ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Sprachen zur Beschreibung und Vergleich zum Einsatzzweck von Sprachen zum Beschreiben von Kommunikationsabläufen

an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Kommunikationsinformatik
der Fakultät für Ingenieurwissenschaften

vorgelegt von
Majdi Taher und Alexander Huber

Saarbrücken, 02. August 2019

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 02. August 2019

Majdi Taher

Alexander Huber

Inhaltsverzeichnis

1 Einleitung

Seit einigen Jahrzehnten spielt die Art Prozessabläufe und abstrakte Sachverhalte für den Anwenderin einer möglichst nützlichen Beschreibung darzustellen eine wesentliche Rolle in der Informatik. So ist die Möglichkeit diese verschiedenen Sachverhalte zu beschreiben in den unterschiedlichsten Sprachen, ob formell oder informell, definiert worden. Zwar ist es möglich einen Sachverhalt individuell und intuitiv, von Person zu Person, zu beschreiben und zu Modellieren. Jedoch ist die Darstellung des Sachverhalts darauf bedacht, die Verwendung dieser durch mehrere Personen zu ermöglichen, da Missverständnisse auftreten können. Deswegen ist es notwendig eine einheitliche Syntax zu verwenden und eine Semantik zur Interpretationsminderung zu definieren. So können zum einen Domänenspezifische Sprachen verwendet werden oder globale Modellierungssprachen. Diese genannten Arten von Sprachen dienen zur Abstraktion und zum besseren Verständnis zwischen Mensch zu Mensch bzw. Computer. Sie sind sogenannte Formale Sprachen, welche künstlich erstellte Sprachen sind und sich für eine präzise Beschreibung von Eigenschaften und Verhalten eignen. Sie bestehen aus einer konkreten Syntax, abstrakten Syntax und Semantik. Das vorgestellte Projekt soll nun ausgewählte Modellierungssprachen in Hinsicht auf ihre Funktionsweise und Kommunikationsabläufe kritisch beobachten, beschreiben und Eignung bzw. Uneignung anhand ausgewählter Kriterien vornehmen.

1.1 Problemstellung

Da formale Sprachen künstliche Sprachen sind und einen Kontext über eine formale mathematische Syntax beschreiben, werden diese bevorzugt zur Abstraktion von Sachverhalten, wie der von Kommunikationsabläufen in Systemen, verwendet. Nun können unterschiedliche Sprachen einen Sachverhalt unterschiedlich gut oder schlecht wiedergeben. Wobei gut oder schlecht, abhängig des Einsatzortes ist und welche formalen Kriterien eine Wiedergabe erleichtert oder erschwert.

1.2 Zielsetzung

Das Ziel dieser Projektarbeit ist es den Einsatz unterschiedlicher Modellierungssprachen im Einsatz eines Kommunikationssystems zu vergleichen. Der Vergleich ist mit eigens gewählten Bewertungskriterien vorzunehmen und über den Einsatzort der Sprachen in einem Kommunikationssystem zu beschreiben. Dazu sollen die Informationen über die ausgewählten Modellierungssprachen aus den Dokumenten der Spezifikationen SDL Z.100 [13], MSC Z.120 [12] und OMG UML [8] entnommen werden. Die Vergleichskriterien werden über die verfügbare Literatur zusammen getragen und anhand damit die vorgestellten Sprachen ausgewertet.

1.3 Gliederung

Die Arbeit gliedert sich wie folgt in weitere Kapitel. Im Folgenden wird in Kapitel 3 auf die technischen Grundlagen der einzelnen Sprachen, deren Modellkonzeption, Notation

und Geschichte eingegangen. Dazu übernimmt Herr Alexander Huber die Verantwortung für Abschnitt 3 und Herr Majdi Taher Abschnitte 9 und 16. Danach konzentriert sich das dritte Kapitel auf die ausgewählten Bewertungskriterien, in denen wir diese beschreiben und den Bezugsraum darstellen. Herr Majdi Taher wird hierzu Abschnitt 24 und 26 und Herr Alenxander Huber Abschnitt 27 übernehmen. Nach diesen werden wir die einzelnen Sprachen auswerten. In Kapitel 31 werden dann die Sprachen hinsichtlich der genannten Kriterien aus Kapitel 23 analysiert und eine Eignung bzw. Uneignung ausgesprochen. Herr Majdi Taher wird hierzu Abschnitt 31 und Herr Alexander Huber Abschnitt 33 und 35 übernehmen. Das letzte Kapitel 37 handelt von den Erfahrungen und Erkenntnissen, die während der Arbeit gesammelt wurden. Sie wird noch einmal überblickt und schließt mit einem Fazit ab.

2 Grundlagen

Im Gegensatz zu weiten Teilen der Informatik besteht in der Telekommunikation seit langem die Notwendigkeit, herstellerunabhängige und präzise Standards für Kommunikationsprotokolle und Dienste zu erstellen. Diese Anforderung hat dazu geführt, dass man sich schon frühzeitig mit der Entwicklung von formalen Modellierungssprachen wie Message Sequence Chart (MSC), Unified Modelling Language (UML) und SDL beschäftigt hat. Im Gegensatz zur UML liegt MSC und SDL eine formale Semantik zugrunde, welche die Bedeutung einzelner Sprachkonstrukte klar und eindeutig mathematisch definiert. MSC und SDL sind von der International Telecommunication Union standardisiert worden. Für beide Sprachen existiert eine graphische Repräsentation (MSC /GR bzw. SDL/GR) für die Benutzung durch den Menschen und eine textuelle Notation (MSC/PR bzw. SDL/PR) für eine maschinelle Weiterbearbeitung. MSC und SDL werden häufig gemeinsam im Software-Entwicklungsprozess eingesetzt. MSC dient dabei zur Anforderungsdefinition, Testfallspezifikation und Dokumentation, wahrend SDL in der Spezifikationsphase und der Implementierungsphase eingesetzt wird und UML sich dazu eignet, Geschäftsprozesse bis hin zu ausführbaren Modellen abzubilden.

2.1 Specification and Description Language (SDL)

Die SDL (SDL, engl. Spezifikations und Beschreibungssprache) ist eine von der International Telecommunication Union (ITU) Telecommunication Standardization Sector (ITU-T) (ITU-T, engl. Internationalen Telekommunikations- Vereinigung für Standardisierung der Telekommunikation) standardisierte objektorientierte Modellierungssprache und die erste Version wurde erstmalig 1976 definiert. Die aktuellste Version von SDL ist SDL-2010 aus dem Jahre 2007, welche eine Überarbeitung der SDL-Version SDL-2000 aus dem Jahre 1999 ist [13, S. vii, 51]. Wenn im folgenden von SDL gesprochen wird, wird immer die Version SDL-2010 gemeint. SDL wird zur Beschreibung von Telekommunikationssystemen, deren Abläufe, sowie für Protokoll Definitionen und in verteilten Systemen eingesetzt. Der Anwendungsbereich erstreckt sich über komplexe ereignisgesteuerte, interaktiven Echtzeitanwendungen, welche über Nachrichten miteinander kommunizieren. Der Hauptfokus von SDL ist, sich einen genauen Überblick über das Verhalten genannter Systeme zu machen, wobei Eigenschaften mit anderen Techniken beschrieben werden müssen [13, S. 1 f.]. So wird MSC unter anderem zum Beschreiben von Interaktionsverhalten zwischen Systemen, Abstract Syntax Notation One (ASN1) für die Beschreibung von Datentypen und Testing and Test Control Notation Three (TTCN3) für Tests verwendet. In den letzten Überarbeitungen wurde die Spezifikation von SDL um objekt-orientierte Aspekte erweitert und mit Sprachen wie UML und ASN1 harmonisiert [13, S. vii ff.]. SDL kann entweder grafisch oder textuell beschrieben werden. Beide Notationen sind semantisch äquivalent und lassen sich ineinander überführen [17, S. 3].

2.1.1 Architektur

Die SDL-Spezifikation beschreibt ein hierarchisches System und erlaubt die strukturierte Trennung in Diagramme und Pakete und somit ein aufteilen eines Systems in viele

kleinere Teilsysteme. Dies wirkt sich positiv auf die Entwicklung eines Systems in größeren Gruppen aus und vereinfacht diese. Ein System ist eine Menge an Blöcken, welche über Kanäle miteinander und der Umgebung außerhalb des Systems verbunden sind. Ein Kanal besitzt die Namen der Nachrichten bzw. Signale, welche über ihn übertragen werden können. Blöcke können Mengen von weiteren Blöcken enthalten oder als Prozess verfeinert dargestellt werden. Ebenso können Prozesse eine Menge von Prozessen in sich definiert haben. Ein Prozess ist auf der untersten hierarchischen Ebene und beschreibt das dynamische Systemverhalten. Dies wird durch endliche Automaten, den Extended Finite State Machine (EFSM) um genauer zu sein, beschrieben. Diese besitzen abstrakte Datentypen und Merkmale von Objektorientierung [13, S. 3 f.]. Diese Strukturhierarchie wird anhand folgender Abbildung 2.1 veranschaulicht.

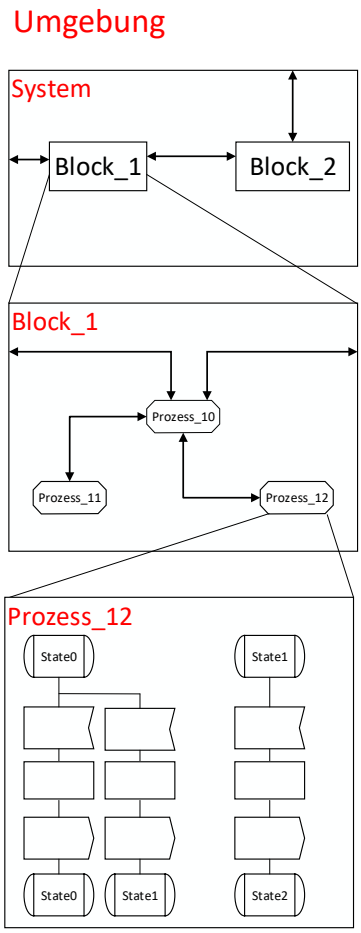

Abbildung 2.1: SDL Architekturhierarchie

2.1.1.1 Agenten

Seit SDL-2000 werden alle Funktionsblöcke übergreifend als Agenten bezeichnet. Der System-Agent, ist ein speziell ausgezeichneter Block-Agent. Er beinhaltet weitere Block-Agenten und definiert die Nachrichten mit denen über die Umgebung kommuniziert wird. Block- und Prozess-Agenten unterscheiden sich in über die Kontrolle ihrer Ausführungsart. Wo die Zustandsmaschinen von Block-Agenten nebenläufig ablaufen können, müssen Prozess-Agenten auf den Abschluss der in Ausführung befindlichen Transaktion warten um ihren Zustand wechseln zu können [14, S. 29 f.].

