: ProcessVariables8
```

```
2022-10-15 00:07:20.926 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.934 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:20.942 Expected IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:20.951 Actual IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:20.958 Submodel element SMC_ProcessController is an element collection
2022-10-15 00:07:20.967 SMC_ProcessController: Expected elements count: 5
2022-10-15 00:07:20.975 SMC_ProcessController: Actual submodel elements count: 5
2022-10-15 00:07:20.982 Expected IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:20.990 Actual IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:20.998 Submodel element SMC_ElementOfProcessController1 is an element
    collection
2022-10-15 00:07:21.005 SMC_ElementOfProcessController1: Expected elements count: 5
2022-10-15 00:07:21.013 SMC_ElementOfProcessController1: Actual submodel elements count: 5
2022-10-15 00:07:21.022 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.030 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.038 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.045 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.052 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.061 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.068 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.076 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.083 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.090 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.099 Expected IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:21.106 Actual IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:21.114 Submodel element SMC_ElementOfProcessController2 is an element
    collection
2022-10-15 00:07:21.122 SMC_ElementOfProcessController2: Expected elements count: 5
2022-10-15 00:07:21.129 SMC_ElementOfProcessController2: Actual submodel elements count: 5
2022-10-15 00:07:21.136 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.146 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.155 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.164 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.171 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.180 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.188 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.196 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.203 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.211 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.220 Expected IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:21.228 Actual IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:21.237 Submodel element SMC_ElementOfProcessController3 is an element
    collection
2022-10-15 00:07:21.244 SMC_ElementOfProcessController3: Expected elements count: 5
2022-10-15 00:07:21.252 SMC_ElementOfProcessController3: Actual submodel elements count: 5
2022-10-15 00:07:21.260 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.268 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.274 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.282 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.288 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.296 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.304 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.311 Actual IdShort of submodel element 3: Parameter4
```

```
2022-10-15 00:07:21.318 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.326 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.333 Expected IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:21.341 Actual IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:21.350 Submodel element SMC_ElementOfProcessController4 is an element
    collection
2022-10-15 00:07:21.359 SMC_ElementOfProcessController4: Expected elements count: 5
2022-10-15 00:07:21.369 SMC_ElementOfProcessController4: Actual submodel elements count: 5
2022-10-15 00:07:21.380 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.387 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.400 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.407 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.416 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.423 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.432 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.439 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.447 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.455 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.463 Expected IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:21.471 Actual IdShort of submodel element 4:
    SMC_ElementOfProcessController5
2022-10-15 00:07:21.478 Submodel element SMC_ElementOfProcessController5 is an element
    collection
2022-10-15 00:07:21.486 SMC_ElementOfProcessController5: Expected elements count: 5
2022-10-15 00:07:21.495 SMC_ElementOfProcessController5: Actual submodel elements count: 5
2022-10-15 00:07:21.502 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.510 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:21.518 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.525 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:21.534 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.541 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:21.549 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.556 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:21.563 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.570 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:21.579 Expected IdShort of submodel number 4: SM_Process5
2022-10-15 00:07:21.587 Actual IdShort of submodel number 4: SM_Process5
2022-10-15 00:07:21.595 Expected submodel elements count: 3
2022-10-15 00:07:21.602 Actual submodel elements count: 3
2022-10-15 00:07:21.611 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:21.619 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:21.626 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:21.635 SMC_Statemachine: Expected elements count: 3
2022-10-15 00:07:21.642 SMC_Statemachine: Actual submodel elements count: 3
2022-10-15 00:07:21.650 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:21.658 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:21.665 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:21.673 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:21.680 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:21.687 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:21.695 Expected IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:21.702 Actual IdShort of submodel element 1: SMC_ProcessInfo
2022-10-15 00:07:21.709 Submodel element SMC_ProcessInfo is an element collection
2022-10-15 00:07:21.717 SMC_ProcessInfo: Expected elements count: 5
2022-10-15 00:07:21.723 SMC_ProcessInfo: Actual submodel elements count: 5
2022-10-15 00:07:21.731 Expected IdShort of submodel element 0: SMC_ElementOfProcessInfo1
```

```
2022-10-15 00:07:21.739 Actual IdShort of submodel element 0: SMC_ElementOfProcessInfo1
2022-10-15 00:07:21.746 Submodel element SMC_ElementOfProcessInfo1 is an element collection
2022-10-15 00:07:21.755 SMC_ElementOfProcessInfo1: Expected elements count: 9
2022-10-15 00:07:21.762 SMC_ElementOfProcessInfo1: Actual submodel elements count: 9
2022-10-15 00:07:21.769 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:21.778 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:21.785 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:21.795 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:21.803 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:21.810 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:21.817 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:21.825 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:21.833 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:21.841 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:21.849 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:21.856 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:21.864 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:21.872 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:21.880 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:21.887 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:21.896 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:21.903 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:21.911 Expected IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:21.918 Actual IdShort of submodel element 1: SMC_ElementOfProcessInfo2
2022-10-15 00:07:21.925 Submodel element SMC_ElementOfProcessInfo2 is an element collection
2022-10-15 00:07:21.933 SMC_ElementOfProcessInfo2: Expected elements count: 9
2022-10-15 00:07:21.941 SMC_ElementOfProcessInfo2: Actual submodel elements count: 9
2022-10-15 00:07:21.949 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:21.958 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:21.965 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:21.973 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:21.982 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:21.990 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:21.999 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.007 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.015 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.022 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.031 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.038 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.046 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.054 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.062 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.069 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.077 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.085 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.092 Expected IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:22.100 Actual IdShort of submodel element 2: SMC_ElementOfProcessInfo3
2022-10-15 00:07:22.107 Submodel element SMC_ElementOfProcessInfo3 is an element collection
2022-10-15 00:07:22.114 SMC_ElementOfProcessInfo3: Expected elements count: 9
2022-10-15 00:07:22.122 SMC_ElementOfProcessInfo3: Actual submodel elements count: 9
2022-10-15 00:07:22.129 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:22.137 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:22.145 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:22.153 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:22.161 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:22.169 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:22.183 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.191 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.199 Expected IdShort of submodel element 4: ProcessVariables5
```

```
2022-10-15 00:07:22.207 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.216 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.223 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.230 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.239 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.246 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.253 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.261 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.268 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.276 Expected IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:22.284 Actual IdShort of submodel element 3: SMC_ElementOfProcessInfo4
2022-10-15 00:07:22.291 Submodel element SMC_ElementOfProcessInfo4 is an element collection
2022-10-15 00:07:22.299 SMC_ElementOfProcessInfo4: Expected elements count: 9
2022-10-15 00:07:22.306 SMC_ElementOfProcessInfo4: Actual submodel elements count: 9
2022-10-15 00:07:22.314 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:22.322 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:22.329 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:22.336 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:22.343 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:22.351 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:22.358 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.365 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.373 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.380 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.387 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.398 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.406 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.414 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.421 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.429 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.436 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.443 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.450 Expected IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:22.457 Actual IdShort of submodel element 4: SMC_ElementOfProcessInfo5
2022-10-15 00:07:22.464 Submodel element SMC_ElementOfProcessInfo5 is an element collection
2022-10-15 00:07:22.472 SMC_ElementOfProcessInfo5: Expected elements count: 9
2022-10-15 00:07:22.479 SMC_ElementOfProcessInfo5: Actual submodel elements count: 9
2022-10-15 00:07:22.487 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:22.495 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:22.503 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:22.510 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:22.518 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:22.525 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:22.533 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.540 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:22.547 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.554 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:22.562 Expected IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.569 Actual IdShort of submodel element 5: ProcessVariables6
2022-10-15 00:07:22.577 Expected IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.583 Actual IdShort of submodel element 6: ProcessVariables7
2022-10-15 00:07:22.592 Expected IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.599 Actual IdShort of submodel element 7: ProcessVariables8
2022-10-15 00:07:22.606 Expected IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.613 Actual IdShort of submodel element 8: ProcessVariables9
2022-10-15 00:07:22.621 Expected IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:22.628 Actual IdShort of submodel element 2: SMC_ProcessController
2022-10-15 00:07:22.635 Submodel element SMC_ProcessController is an element collection
2022-10-15 00:07:22.642 SMC_ProcessController: Expected elements count: 5
```

```
2022-10-15 00:07:22.650 SMC_ProcessController: Actual submodel elements count: 5
2022-10-15 00:07:22.658 Expected IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:22.665 Actual IdShort of submodel element 0:
    SMC_ElementOfProcessController1
2022-10-15 00:07:22.672 Submodel element SMC_ElementOfProcessController1 is an element
    collection
2022-10-15 00:07:22.680 SMC_ElementOfProcessController1: Expected elements count: 5
2022-10-15 00:07:22.687 SMC_ElementOfProcessController1: Actual submodel elements count: 5
2022-10-15 00:07:22.695 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:22.702 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:22.711 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:22.717 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:22.725 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:22.734 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:22.741 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:22.749 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:22.756 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:22.764 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:22.773 Expected IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:22.779 Actual IdShort of submodel element 1:
    SMC_ElementOfProcessController2
2022-10-15 00:07:22.788 Submodel element SMC_ElementOfProcessController2 is an element
    collection
2022-10-15 00:07:22.795 SMC_ElementOfProcessController2: Expected elements count: 5
2022-10-15 00:07:22.802 SMC_ElementOfProcessController2: Actual submodel elements count: 5
2022-10-15 00:07:22.809 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:22.817 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:22.826 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:22.833 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:22.841 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:22.849 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:22.856 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:22.865 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:22.873 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:22.880 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:22.888 Expected IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:22.896 Actual IdShort of submodel element 2:
    SMC_ElementOfProcessController3
2022-10-15 00:07:22.903 Submodel element SMC_ElementOfProcessController3 is an element
    collection
2022-10-15 00:07:22.910 SMC_ElementOfProcessController3: Expected elements count: 5
2022-10-15 00:07:22.918 SMC_ElementOfProcessController3: Actual submodel elements count: 5
2022-10-15 00:07:22.925 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:22.933 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:22.940 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:22.948 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:22.956 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:22.964 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:22.972 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:22.979 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:22.987 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:22.994 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:23.001 Expected IdShort of submodel element 3:
    SMC_ElementOfProcessController4
2022-10-15 00:07:23.009 Actual IdShort of submodel element 3:
    SMC_ElementOfProcessController4
```

```
2022-10-15 00:07:23.016 Submodel element SMC_ElementOfProcessController4 is an element collection
2022-10-15 00:07:23.024 SMC_ElementOfProcessController4: Expected elements count: 5
2022-10-15 00:07:23.032 SMC_ElementOfProcessController4: Actual submodel elements count: 5
2022-10-15 00:07:23.040 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:23.048 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:23.054 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:23.062 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:23.069 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:23.076 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:23.083 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:23.090 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:23.097 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:23.104 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:23.112 Expected IdShort of submodel element 4: SMC_ElementOfProcessController5
2022-10-15 00:07:23.119 Actual IdShort of submodel element 4: SMC_ElementOfProcessController5
2022-10-15 00:07:23.127 Submodel element SMC_ElementOfProcessController5 is an element collection
2022-10-15 00:07:23.134 SMC_ElementOfProcessController5: Expected elements count: 5
2022-10-15 00:07:23.142 SMC_ElementOfProcessController5: Actual submodel elements count: 5
2022-10-15 00:07:23.150 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:23.157 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:23.165 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:23.172 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:23.179 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:23.187 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:23.194 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:23.202 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:23.209 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:23.217 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:23.225 Expected IdShort of submodel number 5: SM_Unit1_(SuperiorStateMachine)
2022-10-15 00:07:23.232 Actual IdShort of submodel number 5: SM_Unit1_(SuperiorStateMachine)
2022-10-15 00:07:23.239 Expected submodel elements count: 3
2022-10-15 00:07:23.246 Actual submodel elements count: 3
2022-10-15 00:07:23.254 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:23.262 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:23.268 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:23.277 SMC_Statemachine: Expected elements count: 5
2022-10-15 00:07:23.283 SMC_Statemachine: Actual submodel elements count: 5
2022-10-15 00:07:23.291 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:23.299 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:23.306 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:23.314 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:23.322 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:23.329 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:23.336 Expected IdShort of submodel element 3: Prop_ControlCommand
2022-10-15 00:07:23.344 Actual IdShort of submodel element 3: Prop_ControlCommand
2022-10-15 00:07:23.352 Expected IdShort of submodel element 4: Prop_UnitMode
2022-10-15 00:07:23.359 Actual IdShort of submodel element 4: Prop_UnitMode
2022-10-15 00:07:23.366 Expected IdShort of submodel element 1: SMC_aUnitInfo
2022-10-15 00:07:23.374 Actual IdShort of submodel element 1: SMC_aUnitInfo
2022-10-15 00:07:23.382 Submodel element SMC_aUnitInfo is an element collection
2022-10-15 00:07:23.390 SMC_aUnitInfo: Expected elements count: 5
2022-10-15 00:07:23.397 SMC_aUnitInfo: Actual submodel elements count: 5
2022-10-15 00:07:23.404 Expected IdShort of submodel element 0: SMC_ElementOfUnitInfo1
2022-10-15 00:07:23.412 Actual IdShort of submodel element 0: SMC_ElementOfUnitInfo1
```

```
2022-10-15 00:07:23.419 Submodel element SMC_ElementOfUnitInfo1 is an element collection
2022-10-15 00:07:23.426 SMC_ElementOfUnitInfo1: Expected elements count: 5
2022-10-15 00:07:23.434 SMC_ElementOfUnitInfo1: Actual submodel elements count: 5
2022-10-15 00:07:23.441 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.449 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.457 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.466 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.473 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.482 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.489 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.499 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.506 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.514 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.521 Expected IdShort of submodel element 1: SMC_ElementOfUnitInfo2
2022-10-15 00:07:23.529 Actual IdShort of submodel element 1: SMC_ElementOfUnitInfo2
2022-10-15 00:07:23.536 Submodel element SMC_ElementOfUnitInfo2 is an element collection
2022-10-15 00:07:23.545 SMC_ElementOfUnitInfo2: Expected elements count: 5
2022-10-15 00:07:23.552 SMC_ElementOfUnitInfo2: Actual submodel elements count: 5
2022-10-15 00:07:23.561 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.570 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.578 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.585 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.592 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.600 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.607 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.615 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.622 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.629 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.636 Expected IdShort of submodel element 2: SMC_ElementOfUnitInfo3
2022-10-15 00:07:23.644 Actual IdShort of submodel element 2: SMC_ElementOfUnitInfo3
2022-10-15 00:07:23.651 Submodel element SMC_ElementOfUnitInfo3 is an element collection
2022-10-15 00:07:23.659 SMC_ElementOfUnitInfo3: Expected elements count: 5
2022-10-15 00:07:23.667 SMC_ElementOfUnitInfo3: Actual submodel elements count: 5
2022-10-15 00:07:23.676 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.684 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.691 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.701 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.709 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.716 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.724 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.732 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.740 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.748 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.755 Expected IdShort of submodel element 3: SMC_ElementOfUnitInfo4
2022-10-15 00:07:23.763 Actual IdShort of submodel element 3: SMC_ElementOfUnitInfo4
2022-10-15 00:07:23.771 Submodel element SMC_ElementOfUnitInfo4 is an element collection
2022-10-15 00:07:23.779 SMC_ElementOfUnitInfo4: Expected elements count: 5
2022-10-15 00:07:23.787 SMC_ElementOfUnitInfo4: Actual submodel elements count: 5
2022-10-15 00:07:23.794 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.802 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.810 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.818 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.826 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.833 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.841 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.849 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.856 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.864 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.871 Expected IdShort of submodel element 4: SMC_ElementOfUnitInfo5
```