2.1.2 Kommunikation

Die Kommunikation innerhalb von SDL findet asynchron über Nachrichten bzw. mit diskreten Signalen (in SDL signals) über Kanäle (in SDL channel) statt. Diese können sowohl auf Systemebene, als auch auf Blockebene definiert werden und verbinden die Agenten untereinander oder mit der Umgebung. Jeder Kanal besitzt einen Namen, genauso wie Nachricht aber auch optionale Parameter besitzen und in eine Liste gruppiert werden können. Ein Kanal besteht immer aus einer Quelle und einem Endpunkt. Kanäle werden entweder Bi- oder Unidirektional definiert. Dabei kann ein Kanal verzögert oder nicht verzögert wirken. So können Nachrichten verzögert ankommen. Der Kanal muss entweder in einem anderen Agenten oder in die Umgebung (in SDL environment) enden. Er kann aber auch den innerbefindlichen Prozess eines Agenten mit der Umgebung bzw. mit einem anderen innerbefindlichen Agenten verbinden [14, S. 39–42]. Dabei ist die Umgebung alles, was in einer SDL spezifischen Sprache mit dem System kommunizieren kann [13, S. 3 f.]. Abbildung 2.2 zeigt die verschiedenen Kanäle beispielhaft.

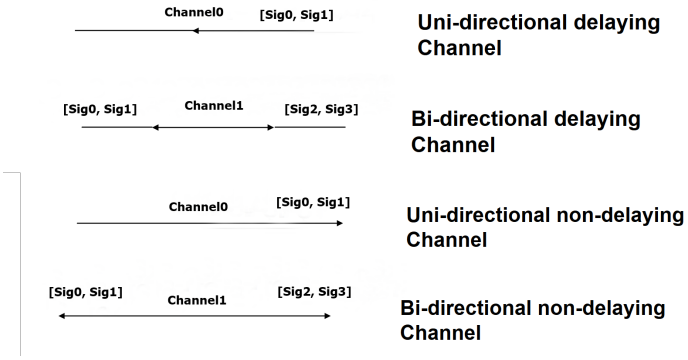

Abbildung 2.2: Verzögerte und nicht verzögerte Kanäle in SDL

2.1.3 Systemverhalten

Das Verhalten von SDL wird in Prozessen definiert, welche automatenbasiert und wie eben besprochen, mittels EFSMs repräsentiert werden. Sie bestehen aus einer Menge an Zustandsübergängen, welche Beispielhaft in Abbildung 2.3 zu sehen sind. Darunter gehören Zustände wie Start, Bedingung, Eingabe und Ausgabe. Da eine Aufzählung dem Sinn der Arbeit nicht dienlich ist, wird in diesem Zusammenhang auf die Spezifikation verwiesen [14, S. 44 ff.].

Ein Prozess besitzt eine Prozess-Id (in SDL PId), damit bei Instanziierung eines Prozesses, diese unterschieden werden können. Zusätzlich besitzt er einen Eingabepuffer zum Speichern von Nachrichten. Dieser Puffer übergibt nach dem First In First Out (FIFO)-Prinzip nacheinander die anstehenden Nachrichten des verbundenen Kanals an den Prozess [14, S. 42]. Abbildung 7 zeigt einen Prozess beispielhaft. Kommen zwei Nachrichten von mehreren Kanälen gleichzeitig an einem Prozess an, so ist das Systemverhalten undefiniert und nach dem Zufall wird eine der beiden Nachrichten zuerst ausgeführt. Die Modellierung der Automaten richtet sich oft nach Mealy, wobei in diesem die Ausgabe von seinem Zustand und seiner Eingabe abhängt. Im Gegensatz zu Moore-Automaten enthält er keinen Startzustand, wobei jeder Mealy-Automat in einen Moore-Automat übertragen werden kann.

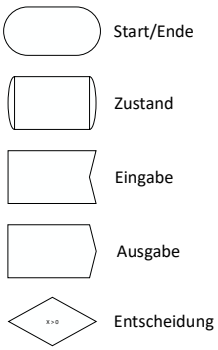

Abbildung 2.3: SDL-Basiskonstrukte

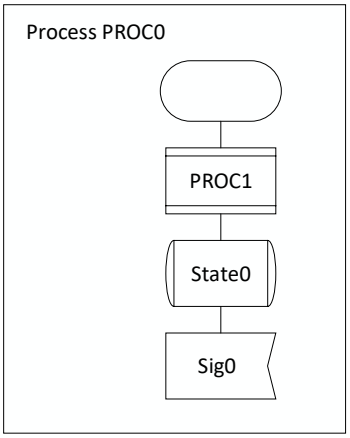

Abbildung 2.4: SDL-Prozessinitialisierung

2.1.4 Daten

In SDL können Daten auf zwei Arten beschrieben werden, einerseits mit dem ADT oder mit ASN1. Abstract Data Type (ADT) definiert keine Datenstrukturen, sondern jeweils einen Satz (in SDL sorts) an Werten, Operation und Bedingungen [16, S. 67]. In ADT sind einige Datentypen wie Boolean, Character, Charstring, Integer und Real vordefiniert, aber auch Konstrukte wie Array und Enum. Eine Liste der vordefinierten Datentypen kann aus der Spezifikation entnommen werden [16, S. 47 ff.]. Neue Datentypen lassen sich durch bereits vorhandene Datentypen definieren, hierzu sind Einschränkungen und ein Standardwert in der Definition mit anzugeben. Abbildung 2.5 verdeutlicht dies in einem Beispiel. In jedem Agenten lassen sich Konstanten, basierend auf den gewählten Datentypen definieren. Sie müssen einen konstanten Wert enthalten und dürfen nicht verändert werden. Variablen können nur in Prozessen definiert werden, enthalten denselben Aufbau wie Konstanten und können nur in ihrem definierten Prozess geändert werden. Zwar sind keine globalen Variablen zulässig, jedoch können Prozess eigene Variablen anderen Prozessen sichtbar gemacht werden. Diese Variablen nennt man remote variables [15, S. 31].

Zu aller erst wird der Name des zu erstellenden Typs definiert. Dann das Set an Werten, welche verwendet werden dürfen. Und zu guter Letzt werden die Bedingungen des Typs definiert und mit dem Namen des erstellten Typs wieder abgeschlossen.

```
NEWTYPE Boolean
        LITERALS True, False;
        OPERATORS
                ,NOT‘:Boolean->Boolean;
                ,=‘:Boolean.Boolean->Boolean
        AXIOMS
                ,NOT‘(True)== False;
                ,NOT‘(False)==True;
                ,=‘(True,True)==True;
                ,=‘(True,False)==False;
                ,=‘(False,True)==False;
                ,=‘( False, False)==True;
ENDNEWTYPE Boolean;
```

Abbildung 2.5: SDL-Datentypdefinition

2.1.5 Objektorientierung

Die objektorientierten Konzepte von SDL bieten dem Anwender eine Möglichkeit sein System zu strukturieren, wiederzuverwenden und das Modell ähnlicher der Realwelt abzubilden. So ist es möglich für einen Prozess-/Block-Agenten eine Prozess-/Block-Klasse zu erstellen. Im objektorientierten Ansatz werden diese auch als Typ (in SDL type) bezeichnet. Diese werden für die Instanziierung, Vererbung und Spezialisierung von Typen verwendet. Ein Klassenagent kann auf jeder beliebigen Ebene definiert werden, egal ob nahe beim Kontext oder nicht. Bei der Verwendung von Klassen, unterscheiden sich der objektorientierte und nicht-objektorientierte Ansatz syntaktisch zuerst nicht. Jedoch ist es möglich im objektorientierten Ansatz Instanzen der Klasse zu erzeugen [13, S. 4 ff.].

2.1.6 Spracheigenschaften

Die Eigenschaften der Sprachdefinition von SDL sind folgend beschrieben[**ITUT111_2016**]:

Abstrakte Grammatik Die abstrakte Grammatik von SDL wird von einer abstrakten Syntax und statischen Bedingungen beschrieben. Die Abstrakte Syntax kann entweder mit einer textbasierten Grammatik oder einem grafischen Metamodell erstellt werden.

Konkrete Grammatik Die konkrete Grammatik wird durch eine grafische Syntax, statischen Bedingungen und Regeln für die grafische Syntax beschrieben. Beschrieben wird sie durch die erweiterte Backus-Naur Form. Wenn jedoch in der abstrakten Grammatik ein grafisches Metamodell verwendet wurde, ist es erlaubt dieses um konkrete Eigenschaften zu erweitert und zu verwenden.

Semantik Die Semantik gibt einem Konstrukt eine klare Bedeutung. Sie enthält dessen Eigenschaften, die Art der Interpretation und dynamischen Bedingungen die erfüllt sein müssen, damit das Konstrukt sich so verhält, sodass es in der Art verhält wie die Sprache gestaltet wurde. Die Eigenschaften sind wohl definierte Beziehungen, zwischen verschiedenen Konzepten.

Model Das Model gibt Notationen eine Abbildungsform, wenn diese keine direkte abstrakten Syntax besitzen. Diese sind dann nach der konkreten Syntax anderer Konstrukte zu modellieren. Diese Konstrukte werden als ßhorthand"bezeichnet und beinhalten die Syntax der transformierten Form.

2.1.6.1 Metamodell

Es werden derzeit Anstrengungen unternommen ein öffentlich zugängliches Metamodell von SDL zu erstellen, welches alle Aspekte der Sprache in sich vereinigt. Zu dem derzeitigen Standpunkt existiert jedoch keines oder ist nicht uns bekannt. Deswegen hat die ITU-T selbst ein Meta-Metamodell auf Grundlage von UML erstellst, welches jedoch auch die Definition nicht vollends abdeckt. Das Meta-Metamodell wird auch als SDL-UML bezeichnet [**ITUT109_2016**].

2.2 Unified Modeling Language (UML)

2.2.1 Historie und Ziel

Die UML ist eine durchgängige Modellierungssprache von der organisatorischen Beschreibung von Geschäftsprozessen bis zu direkt ausführbaren Modellen, d.h. bis zur Implementierung. Sie ist über ISO standardisiert (ISO /IC 19501).
Ein erster Ansatz wurde 1990 auf der Grundlage verschiedener Notationssysteme entwickelt. Die Standardisierung, Pflege und Weiterentwicklung der Sprache wurde an die OMG übergeben, die die Sprache im Jahr 1997 zur Version UML 1.1 weiterentwickelte. Seit Ende der 1990er Jahre haben zahlreiche Personen und Institutionen intensiv an der UML Version 2.0 gearbeitet, die im Jahr 2006 vollständig fertig gestellt und Anfang 2009 von der leicht überarbeiteten Version 2.2 abgelöst wurde. Eine Standardisierung durch die International Standardization Organization (ISO) hat die Version 2.2 allerdings noch nicht erreicht. Diese bleibt bisher nur der Version 1.4.2 vorbehalten [10].