```
2022-10-15 00:07:23.880 Actual IdShort of submodel element 4: SMC_ElementOfUnitInfo5
2022-10-15 00:07:23.888 Submodel element SMC_ElementOfUnitInfo5 is an element collection
2022-10-15 00:07:23.897 SMC_ElementOfUnitInfo5: Expected elements count: 5
2022-10-15 00:07:23.905 SMC_ElementOfUnitInfo5: Actual submodel elements count: 5
2022-10-15 00:07:23.913 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.921 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:23.929 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.936 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:23.944 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.952 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:23.959 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.967 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:23.974 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.982 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:23.990 Expected IdShort of submodel element 2: SMC_aUnitController
2022-10-15 00:07:23.997 Actual IdShort of submodel element 2: SMC_aUnitController
2022-10-15 00:07:24.005 Submodel element SMC_aUnitController is an element collection
2022-10-15 00:07:24.012 SMC_aUnitController: Expected elements count: 5
2022-10-15 00:07:24.019 SMC_aUnitController: Actual submodel elements count: 5
2022-10-15 00:07:24.027 Expected IdShort of submodel element 0: SMC_ElementOfUnitController1
2022-10-15 00:07:24.034 Actual IdShort of submodel element 0: SMC_ElementOfUnitController1
2022-10-15 00:07:24.041 Submodel element SMC_ElementOfUnitController1 is an element
    collection
2022-10-15 00:07:24.048 SMC_ElementOfUnitController1: Expected elements count: 5
2022-10-15 00:07:24.055 SMC_ElementOfUnitController1: Actual submodel elements count: 5
2022-10-15 00:07:24.063 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.070 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.078 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.087 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.095 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.103 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.111 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.118 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.125 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.132 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.139 Expected IdShort of submodel element 1: SMC_ElementOfUnitController2
2022-10-15 00:07:24.148 Actual IdShort of submodel element 1: SMC_ElementOfUnitController2
2022-10-15 00:07:24.154 Submodel element SMC_ElementOfUnitController2 is an element
    collection
2022-10-15 00:07:24.161 SMC_ElementOfUnitController2: Expected elements count: 5
2022-10-15 00:07:24.168 SMC_ElementOfUnitController2: Actual submodel elements count: 5
2022-10-15 00:07:24.175 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.182 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.189 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.196 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.204 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.210 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.217 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.224 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.231 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.239 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.246 Expected IdShort of submodel element 2: SMC_ElementOfUnitController3
2022-10-15 00:07:24.254 Actual IdShort of submodel element 2: SMC_ElementOfUnitController3
2022-10-15 00:07:24.261 Submodel element SMC_ElementOfUnitController3 is an element
    collection
2022-10-15 00:07:24.268 SMC_ElementOfUnitController3: Expected elements count: 5
2022-10-15 00:07:24.276 SMC_ElementOfUnitController3: Actual submodel elements count: 5
2022-10-15 00:07:24.284 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.292 Actual IdShort of submodel element 0: Parameter1
```

```
2022-10-15 00:07:24.301 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.307 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.315 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.323 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.329 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.337 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.344 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.351 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.358 Expected IdShort of submodel element 3: SMC_ElementOfUnitController4
2022-10-15 00:07:24.365 Actual IdShort of submodel element 3: SMC_ElementOfUnitController4
2022-10-15 00:07:24.374 Submodel element SMC_ElementOfUnitController4 is an element
    collection
2022-10-15 00:07:24.381 SMC_ElementOfUnitController4: Expected elements count: 5
2022-10-15 00:07:24.388 SMC_ElementOfUnitController4: Actual submodel elements count: 5
2022-10-15 00:07:24.395 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.403 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.410 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.417 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.424 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.432 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.439 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.447 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.455 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.462 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.470 Expected IdShort of submodel element 4: SMC_ElementOfUnitController5
2022-10-15 00:07:24.478 Actual IdShort of submodel element 4: SMC_ElementOfUnitController5
2022-10-15 00:07:24.485 Submodel element SMC_ElementOfUnitController5 is an element
    collection
2022-10-15 00:07:24.493 SMC_ElementOfUnitController5: Expected elements count: 5
2022-10-15 00:07:24.500 SMC_ElementOfUnitController5: Actual submodel elements count: 5
2022-10-15 00:07:24.508 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.515 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:24.522 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.528 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:24.535 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.543 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:24.551 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.557 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:24.566 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.573 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:24.580 Expected IdShort of submodel number 6: SM_Unit2_(SubUnit1)
2022-10-15 00:07:24.588 Actual IdShort of submodel number 6: SM_Unit2_(SubUnit1)
2022-10-15 00:07:24.595 Expected submodel elements count: 3
2022-10-15 00:07:24.602 Actual submodel elements count: 3
2022-10-15 00:07:24.611 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:24.618 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:24.624 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:24.632 SMC_Statemachine: Expected elements count: 5
2022-10-15 00:07:24.639 SMC_Statemachine: Actual submodel elements count: 5
2022-10-15 00:07:24.647 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:24.654 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:24.662 Expected IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:24.670 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:24.677 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:24.685 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:24.693 Expected IdShort of submodel element 3: Prop_UnitSelect
2022-10-15 00:07:24.701 Actual IdShort of submodel element 3: Prop_UnitSelect
2022-10-15 00:07:24.708 Expected IdShort of submodel element 4: Prop_eCommand
2022-10-15 00:07:24.716 Actual IdShort of submodel element 4: Prop_eCommand
```

```
2022-10-15 00:07:24.723 Expected IdShort of submodel element 1: SMC_aUnitInfo
2022-10-15 00:07:24.730 Actual IdShort of submodel element 1: SMC_aUnitInfo
2022-10-15 00:07:24.737 Submodel element SMC_aUnitInfo is an element collection
2022-10-15 00:07:24.744 SMC_aUnitInfo: Expected elements count: 5
2022-10-15 00:07:24.751 SMC_aUnitInfo: Actual submodel elements count: 5
2022-10-15 00:07:24.758 Expected IdShort of submodel element 0: SMC_ElementOfUnitInfo1
2022-10-15 00:07:24.766 Actual IdShort of submodel element 0: SMC_ElementOfUnitInfo1
2022-10-15 00:07:24.773 Submodel element SMC_ElementOfUnitInfo1 is an element collection
2022-10-15 00:07:24.781 SMC_ElementOfUnitInfo1: Expected elements count: 5
2022-10-15 00:07:24.788 SMC_ElementOfUnitInfo1: Actual submodel elements count: 5
2022-10-15 00:07:24.795 Expected IdShort of submodel element 0: Axis_Velocity
2022-10-15 00:07:24.802 Actual IdShort of submodel element 0: Axis_Velocity
2022-10-15 00:07:24.810 Expected IdShort of submodel element 1: Axis_Actual_Position
2022-10-15 00:07:24.817 Actual IdShort of submodel element 1: Axis_Actual_Position
2022-10-15 00:07:24.825 Expected IdShort of submodel element 2: Axis_State
2022-10-15 00:07:24.833 Actual IdShort of submodel element 2: Axis_State
2022-10-15 00:07:24.840 Expected IdShort of submodel element 3: Axis_Target_Position
2022-10-15 00:07:24.847 Actual IdShort of submodel element 3: Axis_Target_Position
2022-10-15 00:07:24.856 Expected IdShort of submodel element 4: Axis_Position_Start
2022-10-15 00:07:24.864 Actual IdShort of submodel element 4: Axis_Position_Start
2022-10-15 00:07:24.870 Expected IdShort of submodel element 1: SMC_ElementOfUnitInfo2
2022-10-15 00:07:24.878 Actual IdShort of submodel element 1: SMC_ElementOfUnitInfo2
2022-10-15 00:07:24.886 Submodel element SMC_ElementOfUnitInfo2 is an element collection
2022-10-15 00:07:24.894 SMC_ElementOfUnitInfo2: Expected elements count: 5
2022-10-15 00:07:24.902 SMC_ElementOfUnitInfo2: Actual submodel elements count: 5
2022-10-15 00:07:24.909 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:24.917 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:24.924 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:24.931 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:24.938 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:24.946 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:24.953 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:24.961 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:24.968 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:24.976 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:24.984 Expected IdShort of submodel element 2: SMC_ElementOfUnitInfo3
2022-10-15 00:07:24.991 Actual IdShort of submodel element 2: SMC_ElementOfUnitInfo3
2022-10-15 00:07:24.998 Submodel element SMC_ElementOfUnitInfo3 is an element collection
2022-10-15 00:07:25.005 SMC_ElementOfUnitInfo3: Expected elements count: 5
2022-10-15 00:07:25.013 SMC_ElementOfUnitInfo3: Actual submodel elements count: 5
2022-10-15 00:07:25.021 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:25.028 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:25.036 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:25.044 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:25.051 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:25.059 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:25.065 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:25.073 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:25.081 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:25.088 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:25.096 Expected IdShort of submodel element 3: SMC_ElementOfUnitInfo4
2022-10-15 00:07:25.104 Actual IdShort of submodel element 3: SMC_ElementOfUnitInfo4
2022-10-15 00:07:25.111 Submodel element SMC_ElementOfUnitInfo4 is an element collection
2022-10-15 00:07:25.118 SMC_ElementOfUnitInfo4: Expected elements count: 5
2022-10-15 00:07:25.126 SMC_ElementOfUnitInfo4: Actual submodel elements count: 5
2022-10-15 00:07:25.134 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:25.141 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:25.149 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:25.157 Actual IdShort of submodel element 1: ProcessVariables2
```

```
2022-10-15 00:07:25.164 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:25.172 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:25.179 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:25.186 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:25.194 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:25.201 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:25.207 Expected IdShort of submodel element 4: SMC_ElementOfUnitInfo5
2022-10-15 00:07:25.215 Actual IdShort of submodel element 4: SMC_ElementOfUnitInfo5
2022-10-15 00:07:25.224 Submodel element SMC_ElementOfUnitInfo5 is an element collection
2022-10-15 00:07:25.230 SMC_ElementOfUnitInfo5: Expected elements count: 5
2022-10-15 00:07:25.239 SMC_ElementOfUnitInfo5: Actual submodel elements count: 5
2022-10-15 00:07:25.246 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:25.253 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:25.261 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:25.268 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:25.276 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:25.283 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:25.290 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:25.297 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:25.304 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:25.311 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:25.319 Expected IdShort of submodel element 2: SMC_aUnitController
2022-10-15 00:07:25.327 Actual IdShort of submodel element 2: SMC_aUnitController
2022-10-15 00:07:25.334 Submodel element SMC_aUnitController is an element collection
2022-10-15 00:07:25.342 SMC_aUnitController: Expected elements count: 5
2022-10-15 00:07:25.349 SMC_aUnitController: Actual submodel elements count: 5
2022-10-15 00:07:25.357 Expected IdShort of submodel element 0: SMC_ElementOfUnitController1
2022-10-15 00:07:25.366 Actual IdShort of submodel element 0: SMC_ElementOfUnitController1
2022-10-15 00:07:25.372 Submodel element SMC_ElementOfUnitController1 is an element
    collection
2022-10-15 00:07:25.380 SMC_ElementOfUnitController1: Expected elements count: 5
2022-10-15 00:07:25.387 SMC_ElementOfUnitController1: Actual submodel elements count: 5
2022-10-15 00:07:25.394 Expected IdShort of submodel element 0: lenght
2022-10-15 00:07:25.400 Actual IdShort of submodel element 0: lenght
2022-10-15 00:07:25.407 Expected IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.415 Actual IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.422 Expected IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.430 Actual IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.437 Expected IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.445 Actual IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.452 Expected IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.460 Actual IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.467 Expected IdShort of submodel element 1: SMC_ElementOfUnitController2
2022-10-15 00:07:25.474 Actual IdShort of submodel element 1: SMC_ElementOfUnitController2
2022-10-15 00:07:25.480 Submodel element SMC_ElementOfUnitController2 is an element
    collection
2022-10-15 00:07:25.488 SMC_ElementOfUnitController2: Expected elements count: 5
2022-10-15 00:07:25.496 SMC_ElementOfUnitController2: Actual submodel elements count: 5
2022-10-15 00:07:25.504 Expected IdShort of submodel element 0: lenght
2022-10-15 00:07:25.511 Actual IdShort of submodel element 0: lenght
2022-10-15 00:07:25.518 Expected IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.525 Actual IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.532 Expected IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.539 Actual IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.548 Expected IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.555 Actual IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.562 Expected IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.570 Actual IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.577 Expected IdShort of submodel element 2: SMC_ElementOfUnitController3
```