2.2.2 Metamodell

UML legt Spracheinheiten fest, die auf verschiedenen Ebenen agieren. Mit diesen drücken sie die Struktur und das Verhalten eines Systems aus. Einige Elemente nutzt die Modellierungssprache, um sich selbst zu definieren. Die Meta-Modellierung umfasst alle Elemente von UML, auch solche, die UML selbst beschreiben. Dafür nutzt es vier hierarchisch angeordnete Ebenen (M0 bis M3).

Die Meta-Metaebene M3 spezifiziert die Metadaten der Modellierungssprache und deren Zusammenhänge mithilfe der Meta Object Facility (MOF). Sie definiert das Metamodell. Zudem befähigt Sie den Metadaten-Transfer. Das von der OMG definierte Format XMI ist ein praktisches Tool, um objektorientierte Daten auf Meta-Metaebene zwischen Entwicklungstools zu teilen. Die Object Constraint Language (OCL), eine deklarative Programmiersprache, ergänzt UML und reguliert Randbedingungen der jeweiligen Modellierung. Als Textsprache wirkt sie jedoch nur unterstützend, statt selbst für Modellierung zur Verfügung zu stehen.

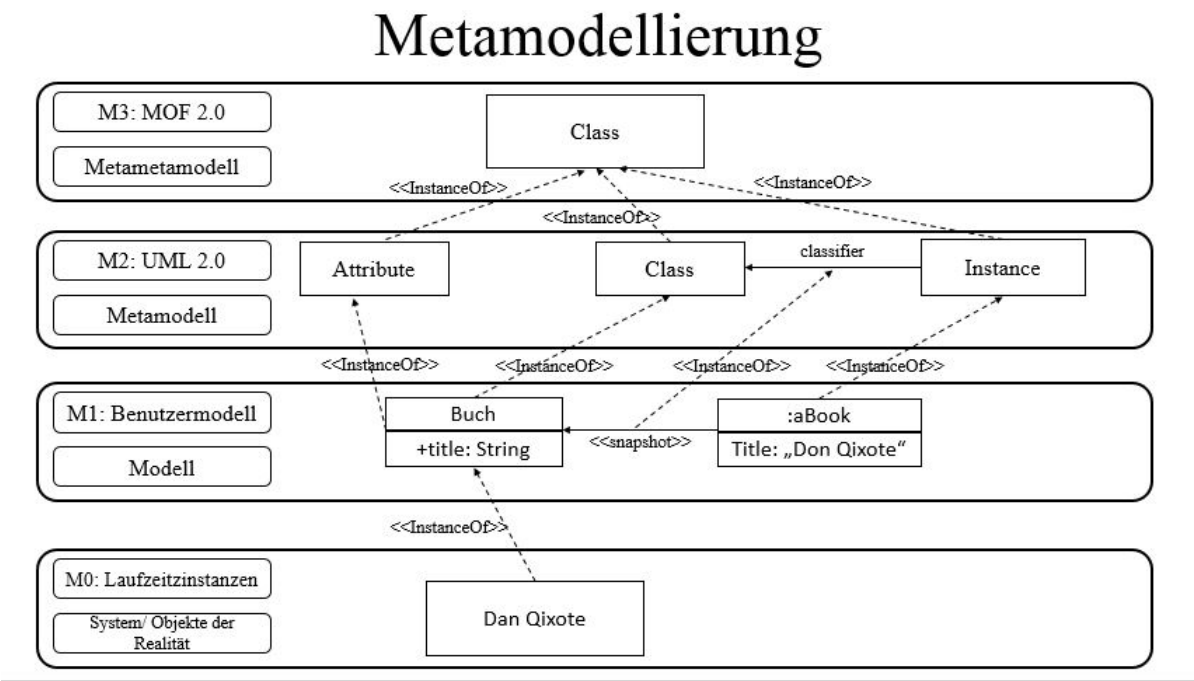

Abbildung 2.6: Die Metamodellierung zeigt die hierarchische Beziehung zwischen den Sprachebenen

Die obere Grafik zeigt die Metamodellierung von UML 2.0. Ebene M0 ist die grundlegende Ebene. Sie stellt konkrete, reale Objekte und einzelne Datensätze dar – z. B. ein Objekt oder eine Komponente. Ebene M1 umfasst alle Modelle, die die Daten der Ebene M0 beschreiben und strukturieren. Das sind UML-Diagramme wie das Aktivitätsdiagramm oder das Paketdiagramm (weiter unten erklärt). Um den Aufbau dieser Modelle zu definieren, legen Metamodelle der Ebene M2 die Spezifikationen und Semantik der Modellelemente fest.

Wollen Sie ein verständliches UML-Diagramm erstellen, müssen Sie das Metamodell UML mit seinen Regeln kennen. Die höchste Ebene, M3, ist ein Metamodell des Metamodells. Die erwähnte Meta Object Facility arbeitet auf einer abstrakten Ebene, die Metamodelle definiert. Diese Ebene definiert sich selbst, da sonst weitere, übergeordnete Meta-Ebenen entstünden.

2.2.3 Kurzbeschreibung

Bei der Unified Modeling Language (UML) handelt es sich nicht um eine bestimmte Methode, sondern vielmehr um einen Sammelbegriff für grafische Methoden der objektorientierten Entwicklung und Dokumentation von Software (Object Oriented Design – OOD). Dies umfasst Methoden und Notationen für Planung, Design, Entwurf und Implementierung von Software, die seit den 90er Jahren durch die Object Management Group (die auch BPMN pflegt) zu einem offiziellen Standard, der UML, zusammengeführt wurden.[7]
Im Gegensatz zu anderen Methoden, die primär auf die Modellierung von Prozessen abzielen, kann die UML direkt zur Software-Entwicklung genutzt werden.
Die objektorientierte Sichtweise, auf der UML basiert, zieht ausgehend von der realen Welt Objekte heraus, die mit Attributen beschrieben werden. Die Objekte werden zu Klassen verdichtet, wenn Eigenschaften und Verhalten der Objekte identisch oder ähnlich

sind. Klassen können daher als Baupläne für die zu erzeugenden Objekte (die Instanzen einer Klasse) interpretiert werden. Die Objekte, Klassen, Attribute und Methoden bilden die Basis für sämtliche Diagrammtypen der UML. Die verschiedenen Diagrammtypen können in

statische Modelle (-> Strukturdiagramme)
- das Klassendiagramm,
- das Kompositionsstrukturdiagramm (auch: Montagediagramm),
- das Komponentendiagramm,
- das Verteilungsdiagramm,
- das Objektdiagramm,
- das Paketdiagramm und
- das Profildiagramm.

und
dynamische Modelle (-> Verhaltensdiagramme)
- das Aktivitätsdiagramm,
- das Anwendungsfalldiagramm (auch: Use-Case o. Nutzfalldiagramm),
- das Interaktionsübersichtsdiagramm,
- das Kommunikationsdiagramm,
- das Sequenzdiagramm,
- das Zeitverlaufsdiagramm und
- das Zustandsdiagramm

unterteilt werden.
Statische Modelle (wie z.B. das Klassendiagramm) zeigen die Beziehungen zwischen den Klassen und den beteiligten Akteuren auf. Demgegenüber zeigen dynamische Modelle (wie z.B. das Sequenzdiagramm) den Prozessablauf auf. Aufgrund der Vielzahl an Diagrammtypen ergeben sich vielfältige Anwendungsmöglichkeiten, da jeder Typ eine spezifische Sicht auf den zu modellierenden Prozess sowie das System ermöglicht.
Im Folgenden werden einige ausgewählte UML-Diagrammtypen erläutert, wobei jeweils der Nutzen im Rahmen des Geschäftsprozessmanagements herausgearbeitet wird.

Das Anwendungsfalldiagramm (use case diagram) dient im Rahmen der Softwareentwicklung dem Einstieg in die Anforderungsanalyse. Gleichzeitig kann es zur Darstellung der relevanten Geschäftsprozesse einschließlich der Beziehung zu den an den Prozessen beteiligten Personen genutzt werden. Ein Anwendungsfall entspricht hierbei entweder einem Geschäftsprozess oder einem Teilprozess. Durch das Anwendungsfalldiagramm kann dann dargestellt werden, welche Akteure an den betrachteten Prozessen beteiligt sind und welche Prozesse weitere Prozesse beinhalten. Die Akteure sind dabei im Sinne von Rollen eines Benutzers innerhalb des Systems zu verstehen, wobei diese nicht zwingend menschlich sein müssen. Dies ist beispielsweise der Fall, wenn ein anderes System eingebunden wird. Die Anwendungsfälle werden in diesem Diagrammtyp über Ellipsen abgebildet, während die Akteure als Strichmännchen dargestellt werden. Die Beziehungen zwischen dem Anwendungsfall und den beteiligten Akteur werden über ungerichtete Kanten visualisiert.

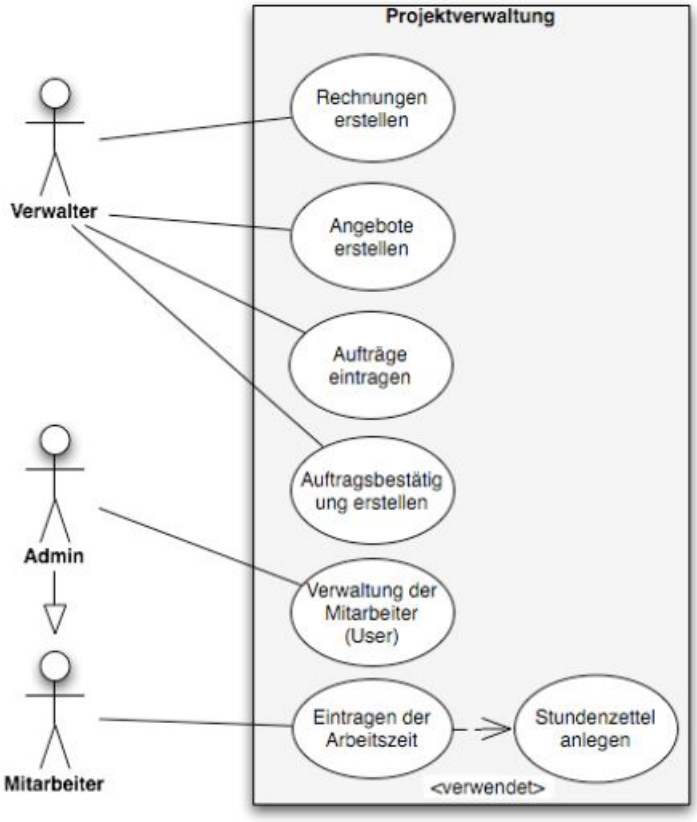

Abbildung 2.7: Beispiel eines Anwendungsfalldiagramms in UML

Quelle : [7]

Anwendungsfalldiagramme weisen jedoch den Nachteil auf, dass keine Reihenfolge bei der Bearbeitung der Anwendungsfälle abgebildet werden kann. Es wird somit nicht deutlich, welcher Anwendungsfall vor einem anderen durchgeführt werden muss. Allerdings kann diese Reihenfolge beispielsweise durch andere Methoden der UML, wie das Aktivitätsdiagramm, kompensiert werden. Zudem muss jeder Anwendungsfall anhand einer Beschreibung dokumentiert werden. Hierbei bestehen jedoch keine Vorgaben, weshalb sich Inkonsistenzen und Redundanzen ergeben können. Darüber hinaus kann nicht sichergestellt werden, dass Dritte die Dokumentation auch verstehen. Daher sollte auf Vorlagen, die nicht offiziell zum Standard der UML gehören, zurückgegriffen werden.[7]

Das Klassendiagramm (Class Diagram) eignet sich zur Darstellung von Klassenstrukturen innerhalb eines IKS. Klassendiagramme können nicht direkt für die Geschäftsprozessmodellierung verwendet werden. Stattdessen handelt es sich eine statische Darstellung von Klassen, Objekten und deren Beziehungen untereinander. Klassendiagramme beschreiben jedoch nur, dass eine Interaktion besteht; wie diese ausgestaltet ist kann jedoch nicht dargestellt werden.