```
2022-10-15 00:07:25.585 Actual IdShort of submodel element 2: SMC_ElementOfUnitController3
2022-10-15 00:07:25.594 Submodel element SMC_ElementOfUnitController3 is an element
    collection
2022-10-15 00:07:25.600 SMC_ElementOfUnitController3: Expected elements count: 5
2022-10-15 00:07:25.608 SMC_ElementOfUnitController3: Actual submodel elements count: 5
2022-10-15 00:07:25.616 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:25.623 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:25.631 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:25.639 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:25.645 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:25.654 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:25.661 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:25.668 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:25.675 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:25.683 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:25.689 Expected IdShort of submodel element 3: SMC_ElementOfUnitController4
2022-10-15 00:07:25.697 Actual IdShort of submodel element 3: SMC_ElementOfUnitController4
2022-10-15 00:07:25.704 Submodel element SMC_ElementOfUnitController4 is an element
    collection
2022-10-15 00:07:25.711 SMC_ElementOfUnitController4: Expected elements count: 5
2022-10-15 00:07:25.719 SMC_ElementOfUnitController4: Actual submodel elements count: 5
2022-10-15 00:07:25.726 Expected IdShort of submodel element 0: lenght
2022-10-15 00:07:25.734 Actual IdShort of submodel element 0: lenght
2022-10-15 00:07:25.741 Expected IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.748 Actual IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.756 Expected IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.765 Actual IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.771 Expected IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.779 Actual IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.786 Expected IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.793 Actual IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.800 Expected IdShort of submodel element 4: SMC_ElementOfUnitController5
2022-10-15 00:07:25.807 Actual IdShort of submodel element 4: SMC_ElementOfUnitController5
2022-10-15 00:07:25.815 Submodel element SMC_ElementOfUnitController5 is an element
    collection
2022-10-15 00:07:25.822 SMC_ElementOfUnitController5: Expected elements count: 5
2022-10-15 00:07:25.830 SMC_ElementOfUnitController5: Actual submodel elements count: 5
2022-10-15 00:07:25.837 Expected IdShort of submodel element 0: lenght
2022-10-15 00:07:25.844 Actual IdShort of submodel element 0: lenght
2022-10-15 00:07:25.851 Expected IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.858 Actual IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:25.866 Expected IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.872 Actual IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:25.880 Expected IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.887 Actual IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:25.894 Expected IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.902 Actual IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:25.908 Expected IdShort of submodel number 7: SM_Unit3_(SubUnit2)
2022-10-15 00:07:25.917 Actual IdShort of submodel number 7: SM_Unit3_(SubUnit2)
2022-10-15 00:07:25.924 Expected submodel elements count: 3
2022-10-15 00:07:25.931 Actual submodel elements count: 3
2022-10-15 00:07:25.938 Expected IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:25.946 Actual IdShort of submodel element 0: SMC_Statemachine
2022-10-15 00:07:25.956 Submodel element SMC_Statemachine is an element collection
2022-10-15 00:07:25.965 SMC_Statemachine: Expected elements count: 5
2022-10-15 00:07:25.972 SMC_Statemachine: Actual submodel elements count: 5
2022-10-15 00:07:25.980 Expected IdShort of submodel element 0: Prop_State
2022-10-15 00:07:25.987 Actual IdShort of submodel element 0: Prop_State
2022-10-15 00:07:25.995 Expected IdShort of submodel element 1: Prop_Mode
```

```
2022-10-15 00:07:26.003 Actual IdShort of submodel element 1: Prop_Mode
2022-10-15 00:07:26.011 Expected IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:26.018 Actual IdShort of submodel element 2: Prop_Running
2022-10-15 00:07:26.028 Expected IdShort of submodel element 3: Prop_UnitSelect
2022-10-15 00:07:26.035 Actual IdShort of submodel element 3: Prop_UnitSelect
2022-10-15 00:07:26.042 Expected IdShort of submodel element 4: Prop_eCommand
2022-10-15 00:07:26.050 Actual IdShort of submodel element 4: Prop_eCommand
2022-10-15 00:07:26.058 Expected IdShort of submodel element 1: SMC_aUnitInfo
2022-10-15 00:07:26.066 Actual IdShort of submodel element 1: SMC_aUnitInfo
2022-10-15 00:07:26.074 Submodel element SMC_aUnitInfo is an element collection
2022-10-15 00:07:26.081 SMC_aUnitInfo: Expected elements count: 5
2022-10-15 00:07:26.088 SMC_aUnitInfo: Actual submodel elements count: 5
2022-10-15 00:07:26.095 Expected IdShort of submodel element 0: SMC_ElementOfUnitInfo1
2022-10-15 00:07:26.102 Actual IdShort of submodel element 0: SMC_ElementOfUnitInfo1
2022-10-15 00:07:26.109 Submodel element SMC_ElementOfUnitInfo1 is an element collection
2022-10-15 00:07:26.116 SMC_ElementOfUnitInfo1: Expected elements count: 5
2022-10-15 00:07:26.123 SMC_ElementOfUnitInfo1: Actual submodel elements count: 5
2022-10-15 00:07:26.132 Expected IdShort of submodel element 0: Axis_Velocity
2022-10-15 00:07:26.139 Actual IdShort of submodel element 0: Axis_Velocity
2022-10-15 00:07:26.146 Expected IdShort of submodel element 1: Axis_Actual_Position
2022-10-15 00:07:26.155 Actual IdShort of submodel element 1: Axis_Actual_Position
2022-10-15 00:07:26.163 Expected IdShort of submodel element 2: Axis_State
2022-10-15 00:07:26.170 Actual IdShort of submodel element 2: Axis_State
2022-10-15 00:07:26.179 Expected IdShort of submodel element 3: Axis_Target_Position
2022-10-15 00:07:26.187 Actual IdShort of submodel element 3: Axis_Target_Position
2022-10-15 00:07:26.194 Expected IdShort of submodel element 4: Axis_Position_Start
2022-10-15 00:07:26.201 Actual IdShort of submodel element 4: Axis_Position_Start
2022-10-15 00:07:26.208 Expected IdShort of submodel element 1: SMC_ElementOfUnitInfo2
2022-10-15 00:07:26.216 Actual IdShort of submodel element 1: SMC_ElementOfUnitInfo2
2022-10-15 00:07:26.223 Submodel element SMC_ElementOfUnitInfo2 is an element collection
2022-10-15 00:07:26.230 SMC_ElementOfUnitInfo2: Expected elements count: 5
2022-10-15 00:07:26.237 SMC_ElementOfUnitInfo2: Actual submodel elements count: 5
2022-10-15 00:07:26.244 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.251 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.259 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.266 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.273 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.279 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.286 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.293 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.300 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.307 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.314 Expected IdShort of submodel element 2: SMC_ElementOfUnitInfo3
2022-10-15 00:07:26.321 Actual IdShort of submodel element 2: SMC_ElementOfUnitInfo3
2022-10-15 00:07:26.328 Submodel element SMC_ElementOfUnitInfo3 is an element collection
2022-10-15 00:07:26.335 SMC_ElementOfUnitInfo3: Expected elements count: 5
2022-10-15 00:07:26.342 SMC_ElementOfUnitInfo3: Actual submodel elements count: 5
2022-10-15 00:07:26.350 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.357 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.367 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.378 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.385 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.393 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.401 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.409 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.415 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.422 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.430 Expected IdShort of submodel element 3: SMC_ElementOfUnitInfo4
2022-10-15 00:07:26.438 Actual IdShort of submodel element 3: SMC_ElementOfUnitInfo4
```

```
2022-10-15 00:07:26.445 Submodel element SMC_ElementOfUnitInfo4 is an element collection
2022-10-15 00:07:26.453 SMC_ElementOfUnitInfo4: Expected elements count: 5
2022-10-15 00:07:26.460 SMC_ElementOfUnitInfo4: Actual submodel elements count: 5
2022-10-15 00:07:26.468 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.476 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.482 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.490 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.498 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.505 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.512 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.520 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.527 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.535 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.543 Expected IdShort of submodel element 4: SMC_ElementOfUnitInfo5
2022-10-15 00:07:26.549 Actual IdShort of submodel element 4: SMC_ElementOfUnitInfo5
2022-10-15 00:07:26.556 Submodel element SMC_ElementOfUnitInfo5 is an element collection
2022-10-15 00:07:26.564 SMC_ElementOfUnitInfo5: Expected elements count: 5
2022-10-15 00:07:26.571 SMC_ElementOfUnitInfo5: Actual submodel elements count: 5
2022-10-15 00:07:26.580 Expected IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.587 Actual IdShort of submodel element 0: ProcessVariables1
2022-10-15 00:07:26.594 Expected IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.602 Actual IdShort of submodel element 1: ProcessVariables2
2022-10-15 00:07:26.609 Expected IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.617 Actual IdShort of submodel element 2: ProcessVariables3
2022-10-15 00:07:26.625 Expected IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.632 Actual IdShort of submodel element 3: ProcessVariables4
2022-10-15 00:07:26.639 Expected IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.647 Actual IdShort of submodel element 4: ProcessVariables5
2022-10-15 00:07:26.654 Expected IdShort of submodel element 2: SMC_aUnitController
2022-10-15 00:07:26.662 Actual IdShort of submodel element 2: SMC_aUnitController
2022-10-15 00:07:26.669 Submodel element SMC_aUnitController is an element collection
2022-10-15 00:07:26.676 SMC_aUnitController: Expected elements count: 5
2022-10-15 00:07:26.684 SMC_aUnitController: Actual submodel elements count: 5
2022-10-15 00:07:26.691 Expected IdShort of submodel element 0: SMC_ElementOfUnitController1
2022-10-15 00:07:26.698 Actual IdShort of submodel element 0: SMC_ElementOfUnitController1
2022-10-15 00:07:26.706 Submodel element SMC_ElementOfUnitController1 is an element
    collection
2022-10-15 00:07:26.713 SMC_ElementOfUnitController1: Expected elements count: 5
2022-10-15 00:07:26.721 SMC_ElementOfUnitController1: Actual submodel elements count: 5
2022-10-15 00:07:26.728 Expected IdShort of submodel element 0: lenght
2022-10-15 00:07:26.734 Actual IdShort of submodel element 0: lenght
2022-10-15 00:07:26.743 Expected IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:26.750 Actual IdShort of submodel element 1: Axis_Velocity
2022-10-15 00:07:26.757 Expected IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:26.764 Actual IdShort of submodel element 2: Axis_Acceleration
2022-10-15 00:07:26.771 Expected IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:26.778 Actual IdShort of submodel element 3: Axis_Deceleration
2022-10-15 00:07:26.785 Expected IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:26.793 Actual IdShort of submodel element 4: Axis_Target_Position
2022-10-15 00:07:26.800 Expected IdShort of submodel element 1: SMC_ElementOfUnitController2
2022-10-15 00:07:26.808 Actual IdShort of submodel element 1: SMC_ElementOfUnitController2
2022-10-15 00:07:26.815 Submodel element SMC_ElementOfUnitController2 is an element
    collection
2022-10-15 00:07:26.823 SMC_ElementOfUnitController2: Expected elements count: 5
2022-10-15 00:07:26.831 SMC_ElementOfUnitController2: Actual submodel elements count: 5
2022-10-15 00:07:26.838 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:26.845 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:26.853 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:26.860 Actual IdShort of submodel element 1: Parameter2
```

```
2022-10-15 00:07:26.867 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:26.875 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:26.882 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:26.889 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:26.896 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:26.903 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:26.910 Expected IdShort of submodel element 2: SMC_ElementOfUnitController3
2022-10-15 00:07:26.917 Actual IdShort of submodel element 2: SMC_ElementOfUnitController3
2022-10-15 00:07:26.923 Submodel element SMC_ElementOfUnitController3 is an element
    collection
2022-10-15 00:07:26.930 SMC_ElementOfUnitController3: Expected elements count: 5
2022-10-15 00:07:26.937 SMC_ElementOfUnitController3: Actual submodel elements count: 5
2022-10-15 00:07:26.944 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:26.950 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:26.957 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:26.964 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:26.970 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:26.977 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:26.984 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:26.992 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:26.999 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:27.007 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:27.015 Expected IdShort of submodel element 3: SMC_ElementOfUnitController4
2022-10-15 00:07:27.022 Actual IdShort of submodel element 3: SMC_ElementOfUnitController4
2022-10-15 00:07:27.029 Submodel element SMC_ElementOfUnitController4 is an element
    collection
2022-10-15 00:07:27.036 SMC_ElementOfUnitController4: Expected elements count: 5
2022-10-15 00:07:27.043 SMC_ElementOfUnitController4: Actual submodel elements count: 5
2022-10-15 00:07:27.051 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:27.058 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:27.065 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:27.073 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:27.080 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:27.088 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:27.097 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:27.104 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:27.111 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:27.119 Actual IdShort of submodel element 4: Parameter5
2022-10-15 00:07:27.126 Expected IdShort of submodel element 4: SMC_ElementOfUnitController5
2022-10-15 00:07:27.133 Actual IdShort of submodel element 4: SMC_ElementOfUnitController5
2022-10-15 00:07:27.141 Submodel element SMC_ElementOfUnitController5 is an element
    collection
2022-10-15 00:07:27.148 SMC_ElementOfUnitController5: Expected elements count: 5
2022-10-15 00:07:27.155 SMC_ElementOfUnitController5: Actual submodel elements count: 5
2022-10-15 00:07:27.163 Expected IdShort of submodel element 0: Parameter1
2022-10-15 00:07:27.170 Actual IdShort of submodel element 0: Parameter1
2022-10-15 00:07:27.178 Expected IdShort of submodel element 1: Parameter2
2022-10-15 00:07:27.186 Actual IdShort of submodel element 1: Parameter2
2022-10-15 00:07:27.193 Expected IdShort of submodel element 2: Parameter3
2022-10-15 00:07:27.201 Actual IdShort of submodel element 2: Parameter3
2022-10-15 00:07:27.209 Expected IdShort of submodel element 3: Parameter4
2022-10-15 00:07:27.216 Actual IdShort of submodel element 3: Parameter4
2022-10-15 00:07:27.224 Expected IdShort of submodel element 4: Parameter5
2022-10-15 00:07:27.231 Actual IdShort of submodel element 4: Parameter5

2022-10-15 00:07:27.247 Test passed!
```

Listing A.5: Beispielhafter Protokoll als Ergebnis einer Strukturtestung

```
2022-10-15 00:07:28.171 Executing one time setup actions...
2022-10-15 00:07:28.181 Setting property SMC_Statemachine/Prop_UnitMode of submodel
    SM_Unit1_(SuperiorStateMachine) to 2
2022-10-15 00:07:28.333 Property SMC_Statemachine/Prop_UnitMode value of submodel
    $SM_Unit1_(SuperiorStateMachine) equals to 2

2022-10-15 00:07:28.348 Path: 1
2022-10-15 00:07:28.355 Executing path setup actions...
2022-10-15 00:07:28.362 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:28.484 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:28.492 Setting property SMC_Statemachine/Prop_UnitSelect of submodel
    SM_Unit2_(SubUnit1) to 1
2022-10-15 00:07:28.624 Property SMC_Statemachine/Prop_UnitSelect value of submodel
    SM_Unit2_(SubUnit1) equals to True
2022-10-15 00:07:28.631 Setting property SMC_Statemachine/Prop_UnitSelect of submodel
    SM_Unit3_(SubUnit2) to 1
2022-10-15 00:07:28.760 Property SMC_Statemachine/Prop_UnitSelect value of submodel
    SM_Unit3_(SubUnit2) equals to True
2022-10-15 00:07:28.768 Starting path execution...
2022-10-15 00:07:28.776 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:28.783 Sending event Clear
2022-10-15 00:07:28.903 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:28.911 Executing path teardown actions...
2022-10-15 00:07:28.917 Setting property SMC_Statemachine/Prop_UnitSelect of submodel
    SM_Unit2_(SubUnit1) to 0
2022-10-15 00:07:29.043 Property SMC_Statemachine/Prop_UnitSelect value of submodel
    SM_Unit2_(SubUnit1) equals to False
2022-10-15 00:07:29.053 Setting property SMC_Statemachine/Prop_UnitSelect of submodel
    SM_Unit3_(SubUnit2) to 0
2022-10-15 00:07:29.166 Property SMC_Statemachine/Prop_UnitSelect value of submodel
    SM_Unit3_(SubUnit2) equals to False
2022-10-15 00:07:29.174 Path passed!