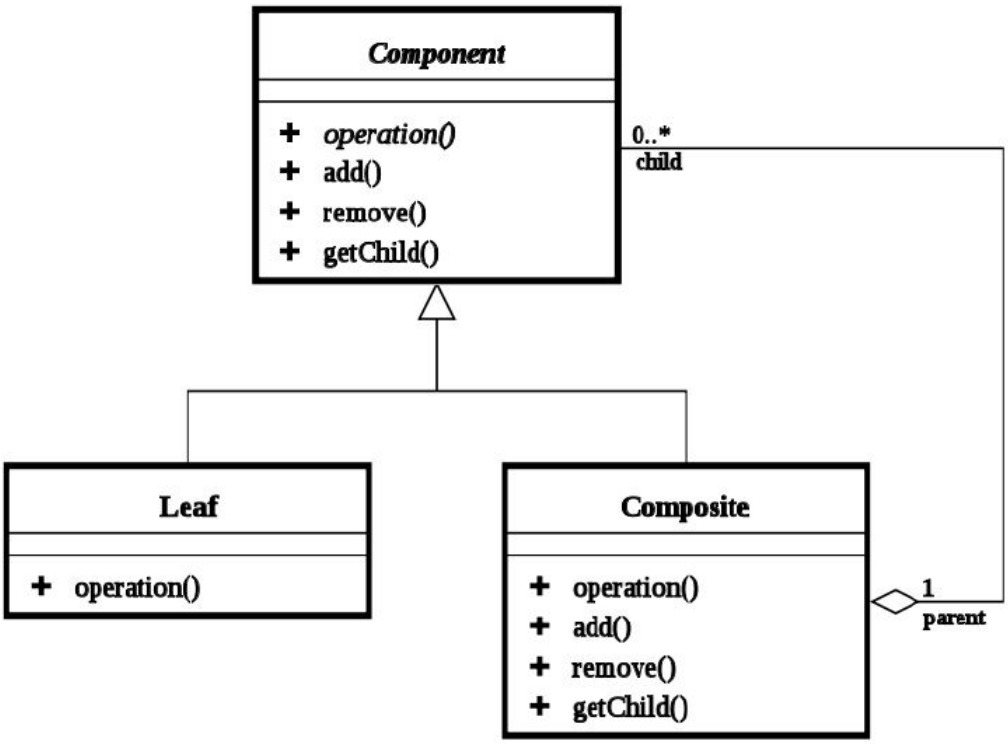

Abbildung 2.8: Beispiel eines Klassendiagramms in UML

Quelle : [7]

Das Aktivitätsdiagramm oder auch Ablaufdiagramm (Activity Diagram) wird zur Darstellung von Abläufen verwendet. Im Mittelpunkt steht dabei die Visualisierung paralleler Abläufe. Aus diesem Grund eignet sich dieser Diagrammtyp in einem besonders hohen Maße zur Abbildung von Geschäftsprozessen, da diese zumeist Parallelitäten vorweisen. Aktivitätsdiagramme sind darüber hinaus zur Modellierung von Workflows und zur Verfeinerung von Anwendungsfällen geeignet. Des Weiteren sind Aktivitätsdiagramme in der Lage unterschiedliche Detaillierungsgrade wiederzugeben. So ist es unter anderem möglich ein anwendungsfallübergreifendes Diagramm zu erzeugen und anschließend die darin enthaltenen Anwendungen einzeln zu modellieren.

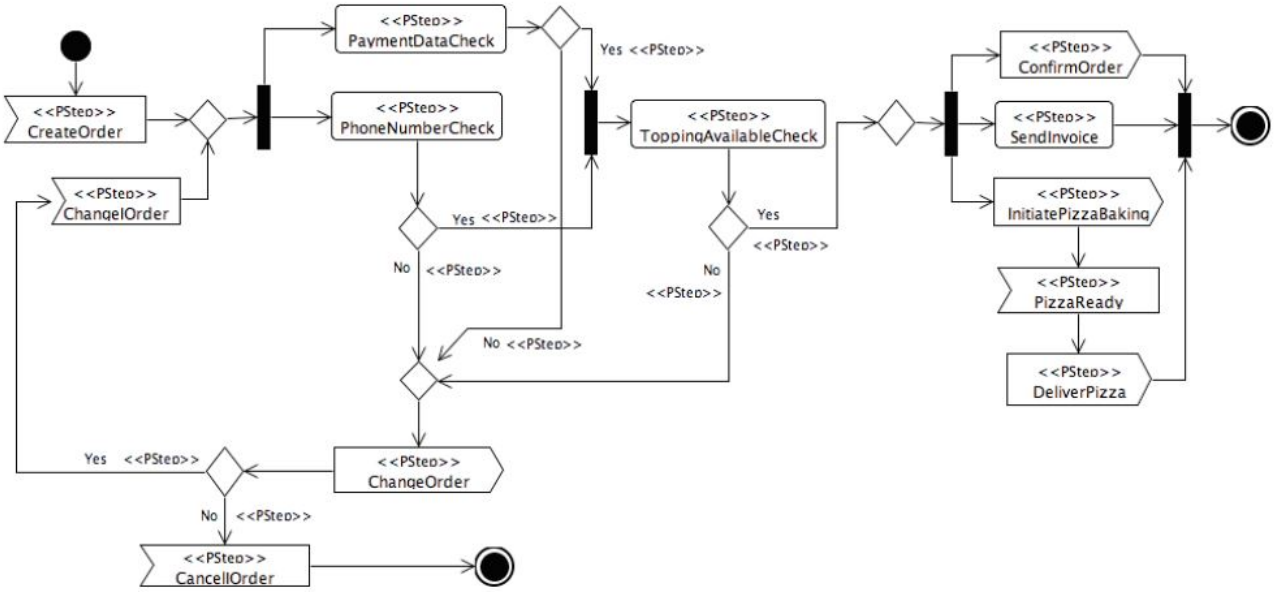

Abbildung 2.9: Beispiel eines Aktivitätsdiagramm in UML

Quelle : [7]

2.2.4 UML-Diagramme: Eine Übersicht

Die folgende Übersicht zeigt übergeordnete Kategorien und Anwendungsmöglichkeiten der einzelnen Diagrammtypen in Kurzform. Wenn Sie ein modellorientiertes Software-System, einen Anwendungsfall in der Wirtschaft o. Ä. visuell darstellen wollen, sollten Sie laut Empfehlung der UML-Task-Force vorher einen der UML-Diagrammtypen wählen. Erst dann lohnt es sich, eines der vielen UML-Tools zu wählen, da diese häufig eine gewisse Methode vorschreiben. Dann erstellen Sie das UML-Diagramm.**[MT011]**

Kategorie	Diagrammtyp	Verwendung
Struktur	Klassendiagramm	Klassen visualisieren
	Objektdiagramm	Systemzustand in einem bestimmten Moment
	Komponenten-diagramm	Komponenten strukturieren und Abhängigkeiten aufzeigen
	Kompositionsstruktur-diagramm	Komponenten oder Klassen in ihre Bestandteile aufteilen und deren Beziehungen verdeutlichen
	Paketdiagramm	Fasst Klassen in Pakete zusammen, stellt Pakethierarchie und -struktur dar
	Verteilungsdiagramm	Verteilung von Komponenten auf Rechnerknoten
	Profildiagramm	Veranschaulicht Verwendungszusammenhänge durch Stereotype, Randbedingungen etc.
Verhalten	Anwendungsfall-diagramm	Stellt diverse Anwendungsfälle dar
	Aktivitätsdiagramm	Beschreibt das Verhalten verschiedener (paralleler) Abläufe in einem System
	Zustandsautomaten-diagramm	Dokumentiert, wie ein Objekt von einem Zustand durch ein Ereignis in einem anderen Zustand versetzt wird.
Verhalten: Interaktion	Sequenzdiagramm	Zeitlicher Ablauf von Interaktionen zwischen Objekten
	Kommunikations Diagramm	Rollerverteilung von Objekten innerhalb einer Interaktion
	Zeitverlaufsdiagramm	Zeitliche Eingrenzung für Ereignisse, die zu einem Zustandswechsel führen
	Interaktionsübersichts-diagramm	Sequenzen und Aktivitäten interagieren

Abbildung 2.10: UML-Diagramme

2.2.5 Vor- und Nachteile im Überblick

UML ist heute eine der dominierenden Sprachen für die Modellierung von betrieblichen Anwendungs- bzw. Softwaresystemen. Der erste Kontakt zu UML besteht häufig darin, dass UML-Diagramme im Rahmen von Softwareprojekten zu erstellen, zu verstehen oder zu beurteilen sind. UML-Diagramme gelten als Standard bei objektorientierter Modellierung.

Vorteile	Nachteile
Standardisierte, weit verbreitete Methode	Grundsätzliches IT-Wissen erforderlich

Ergebnisse der Modellierung in UML können zur Umsetzung in Software-Code verwendet werden. Es ist ein automatisierte Erzeugung eines Coderahmen aus den Diagrammen möglich wodurch der Programmier-aufwand reduziert wird	Sehr großer Umfang und Komplexität Sprache (beispielsweise umfasst Sprachspezifikation mehr als 1200 Sei verursacht Schwierigkeiten bei der Benutz und erfordert einen ho Einarbeitungsaufwand
Einfacher Modellaustausch möglich und damit bessere Modellwiederverwendung	Die UML wird hinsichtlich semantis Inkonsistenzen, Konstruktmehrdeutigkei inadäquaten Notationen und kognit Unzugänglichkeiten kritisiert
Eine große Anzahl an UML-Werkzeugen bietet Export-Möglichkeiten in Java, C und C# an. Der Quellcode kann direkt in die technische Implementierung der IT-Anwendung einfließen	Die UML wurde ursprünglich für Entwicklung von Softwaresystemen konzip daher sind die relevanten Beschreibungs-Darstellungsmittel weniger auf Prozessmodellierung ausgerichtet

Abbildung 2.11: UML: Vor- und Nachteile

Quelle : [7]

2.3 MSC

MSC ist von der International Telecommunication Union (ITU) normierte Sprache. Sie hat eine graphiseche Notation, um den Benutzer zu darstellen, und eine textuelle Notation für die Bearbeitung durch Software-Werkzeuge, genauso wie bei SDL. MSC bietet die Möglichkeit, das Kommunikationsverhalten zwischen Systemkomponenten und deren Umgebung zu beschreiben. Diese Kommunikations ist auf den Austaush von Nachrichten basiert. Mit Msc-Specifikationen können unterschiedliche Sichten auf das Systemverhalten erzeugt werden. Dazu gibt MSC einen globalen Einblick auf das System aus, wie zum Beispiel die Interaktionen von alle Instanzen.
MSC und UML haben fast das gleiche Prinzip und man kann sogar als Variante von MSC den Sequence Diagrams in UML integrieren. Trotz der vielen Gemeinsamkeiten von dieser drei Modellierungssprachen (UML, SDL und MSC), haben sie viele große Unterschied in dem tatsächlichen Entwurf. Beispielsweise kann man mit MSC nur Beispielabläufe und kein vollständig System beschreiben.
Die MSC-Diagramme sind in den frühen Phasen hilfreich, wobei können sie erste Ideen und Einblick zum System anzeigen. Damit kann man einfach in der Entwurfsphase wichtige erwünschte oder unerwünschte Beispielabläufe bemerken oder in späteren Phasen den Testzweck schauen, der von die MSC-Diagramme genau definiert wurde.

Die wichtigsten Konstrukte und Elemente der MSC werden in diesem Abschnitt anhand ein Beispiel dargestellt. Wir glauben, dass ein Beispiel das beste Weise ist, um Das MSC Konzept zu erklären und zu definieren. Das folgende Beispiel beschreibt einen Prozess,

der aus vier verschiedenen Einzelteilen (A, B, C und D) zwei Produkte von Typ AC oder BDC produziert.

2.3.1 Basic-MSC

Das erste wichtige Teil von MSC ist ein Basic-MSC. Ein Basic-MSC umfasst alle notwendige Srachkonstrukte zur Spezifizierung des Nachrichtenfluss. Die Sprachkonstrukte von einem einfachen MSC (Basic MSC9 sind: Instanz, Message, Environment, Action, Timer-Start, Timeout, Timer-Stop, Create, Stop und Condition.
Das wichtigste Teil vom Basic-MSC sind den Nachrichtenaustausch. Diese Nachrichten können verschiedene Parameter tragen, die den Typ einer Nachricht unterscheiden. Auf dieser Weise sind zwei Nachrichten gleich, wenn sie den gleichen Inhalt besitzen. Ein Basic-MSC definiert eine partielle Ordnung auf Ereignissen, die an einem Ort (heißt Instanz) stattfinden. Diese Ereignissen können in verschiedenen Fromen erscheint werden, zum Beispiel: Kommunikationsereignisse (wie das Senden und Empfangen einer Nachricht), Timer-Ereignisse, lokale Aktionen oder das Erzeugen einer Instanz.
Um ein Basic-MSC besser zu erklären, ist ein Beispiel davon in der nächsten Abbildung gezeigt. Ein Basic-MSC mit dem Namen ProduktionAC ist in Abbildung 1.6 gezeigt.

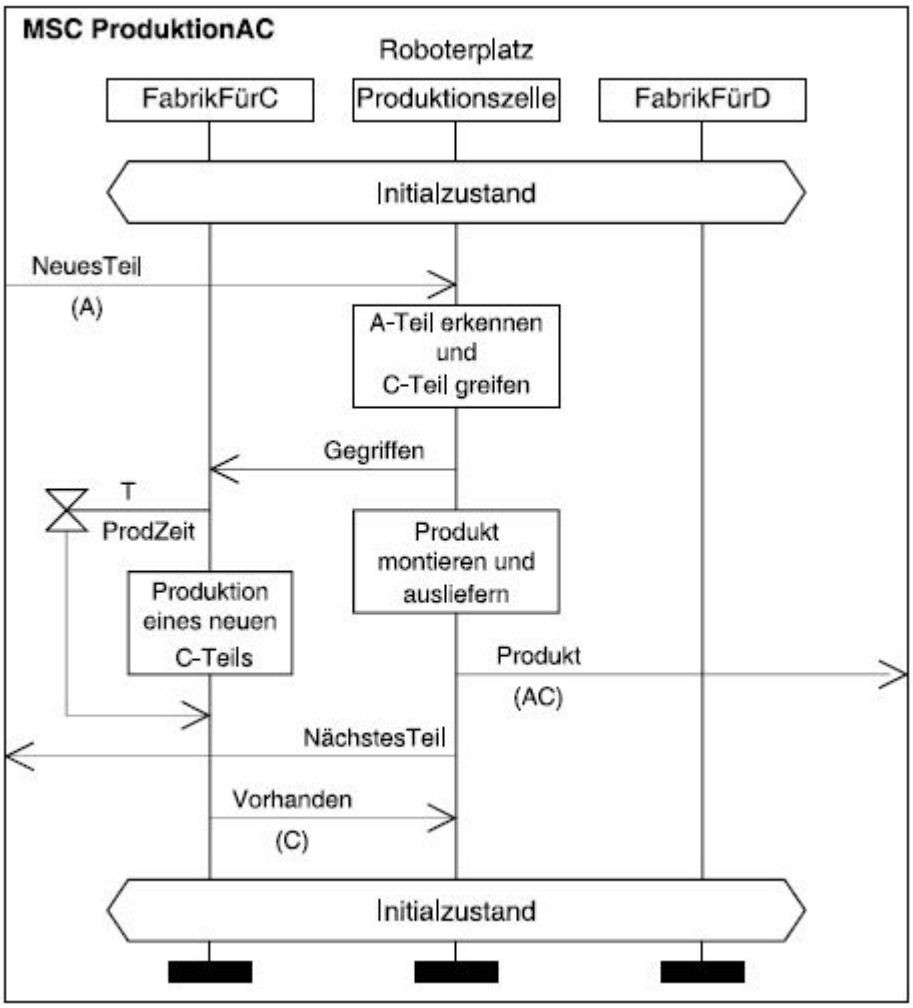

Abbildung 2.12: Ein Basic-MSC

Quelle : [3] Part 1: Message Sequence Chart (MSC)

Der Name von diesem Basic-MSC ist ProduktionAC und er muss oben in der Ecke

geschrieben werden. Das Basic-MSC in diesem Beispiel beschreibt die Produktion eines Produkts, das aus einem A-Teil und einem C-Teil zusammengesetzt ist. «««< HEAD Die wichtige Elemente in dieser Abbildung sind: Der Produktionszelle, FabrikfürC und Fabrik von D. Das MSC beschreibt ein detaillierte Szenario vom Informationsaustausch zwischen diese drei Element.

======= Die wichtige Elemente in dieser Abbildung sind: Der Produktionszelle, FabrikfürC und Fabrik von D. Das MSC beschreibt ein detaillierte Szenario vom Informationsaustausch zwischen diese drei Element. »»»> 321950359661aae8443acada7239309cebbc5118 Am Anfang ist der Produktionsprozess in seinem Initialzustand, Der ein benötige Bedingung für die nächste Aktionen ist. Das System ist initialisiert und Teile der Typen C und D sind gefertigt, eingelagert und der Produktionszelle zur Verfügung gestellt worden. Die Systemumgebung schickt das A-Teil an die Produktionszelle. Danach erkennt die Produktionszelle das empfangende Teil, greift das für die Produktion notwendige C-Teil und meldet das Entfernen des C-Teils vom Greifplatz an die FabrikfürC, die für das C-Teile zuständig ist. Beim letzten Schritt wird ein Produkt von Typ AC produziert, ausgegeben und zu die Systemumgebung geschickt. Danach meldet die Produktionszelle die Bereitschaft für die Bearbeitung des nächsten A- oder B-Teils an die Systemumgebung. In der Gleichzeitig wird ein neues C-Teil wieder produziert und es ist jetzt verfügbar für die nächste Produktion.

2.3.1.1 Instanz und Message

Instanzen und Nachrichten (Message) sind die wichtigste Sprachkunstrukte von Basic-MSC. Instanzen sind Komponenten, die mit der Systemumgebung oder untereinander asynchron Nachrichten austauschen können.
Die Instanzen sind als vertikale Linien dargestell. Am Enfang dieser Linier findet man der Instanzkopf, wo der Instanzname spezifiziert wird. Zusätzlich kann man noch direkt über den Instanzkopf ein Typ zu dieser Instanz angeben (z.B. Instanz Produktionszelle vom Typ Roboterplatz in Abbildung). Die Linie der Instanzen endet sich mit einem Instanz-Ende-Symbol. Ein Instanz-End bedeutet, dass es in dem MSC keine weiteren Ereignisse für diese Instanz gibt, aber das bedeutet nicht unbedingt, dass die Instanz gestoppt ist.

Die Nachrichten werden durch Pfeile dargestellt. Die Pfeile können horizontal oder geneigt sein, um den Zeitverlauf anzuzeigen. Jede Nachricht definiert definiert genau zwei Eereignisse: Das Senden der Nachricht ist durch den Pfeileanfang beschrieben und die Pfeilspitze bezeichnet die Verarbeitung einer Nachricht. Jede Nachricht hat einen Name und optionale Nachricht-Parametern, die in Klammern angegeben werden müssen (Zum Beispiel in Abbildung hat die erste Nachricht den Name NeuesTeil mit dem Parameter A). Bei Jeder Instanzlinie gibt es eine Totalordnung der der spezifizierten Message-Sende- und Message-Verarbeitungsereignisse. Dazu gibt es zwischen die verschieden Instanzachse partiell Ordnung, wobei sollte eine Nachricht zuerst gesendet werden, bevor sie empfängt wird.

2.3.1.2 Environment

Die Systemumgebung ist durch die Diagrammfläsche eines MSC definiert. Diese Diagrammfläsche wird durch einen rechteckigen Rahmen begrenzt. die Systemumgebung und heißt Environment. Nachrichten, die aus der Systemumgebung kommen oder an die Systemumgebung gesendet werden, beginnen und enden auf dem Environment (z.B. die

Messages NeuesTeil und Produkt in Abbildung). Die Sende- und Verarbeitungsereignisse aus und an die Systemumgebung definieren in der Regel keine Ordnung.