2022-10-15 00:07:29.188 Path: 2
2022-10-15 00:07:29.196 Executing path setup actions...
2022-10-15 00:07:29.203 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:29.323 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:29.331 Starting path execution...
2022-10-15 00:07:29.339 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:29.346 Sending event Clear
2022-10-15 00:07:29.460 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:29.468 Testing transition from STOPPED to ABORTED
2022-10-15 00:07:29.475 Sending event Abort
2022-10-15 00:07:29.602 Expected state: Aborted, Current state: Aborted
2022-10-15 00:07:29.610 Path passed!

2022-10-15 00:07:29.624 Path: 3
2022-10-15 00:07:29.631 Executing path setup actions...
2022-10-15 00:07:29.638 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:29.758 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:29.766 Starting path execution...
2022-10-15 00:07:29.774 Testing transition from ABORTED to STOPPED
```

```
2022-10-15 00:07:29.781 Sending event Clear
2022-10-15 00:07:29.893 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:29.901 Testing transition from STOPPED to IDLE
2022-10-15 00:07:29.909 Sending event Reset
2022-10-15 00:07:30.033 Expected state: Idle, Current state: Idle
2022-10-15 00:07:30.042 Path passed!

2022-10-15 00:07:30.056 Path: 4
2022-10-15 00:07:30.064 Executing path setup actions...
2022-10-15 00:07:30.071 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:30.204 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:30.212 Starting path execution...
2022-10-15 00:07:30.219 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:30.227 Sending event Clear
2022-10-15 00:07:30.356 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:30.365 Testing transition from STOPPED to IDLE
2022-10-15 00:07:30.373 Sending event Reset
2022-10-15 00:07:30.497 Expected state: Idle, Current state: Idle
2022-10-15 00:07:30.505 Testing transition from IDLE to ABORTED
2022-10-15 00:07:30.514 Sending event Abort
2022-10-15 00:07:30.638 Expected state: Aborted, Current state: Aborted
2022-10-15 00:07:30.646 Path passed!

2022-10-15 00:07:30.661 Path: 5
2022-10-15 00:07:30.668 Executing path setup actions...
2022-10-15 00:07:30.675 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:30.794 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:30.802 Starting path execution...
2022-10-15 00:07:30.810 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:30.817 Sending event Clear
2022-10-15 00:07:30.930 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:30.937 Testing transition from STOPPED to IDLE
2022-10-15 00:07:30.945 Sending event Reset
2022-10-15 00:07:31.071 Expected state: Idle, Current state: Idle
2022-10-15 00:07:31.080 Testing transition from IDLE to STOPPED
2022-10-15 00:07:31.087 Sending event Stop
2022-10-15 00:07:31.209 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:31.216 Path passed!

2022-10-15 00:07:31.231 Path: 6
2022-10-15 00:07:31.238 Executing path setup actions...
2022-10-15 00:07:31.245 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:31.365 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:31.373 Starting path execution...
2022-10-15 00:07:31.381 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:31.388 Sending event Clear
2022-10-15 00:07:31.516 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:31.523 Testing transition from STOPPED to IDLE
2022-10-15 00:07:31.531 Sending event Reset
2022-10-15 00:07:31.655 Expected state: Idle, Current state: Idle
2022-10-15 00:07:31.664 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:31.671 Sending event Start
2022-10-15 00:07:31.795 Expected state: Execute, Current state: Execute
```

```
2022-10-15 00:07:31.803 Path passed!

2022-10-15 00:07:31.819 Path: 7
2022-10-15 00:07:31.826 Executing path setup actions...
2022-10-15 00:07:31.833 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:31.952 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:31.963 Starting path execution...
2022-10-15 00:07:31.971 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:31.978 Sending event Clear
2022-10-15 00:07:32.105 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:32.114 Testing transition from STOPPED to IDLE
2022-10-15 00:07:32.121 Sending event Reset
2022-10-15 00:07:32.243 Expected state: Idle, Current state: Idle
2022-10-15 00:07:32.251 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:32.258 Sending event Start
2022-10-15 00:07:32.384 Expected state: Execute, Current state: Execute
2022-10-15 00:07:32.392 Testing transition from EXECUTE to ABORTED
2022-10-15 00:07:32.400 Sending event Abort
2022-10-15 00:07:32.525 Expected state: Aborted, Current state: Aborted
2022-10-15 00:07:32.533 Path passed!

2022-10-15 00:07:32.548 Path: 8
2022-10-15 00:07:32.556 Executing path setup actions...
2022-10-15 00:07:32.562 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:32.686 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:32.694 Starting path execution...
2022-10-15 00:07:32.701 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:32.708 Sending event Clear
2022-10-15 00:07:32.823 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:32.832 Testing transition from STOPPED to IDLE
2022-10-15 00:07:32.839 Sending event Reset
2022-10-15 00:07:32.963 Expected state: Idle, Current state: Idle
2022-10-15 00:07:32.971 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:32.978 Sending event Start
2022-10-15 00:07:33.104 Expected state: Execute, Current state: Execute
2022-10-15 00:07:33.113 Testing transition from EXECUTE to STOPPED
2022-10-15 00:07:33.120 Sending event Stop
2022-10-15 00:07:33.243 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:33.251 Path passed!

2022-10-15 00:07:33.266 Path: 9
2022-10-15 00:07:33.273 Executing path setup actions...
2022-10-15 00:07:33.281 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:33.397 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:33.406 Starting path execution...
2022-10-15 00:07:33.416 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:33.423 Sending event Clear
2022-10-15 00:07:33.548 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:33.556 Testing transition from STOPPED to IDLE
2022-10-15 00:07:33.565 Sending event Reset
2022-10-15 00:07:33.688 Expected state: Idle, Current state: Idle
2022-10-15 00:07:33.696 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:33.703 Sending event Start
```

```
2022-10-15 00:07:33.828 Expected state: Execute, Current state: Execute
2022-10-15 00:07:33.836 Testing transition from EXECUTE to HELD
2022-10-15 00:07:33.843 Sending event Hold
2022-10-15 00:07:33.965 Expected state: Held, Current state: Held
2022-10-15 00:07:33.973 Path passed!

2022-10-15 00:07:33.987 Path: 10
2022-10-15 00:07:33.994 Executing path setup actions...
2022-10-15 00:07:34.002 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:34.120 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:34.127 Starting path execution...
2022-10-15 00:07:34.134 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:34.142 Sending event Clear
2022-10-15 00:07:34.257 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:34.266 Testing transition from STOPPED to IDLE
2022-10-15 00:07:34.274 Sending event Reset
2022-10-15 00:07:34.396 Expected state: Idle, Current state: Idle
2022-10-15 00:07:34.404 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:34.411 Sending event Start
2022-10-15 00:07:34.535 Expected state: Execute, Current state: Execute
2022-10-15 00:07:34.546 Testing transition from EXECUTE to SUSPENDED
2022-10-15 00:07:34.553 Sending event Suspend
2022-10-15 00:07:34.677 Expected state: Suspended, Current state: Suspended
2022-10-15 00:07:34.685 Path passed!