2.3.1.3 Action

Aktionen von Instanzen können ind Form von Actions spezifiziert werden. Eine Action wird durch ein Rechtecksymbol dargestellt, der Name der Aktion enthalten kann (z.B. Action ‚Produkt montieren und ausliefern' in Abbildung).

2.3.1.4 Timer

Die Sprachkonstrukte Timer-Start, Timeout und Timer-Stop können zusätzlich die Timern beschreiben, damit wird es deutlicher den Anfang und das Ende der Aktionen definiert. Der Anfang der Pfeile(mit Sanduhr-Symbol) definiert das Time-Start und das Ende der Pfeile definiert das Timeout. Zusätzlich muss man den Name von Timer und optional die Timer-Parametern definieren.
Ein Timer-Start und seine Timeout werden in Bild Abbildung verwendet, um die Produktionszeit für ein Teil des Typs C zu modellieren.In diesem Fall hat der Timer hat den Namen T und den Parameter Prodzeit.

2.3.1.5 Condition

Die Beschreibung vom Zustand definiert durch Condition (Bedingung) Graphisch werden Conditions durch Sechsecke dargestellt, die die Instanzen, auf die sich die Condition bezieht, überdecken. Conditions werden zur Beschreibung von wichtigen Systemzuständen benutzt. In Abbildung befinden sich zwei Conditions, die Initialzustand heißt und den globalen Systemzustand beschreiben.

2.3.2 Strukturelle Sprachkonstrukte

Strukturelle MSC-Sprachkonstrukte bezeichnen Sprachelemente, die über die Beschreibung des reinen Messageflusses hinausgehen. Mit ihnen lassen sich MSCs und MSC-Teile zu komplexeren Abläufen kombinieren (Inline- Expressions und High-Level-MSC), MSC-Diagramme in anderen MSC-Diagrammen wiederverwenden (References), MSC-Instanzen verfeinern (Decomposition) und allgemeine Ereignisstrukturen für Instanzen definieren (Coregion und General Ordering).[3]

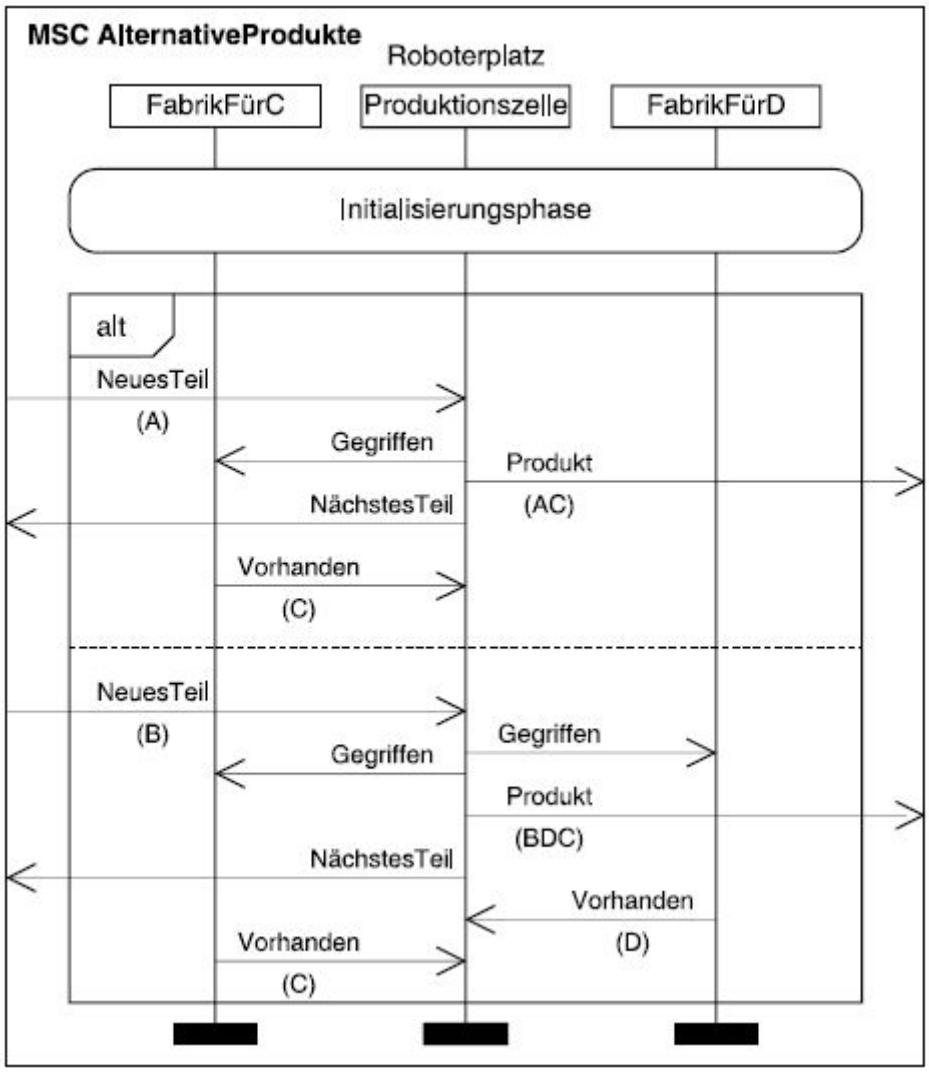

Abbildung 2.13: MSC mit strukturellen Sprachkonstrukten
Quelle : [3] Part 1: Message Sequence Chart (MSC)

2.3.2.1 Inline-Expressions

Inline-Expression werden als Operatoren in MSCs definiert. Damit können Teilabläufe, die innerhalb eines MSC-Diagramms spezifiziert worden sind, zu komplexeren Abläufen kombiniert werden. Sie erlauben es, die Wiederholung von Teilabläufen (loop-Operator), alternative Teilabläufe (alt-Operator), die parallele Komposition von Teilabläufen (par-Operator), optionale Teilabläufe (opt-Operator) und Ausnahmen in Form von Teilabläufen (exc-Operator) zu spezifizieren. Graphisch werden Inline-Expressions als Rechtecke mit in der linken oberen Ecke spezifizierten Namen dargestellt. Teilabläufe werden durch gestrichelten Linien in Sections separiert.

2.3.2.2 References

References ermöglichen es, MSCs in anderen MSCs wieder zu verwenden. Ein Reference unterscheidet und referenziert die andere MSC mit dessen Namen. Die References sind mit abgerundeten Rechteck dargestellt (z.b Ein Reference ist mit dem Name

Initialisierungsphase in Abbildung dargestellt).

2.3.3 High-Level-MSC

High-Level-MSC ist auch als (HMSC) bekannt. Es ermöglicht eine kombinierte Darstellung von mehren MSCs auf abstrakter Ebene. Diese Kombination ist durch gerichteten Graphen beschriebt. Die Knoten eines HMSC-Diagramms sind ein Anfangsknoten, Endknoten, Konnektoren, References und Conditions.
HMSC gibt ein globale Einblick von der gesamten Kombination von MSC-Diagrammen, damit kann man parallele, sequentielle alternative Kombination von MSCs in einer Einzige Abbildung und in einer sehr intuitiven Form beschreiben.
•

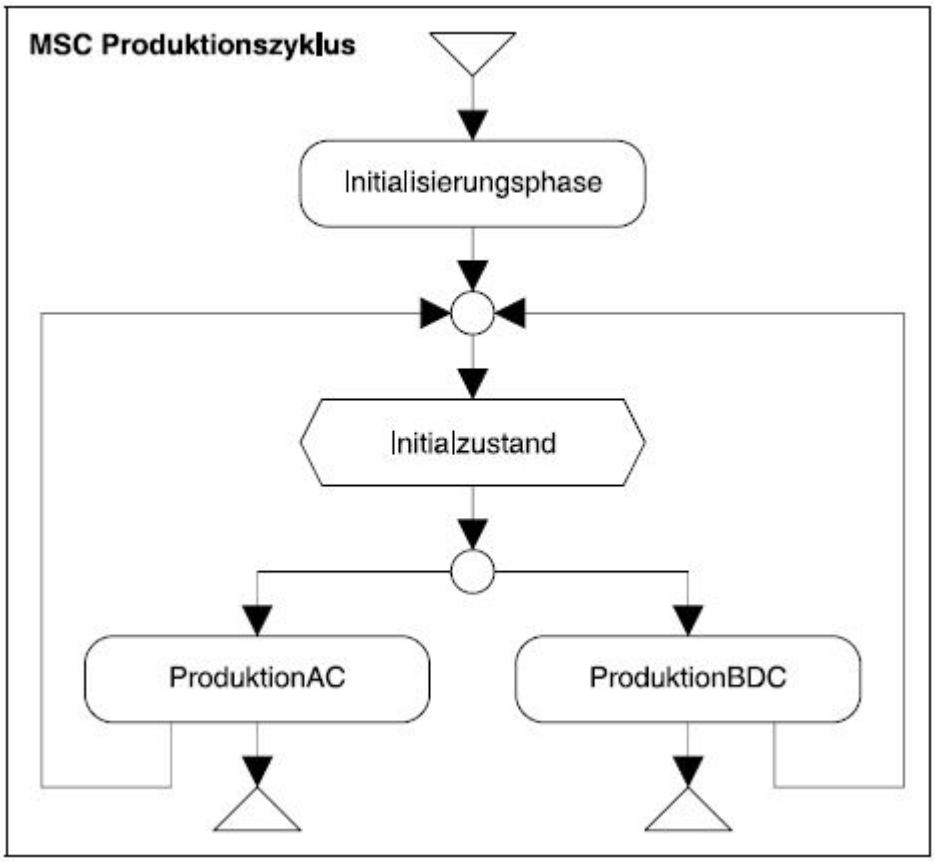

Abbildung 2.14: Ein High-Level-MSC

Quelle : [3] Part 1: Message Sequence Chart (MSC)
In der Abbildung beschreibt das HMSC-Beispiel, die verschiedene Szenarien, die im Produktionsprozess-Beispiel passieren können. Die verschieden Szenarien in dieser Abbildung sind:Am Anfang gibt es direkt die Initialisierungsphase,danach befindet sich der Produktionsprozess in seinem Initialzustand. Im Initialzustand können entweder Produkte vom Typ AC oder vom Typ BCD hergestellt werden. Nach der Herstellung eines Produkts befindet sich der Produktionsprozess wieder im Initialzustand und ein neues Produkt kann entweder gefertigt werden oder der Produktionsprozess endet.

2.3.4 Weitere Sprachkonstrukte

Wir haben in diesem Abschnitt dieses Kapitel nur an die sichtigsten Elemente der MSC-Sprache konzentriert. Damit haben wir ein globale aus ausreichende Einblick, um diese Modellierungssprache zu bewerten. Aber dazu gibt es noch weitergehende Konzepte, die MSC zu einer vollständigen Spezifikationssprache machen

3 Bewertungskriterien

In diesem Kapitel werden die für eine Eignung bzw. Uneignung benötigten Kriterien vorgestellt.
Die Bewertungskriterien sind notwendig, um eine Modellierugnssprache zu bewerten. Da die Modellierungssprachen unterschiedliche Eigenschaften, Spzifikationen und Benutzungszweck haben, lässt sich eine liste von verchiedene Kategorien der Kirtirien schwierig zu bestimmen.
Die Eignungskriterien treffen auf verschiedenste Arten von Modellierungssprachen zu. In diesem Abschnitt werden diese Kriterien in drei Hauptkategorie aufgeteilt: Formale-, Anwenderbezogene- und Anwendungsbezogene -Kriterien. Danach werden diese im Verlauf in weiteren Kategorien aufgeteilt. Die Bewertungen sind oft von dem Bewerter und dem Benutzer abhängig, z.b. kann jemand eine Modellierungssprache sehr einfach finden, wenn jemand anders sie schwierig findet, deswegen gibt es manchmal kein bestimmte Bewertungsverfahren und sichere Entscheidung, ob das Merkmal erfüllt ist.

Die Kriterien sind in drei Kategorien eingeteilt, auf welche im Folgenden eingegangen wird:

- Formale Kriterien
- Anwenderbezogene Kriterien
- Anwendungsbezogene Kriterien

3.1 Formale Kriterien

««««< HEAD Diese Kriterien dienen der maschinellen Prüfung von Modellen sowie der Berechnung von Modelleigenschaften. Werden die Anforderungen von der Modellierungssprache erfüllt, kann ein mit ihr geschaffenes Prozessmodell z. B. auf syntaktische Korrektheit überprüft werden. Die formalen Kriterien spielen eine besondere Rolle, wenn für die Modellierung Softwarewerkzeuge eingesetzt werden.[5]
Die formalen Kriterien werden in die folgende Einzelkriterien untergeteilt: Korrektheit, Vollständigkeit, Einheitlichkeit, Redundanzfreiheit, Strukturierbarkeit, Wiederverwendbarkeit und Wartbarkeit.

3.1.1 Korrektheit

Das Kriterium der Korrektheit steht in Verbindung zum Grundsatz der Richtigkeit der Grundsätze ordnungsgemäßer Modellierung. Das Merkmal hier erfüllt, wenn man einfach die unkorrekte Modelle identifizieren kann. Bei diesem Kriterium gibt es zwei unterschiedliche Ausprägung der Korrektheit und sie sind: syntaktische- sowie die semantische -eindeutige Identifikation fehlerhafter Modelle. Die syntaktische fehlerhafte Modelle sind eindeutig erkennbar ,d.h. die nicht eindeutig identifizierbar fehlerhafte Modelle sind korrekt. Der gleiche Fall ist bei der semantischen eindeutigen Identifikation fehlerhafter Modelle, wobei sind die semantische fehlerhafter Modelle eindeutig erkennbar, wenn die nicht eindeutig identifizierbar fehlerhafte Modelle richtig sind.

3.1.2 Vollständigkeit

Es geht hier um die Vollständigkeit der Sprachbeschreibung. Das Merkmal erfüllt, wenn alle benötigte Modelle mit dieser Modellierungssprache modellierbar sind. Ein vollständiges Modell kann erstellt werden, wenn alle Modelle und die Bedingungen ihrer Verwendung eindeutig und komplett definiert können.

3.1.3 Einheitlichkeit

Unter Vollständigkeit wird in diesem Zusammenhang die ähnliche Darstellung von den ähnlichen Konzepte verstanden. Um das Merkmal der Einheitlichkeit zu erfüllen, muss alle Konstrukte der Sprache verständlich dargestellt und beschrieben werden.

3.1.4 Redundanzfreiheit

Die Redundanzfreiheit setzt voraus, dass die Sprache keine Mehrfachdefinition von Konstrukten vornimmt, die denselben Sachverhalt beschreiben. Ein und derselbe Sachverhalt sollte also nicht mit mehreren verschiedenen Elementen bzw. Symbolen der Modellierungssprache belegt sein.[5]
Der Redundanzfreiheit bedeutet, dass es keine redundante Informationen im Modell gibt. Die unnötige redundante Informationen können der Benutzer ablenken und sie hilfen auf keinen Fall in der Modellbeschreibung, sonst sie bilden mehr Komplexität im Modell und man soll dafür mehr Zeit investieren, um dieses Modell zu verstehen. Nur wenn im Modele die Redundanzen vermieden werden, kann ein redundanzfreie Modell erstellt werde.

3.1.5 Strukturierbarkeit, Wiederverwendbarkeit und Wartbarkeit

das letzte Teil von den formalen Kriterien ist die Strukturierbarkeit, Wiederverwendbarkeit und Wartbarkeit, die miteinander verbunden sind. Dabei sollte die Modellierungssprache Konstrukte bereitstellen, welche die Strukturierung der modellierten Informationen unterstützt.
Die Modellierungssprache muss also Zerlegungen in Prozesskomponenten oder Teilprozesse darstellen können. Die Schnittstellen zwischen den Komponenten müssen übersichtlich dargestellt werden.[9] Um das Merkmal zu erfüllen, müssen die Generalisierungen und die Spezialisierung schlüssig nachvollziehbar sein.
Die Strukturierung in Teilmodelle ermöglicht gleichzeitig die Wiederverwendbarkeit der Strukturen in anderen Modellen. So können etwa modellierte Teilprozesse in anderen Prozessen wieder aufgegriffen werden und müssen nicht mehrfach modelliert werden.[5]Dazu ist die Wartbarkeit der Modelle durch die Strukturierung verbessert, weil die Modifikation des Modellteils verändert nicht die Konsistenz der übrigen Modellteile. Wie bei der Einheitlichkeit fördert das Kriterium der Strukturierbarkeit die Grundsätze ordnungsgemäßer Modellierung. ======= Diese Kriterien dienen der maschinellen Prüfung von Modellen sowie der Berechnung von Modelleigenschaften. Werden die Anforderungen von der Modellierungssprache erfüllt, kann ein mit ihr geschaffenes Prozessmodell z. B. auf syntaktische Korrektheit überprüft werden. Die formalen Kriterien spielen eine besondere Rolle, wenn für die Modellierung Softwarewerkzeuge eingesetzt werden.[5]
Die formalen Kriterien werden in die fünf Einzelkriterien Korrektheit, Vollständigkeit, Einheitlichkeit, Redundanzfreiheit und Strukturierbarkeit unterteilt.

3.1.6 Korrektheit

Eine Modellierungssprache genügt dem Kriterium der Korrektheit, wenn sie unzulässige Modelle eindeutig identifiziert und gleichzeitig erlaubt, die Menge aller zulässigen Modelle zu generieren. Zu unterscheiden ist dabei die syntaktische Korrektheit sowie die semantische Korrektheit der Modelle. Die semi-formalen Prozessmodellierungssprachen, welche für die Prozessdokumentation in Frage kommen, besitzen in der Regel keine eindeutig festgelegte Semantik, sodass hier vor allem die Möglichkeit der automatischen Überprüfung einer korrekten Syntax des Modells im Vordergrund steht. Das Kriterium der Korrektheit steht in enger Verbindung zum Grundsatz der Richtigkeit der Grundsätze ordnungsgemäßer Modellierung.[5]

3.1.7 Vollständigkeit

Unter Vollständigkeit wird in diesem Zusammenhang die Vollständigkeit der Sprachbeschreibung verstanden. Alle Konstrukte sowie die Bedingungen ihrer Verwendung, die von der Modellierungssprache bereitgestellt werden, müssen eindeutig definiert und beschrieben sein.[5]

3.1.8 Einheitlichkeit

Das Kriterium der Einheitlichkeit ist angelehnt an den Grundsatz der Klarheit der Grundsätze ordnungsmäßiger Modellierung. Einheitlichkeit bedeutet in diesem Kontext, dass alle Konstrukte der Sprache verständlich dargestellt und beschrieben werden. Ähnliche Konstrukte sollten somit auch in ähnlicher Weise spezifiziert werden.

3.1.9 Redundanzfreiheit

Die Redundanzfreiheit setzt voraus, dass die Sprache keine Mehrfachdefinition von Konstrukten vornimmt, die denselben Sachverhalt beschreiben. Ein und derselbe Sachverhalt sollte also nicht mit mehreren verschiedenen Elementen bzw. Symbolen der Modellierungssprache belegt sein. Auch dieses Kriterium fördert den Grundsatz der Klarheit. Indem gleiche real-weltliche Sachverhalte in der Modellierungssprache durch gleiche Konstrukte ausgedrückt werden, wird zudem der Grundsatz der Vergleichbarkeit gefördert.

3.1.10 Strukturierbarkeit

Da Informationsmodelle eine hohe Komplexität aufweisen können, sollte die Sprache Konstrukte bereitstellen, welche die Strukturierung der modellierten Informationen unterstützt. Die Modellierungssprache muss also Zerlegungen in Prozesskomponenten oder Teilprozesse darstellen können. Die Schnittstellen zwischen den Komponenten müssen übersichtlich dargestellt werden. Verallgemeinerungen (Generalisierung) und Detaillierungen (Spezialisierung) müssen schlüssig nachvollziehbar sein. Die Strukturierung in Teilmodelle ermöglicht gleichzeitig die Wiederverwendbarkeit der Strukturen in anderen Modellen. So können etwa modellierte Teilprozesse in anderen Prozessen wieder aufgegriffen werden und müssen nicht mehrfach modelliert werden. Zusätzlich wird dadurch die Wartbarkeit der Modelle verbessert, da Änderungen in einem Modellteil, die Konsistenz der übrigen Modellteile nicht beeinflussen. Das Kriterium der Strukturierbarkeit unterstützt den Grundsatz des systematischen Aufbaus der Grundsätze ordnungsgemäßer Modellierung »»»> 321950359661aae8443acada7239309cebbc5118

3.2 Anwenderbezogene Kriterien

Die anwenderbezogenen Kriterien beschreiben das Verhältnis zwischen dem Anwender und der verwendeten Modellierungssprache. Der Anwender kann zum einen der Modellierer und zum anderen der Betrachter eines Modells sein. Die anwenderbezogenen Kriterien besitzen eine besondere Bedeutung, um Nutzern von Modellen das Verständnis zu erleichtern bzw. zu ermöglichen. Für den betrachteten Anwendungsfall der Prozessdokumentation, zur Nutzung als allgemeine Arbeitsgrundlage für die Mitarbeiter einer Organisation, spielen sie daher eine sehr große Rolle. Die Modellierungssprache ermöglicht das Erstellen von Modellen, um Ideen und Gedanken zwischen den Beteiligten auszutauschen. Sie sind somit in erster Linie für den menschlichen Anwender als Kommunikationsmittel zu verstehen.[5]