2022-10-15 00:07:34.700 Path: 11
2022-10-15 00:07:34.706 Executing path setup actions...
2022-10-15 00:07:34.714 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:34.833 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:34.841 Starting path execution...
2022-10-15 00:07:34.849 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:34.856 Sending event Clear
2022-10-15 00:07:34.984 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:34.992 Testing transition from STOPPED to IDLE
2022-10-15 00:07:34.999 Sending event Reset
2022-10-15 00:07:35.125 Expected state: Idle, Current state: Idle
2022-10-15 00:07:35.134 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:35.142 Sending event Start
2022-10-15 00:07:35.265 Expected state: Execute, Current state: Execute
2022-10-15 00:07:35.273 Testing transition from EXECUTE to HELD
2022-10-15 00:07:35.281 Sending event Hold
2022-10-15 00:07:35.405 Expected state: Held, Current state: Held
2022-10-15 00:07:35.414 Testing transition from HELD to ABORTED
2022-10-15 00:07:35.421 Sending event Abort
2022-10-15 00:07:35.546 Expected state: Aborted, Current state: Aborted
2022-10-15 00:07:35.554 Path passed!

2022-10-15 00:07:35.569 Path: 12
2022-10-15 00:07:35.576 Executing path setup actions...
2022-10-15 00:07:35.582 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:35.700 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:35.708 Starting path execution...
2022-10-15 00:07:35.716 Testing transition from ABORTED to STOPPED
```

```
2022-10-15 00:07:35.723 Sending event Clear
2022-10-15 00:07:35.851 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:35.859 Testing transition from STOPPED to IDLE
2022-10-15 00:07:35.866 Sending event Reset
2022-10-15 00:07:35.991 Expected state: Idle, Current state: Idle
2022-10-15 00:07:35.998 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:36.005 Sending event Start
2022-10-15 00:07:36.131 Expected state: Execute, Current state: Execute
2022-10-15 00:07:36.139 Testing transition from EXECUTE to HELD
2022-10-15 00:07:36.146 Sending event Hold
2022-10-15 00:07:36.273 Expected state: Held, Current state: Held
2022-10-15 00:07:36.281 Testing transition from HELD to STOPPED
2022-10-15 00:07:36.289 Sending event Stop
2022-10-15 00:07:36.410 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:36.419 Path passed!

2022-10-15 00:07:36.433 Path: 13
2022-10-15 00:07:36.441 Executing path setup actions...
2022-10-15 00:07:36.448 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:36.570 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:36.578 Starting path execution...
2022-10-15 00:07:36.586 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:36.593 Sending event Clear
2022-10-15 00:07:36.706 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:36.714 Testing transition from STOPPED to IDLE
2022-10-15 00:07:36.722 Sending event Reset
2022-10-15 00:07:36.846 Expected state: Idle, Current state: Idle
2022-10-15 00:07:36.853 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:36.860 Sending event Start
2022-10-15 00:07:36.984 Expected state: Execute, Current state: Execute
2022-10-15 00:07:36.991 Testing transition from EXECUTE to HELD
2022-10-15 00:07:36.999 Sending event Hold
2022-10-15 00:07:37.124 Expected state: Held, Current state: Held
2022-10-15 00:07:37.132 Testing transition from HELD to EXECUTE
2022-10-15 00:07:37.140 Sending event Unhold
2022-10-15 00:07:37.261 Expected state: Execute, Current state: Execute
2022-10-15 00:07:37.270 Path passed!

2022-10-15 00:07:37.286 Path: 14
2022-10-15 00:07:37.293 Executing path setup actions...
2022-10-15 00:07:37.300 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:37.418 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:37.425 Starting path execution...
2022-10-15 00:07:37.432 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:37.440 Sending event Clear
2022-10-15 00:07:37.554 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:37.562 Testing transition from STOPPED to IDLE
2022-10-15 00:07:37.568 Sending event Reset
2022-10-15 00:07:37.693 Expected state: Idle, Current state: Idle
2022-10-15 00:07:37.701 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:37.709 Sending event Start
2022-10-15 00:07:37.835 Expected state: Execute, Current state: Execute
2022-10-15 00:07:37.844 Testing transition from EXECUTE to SUSPENDED
2022-10-15 00:07:37.851 Sending event Suspend
2022-10-15 00:07:37.976 Expected state: Suspended, Current state: Suspended
```

```
2022-10-15 00:07:37.984 Testing transition from SUSPENDED to ABORTED
2022-10-15 00:07:37.991 Sending event Abort
2022-10-15 00:07:38.117 Expected state: Aborted, Current state: Aborted
2022-10-15 00:07:38.125 Path passed!

2022-10-15 00:07:38.139 Path: 15
2022-10-15 00:07:38.146 Executing path setup actions...
2022-10-15 00:07:38.154 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:38.275 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:38.284 Starting path execution...
2022-10-15 00:07:38.291 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:38.299 Sending event Clear
2022-10-15 00:07:38.410 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:38.417 Testing transition from STOPPED to IDLE
2022-10-15 00:07:38.424 Sending event Reset
2022-10-15 00:07:38.550 Expected state: Idle, Current state: Idle
2022-10-15 00:07:38.558 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:38.566 Sending event Start
2022-10-15 00:07:38.692 Expected state: Execute, Current state: Execute
2022-10-15 00:07:38.701 Testing transition from EXECUTE to SUSPENDED
2022-10-15 00:07:38.711 Sending event Suspend
2022-10-15 00:07:38.833 Expected state: Suspended, Current state: Suspended
2022-10-15 00:07:38.841 Testing transition from SUSPENDED to STOPPED
2022-10-15 00:07:38.848 Sending event Stop
2022-10-15 00:07:38.970 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:38.979 Path passed!

2022-10-15 00:07:38.995 Path: 16
2022-10-15 00:07:39.004 Executing path setup actions...
2022-10-15 00:07:39.011 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:39.143 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:39.151 Starting path execution...
2022-10-15 00:07:39.159 Testing transition from ABORTED to STOPPED
2022-10-15 00:07:39.166 Sending event Clear
2022-10-15 00:07:39.295 Expected state: Stopped, Current state: Stopped
2022-10-15 00:07:39.304 Testing transition from STOPPED to IDLE
2022-10-15 00:07:39.312 Sending event Reset
2022-10-15 00:07:39.436 Expected state: Idle, Current state: Idle
2022-10-15 00:07:39.444 Testing transition from IDLE to EXECUTE
2022-10-15 00:07:39.451 Sending event Start
2022-10-15 00:07:39.576 Expected state: Execute, Current state: Execute
2022-10-15 00:07:39.585 Testing transition from EXECUTE to SUSPENDED
2022-10-15 00:07:39.593 Sending event Suspend
2022-10-15 00:07:39.716 Expected state: Suspended, Current state: Suspended
2022-10-15 00:07:39.724 Testing transition from SUSPENDED to EXECUTE
2022-10-15 00:07:39.731 Sending event Unsuspend
2022-10-15 00:07:39.855 Expected state: Execute, Current state: Execute
2022-10-15 00:07:39.863 Path passed!
2022-10-15 00:07:39.870 Executing one time teardown actions...
2022-10-15 00:07:39.878 Setting property SMC_Statemachine/Prop_ControlCommand of submodel
    SM_Unit1_(SuperiorStateMachine) to 8
2022-10-15 00:07:39.997 Property SMC_Statemachine/Prop_ControlCommand value of submodel
    SM_Unit1_(SuperiorStateMachine) equals to Aborted
2022-10-15 00:07:40.005 Test passed!
```

Listing A.6: Beispielhafter Protokoll als Ergebnis einer zustandsbezogenen Testung

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 26.10.2022

Vasil Denkov