3.2.1 Einfachheit und Erlernbarkeit

Das Kriterium der Einfachheit geht von folgendem Prinzip aus: Je einfacher eine Sprache ist, desto weniger Fehler sind bei der Modellierung mit ihr zu erwarten. Die Einfachheit der Modellierungssprache wird bestimmt durch eine geringe Anzahl an Notationselementen und Begriffen, sowie der Verwendung von einfachen Regeln für ihre Anwendung. Der Aspekt betrifft zum einen den Ersteller eines Modells, der bei einfachen Sprachen nur Kenntnis von wenigen Symbolen und Regeln haben muss, um Modelle zu erstellen. Wichtiger sind bei der Prozessdokumentation jedoch die Betrachter der Modelle. Dazu zählen nicht nur Fachexperten, sondern häufig einfache Arbeiter in den Unternehmen. Diese besitzen meist nur sehr eingeschränkte bis gar keine Methodenkenntnisse auf dem Gebiet der Prozessmodellierung. Trotzdem müssen die Modelle verstanden werden, wenn sie als Arbeitsgrundlage Verwendung finden sollen.
Eine Sprache, die für die Dokumentation von Prozessen verwendet wird, sollte nicht nur möglichst einfach sein, sondern auch den erforderlichen Schulungsbedarf auf einem akzeptablen Niveau halten. Vom Kriterium der Erlernbarkeit wird also ein möglichst geringer Aufwand zum Erlernen der notwendigen Konstrukte und Regeln der Modellierungssprache gefordert. Dies hängt wiederum stark von der Anzahl der Konstrukte der Sprache ab, sodass die Einfachheit und die Erlernbarkeit sehr stark korrelieren. Deshalb wurden sie in einem gemeinsamen Kriterium zusammengeführt. Das Kriterium der Einfachheit und Erlernbarkeit unterstützt den Grundsatz der Klarheit der GoM. Außerdem wird der Grundsatz der Wirtschaftlichkeit angesprochen, da insbesondere der Schulungsaufwand zum Nachvollziehen der Modellierungssprache und die Ausbildung zur Geschäftsprozessmodellierung in der Organisation einen erheblichen Kostenfaktor darstellen können.

3.2.2 Verständlichkeit

Die Modellierungssprache gilt als verständlich, wenn sie ein dem Nutzer bekanntes Vokabular benutzt, d. h. die definierten Konstrukte sollten im Wesentlichen mit Begriffen korrespondieren, die dem Anwender vertraut sind. Auf den Zweck der Prozessdokumentation übertragen bedeutet dies, dass die Symbole der Notation eher mit betriebswirtschaftlichen und allgemein bekannten Begriffen besetzt sein sollten, als etwa mit Begriffen aus der Softwareentwicklung oder anderen technischen Richtungen. Die Begriffe müssen dem Anwender aus seiner täglichen Arbeit bekannt sein, damit die Bedeutung leicht zu interpretieren ist. Auch die Verständlichkeit einer Modellierungssprache fördert die Qualität des damit abgebildeten Ergebnismodells und unterstützt somit den Grundsatz der Klarheit.

3.2.3 Anschaulichkeit

Während die Verständlichkeit eher auf die Benennung und verständliche Definition der Konstrukte ausgerichtet ist, umfasst das Kriterium der Anschaulichkeit die Forderung nach Struktur, Übersichtlichkeit und Lesbarkeit der Modelle, um den Grundsatz der Klarheit zu fördern. Die grafische Darstellung sollte möglichst intuitive Konzepte einsetzen um die Anschaulichkeit zu erhöhen. Dies kann z. B. die Verwendung von Piktogrammen sein, die real-weltlichen Objekten nachempfunden sind. Die Lesbarkeit sinkt, umso mehr verschiedene Notationselemente verwendet werden. Es wird davon ausgegangen, dass der Mensch durch das direkte Betrachten nur etwa sechs verschiedene Notationselemente unterscheiden kann. Die Struktur beschreibt, wie die Elemente in einem Modell gruppiert werden. So können z. B. Überschneidungen von Kanten die Anschaulichkeit eines Modells erheblich reduzieren.

3.3 Anwendungsbezogene Kriterien

Anwendungsbezogene Kriterien beschreiben die Anforderungen einer Modellierungssprache an eine Anwendung innerhalb eines Domänenspezifischen Einsatzes. Dies hängt von den jeweiligen Aspekten ab, welche durch die Modellierungssprache abgebildet werden soll. So besitzen verschiedene Modellierungssprachen unter anderem ein unterschiedlich großes Nutzungspotenzial in der jeweiligen Anwendungsdomäne. Man spricht in diesem Zusammenhang auch von der Mächtigkeit der Sprache. Dabei ist zu beachten, das Anwendungs- und Anwenderbezogene Kriterien oft konkurrierend sein können, so kann sich beispielsweise eine leicht erlernbare Sprache sich negativ auf die Mächtigkeit auswirken und umgekehrt. Anwendungsbezogene Kriterien können in Anforderungen wie der Angemessenheit, der Mächtigkeit, der Operationalisierbarkeit, und der Überprüfbarkeit beschrieben werden. Zwar gibt es höchstwahrscheinlich noch eine weit aus größere Anzahl an Kriterien, diese sollen uns aber in dieser Arbeit genügen [2, S. 95 f.].

3.3.1 Angemessenheit

Die Angemessenheit einer Sprache bezieht sich auf die Gesamtheit der Sprache, als auch auf die einzelnen Konzepte der Sprache und umfasst damit das Abstraktionsniveau, den Detaillierungsgrad und den Formalisierungsgrad einer Sprache [6, S. 97].
Die Abstraktionsfähigkeit einer Modellierungssprache beschreibt, worin die Fachterminologie der Domäne sich mit den Konzepten der Modellierungssprache möglichst decken sollte und welche unwesentlichen Details abstrahiert werden. In dem Bereich von Kommunikationsabläufen der Telekommunikation sind diese Begriffe eher technischer Natur und kommen aus einem Informatik-spezifischem Umfeld, welcher Fachsprache auf dem Niveau von Spezialisten voraussetzt.
Der Detaillierungsgrad beschreibt wie Sachlich angemessen detailliert die Konstrukte eines Anwendungszwecks der Domäne sich darstellen lassen können. Dies beinhaltet die zu Modellierenden Modelle, Daten und zusätzlichen Informationen. Damit ist nicht gemeint, dass ein System in einer hohen Abstraktionsform eine eins zu eins Nachbildung aller technischen Prozesse des Informationssystems ausdrücken soll, sondern nur für die Zielgruppe angemessene und verständliche Konzepte. Eine formale Beschreibung der Prozesse ist demnach nicht gewünscht, da der Hauptaugenmerk auf dem Anwender liegt.
Die Angemessenheit der Sprache ist abhängig durch den Anwendungszweck. Da aber der Sachverhalt zumeist unklar definiert ist, ist es nicht immer möglich zu überprüfen ob ein Modell einen Sachverhalt abbildet. Dies ist nicht der Fall, wenn der Sachverhalt exakt

definiert ist und eine formale Überprüfung der Angemessenheit vorgenommen werden kann.
Genauso steht sie somit in direktem Zusammenhang mit der Mächtigkeit, einem weiterem Kriterium, welche später in der Arbeit behandelt wird und dem Anwendungszweck einer Sprache. So bieten Angemessenheit und Mächtigkeit zusammen die Möglichkeit, die bereitgestellten Konzepte einer Sprache bezogen auf den Anwendungszweck zu untersuchen. Demnach sollen ausreichende Konzepte bereitgestellt werden, ohne jedoch überflüssige Konzepte zu enthalten [11, 35F].

3.3.1.1 Konsequenz

Im Hinblick ob ein Konstrukt der Sprache nun Angemessen ist oder nicht, hängt subjektiv vom jeweiligen Betrachter ab. Dies ist insbesondere bei komplizierten Konstrukten Fall. Im Zweifelsfall muss nun entschieden werden, ob es ein notwendiges Sprachkonstrukt darstellt oder ob es nur die Sprachkomplexität erhöht. Dies muss jedoch für jedes einzelne Konstrukt abgewägt werden, da jedes seine eigenen spezifischen Vor- und Nachteile besitzt. Laut Ulrich Frank und Micheal Prasse ist es sogar gar nicht möglich, ein formelles Maß zu Bestimmung von Angemessenheit von Modellierungssprachen zu besitzen, da die Untersuchung eines Konstrukts der Sprache immer nur subjektiv erfolgen kann, man jedoch aber die Bedeutsamkeit des Konstrukts im jeweiligen Anwendungsfalls beurteilen kann [11, S. 35].

3.3.2 Mächtigkeit

Die Mächtigkeit ist ein maß des Nutzungspotenzials einer Sprache und gibt Aussage darüber, in wie weit und wie gut die Konzepte der verwendeten Sprache, die Eigenschaften eines Sachverhalt der Domäne darstellen kann. Darunter fällt wie Präzise diese Aussagen sind und wie hoch der Detailgrad der darzustellenden Eigenschaft ist [1, S. 180]. Der Sprachumfang korreliert mit der Mächtigkeit der Sprache. Je größer dieser Umfang ist, desto größer ist auch die Mächtigkeit der Sprache. Da es für den Anwender von großer Bedeutung ist, seine Anwendung mit einem möglichst umfänglichen Detailgrad beschreiben zu können, muss die Mächtigkeit mindestens alle Aspekte enthalten, die für den gewünschten darzustellenden Sachverhalt notwendig sind. Eine Mächtigkeit der Sprache, die sich über den Anwendungszweck der Domäne bezieht, kann wie schon erwähnt sich negativ auf andere Kriterien auswirken. Ferner jedoch, kann man die Mächtigkeit einer formalen Modellierungssprache in der Automatentheorie anhand ihres Grammatik-Typen festlegen. Eine Modellierungssprache sollte sich auf die Modellierung konzentrieren. So können zu viele Konzepte zu einer unangemessen mächtigen Sprache führen, was zwangsläufig ihre Komplexität erhöht. Zwischen Mächtigkeit und Angemessenheit muss daher ein ausgewogenes Verhältnis existieren.

3.3.2.1 Konsequenz

Mächtigkeit und Angemessenheit bedingen einander. Die Mächtigkeit einer Sprache muss daher relativ zur Anwendungsdomäne betrachtet werden. Geht man grundsätzlich davon aus, dass die hier betrachteten Modellierungssprachen der konzeptuellen Modellierung dienen, misst sich die Mächtigkeit einerseits an der Möglichkeit, als wesentlich erkannte Sachverhalte des abzubildenden Realitätsbereichs natürlich, also ohne aufwendige Rekonstruktionen, beschreiben zu können. Andererseits sollte die Modellierungssprache auch Möglichkeiten bieten, Informationen darzustellen, die für die Implementierung benötigt werden.

3.3.3 Operationalisierbarkeit

Die Operationalisierbarkeit gibt Aussage darüber, ob und wie gut sich die Modellierungssprache über ihre eigene Verwendung hinaus noch weiter verwenden lässt. Darunter fällt die Transformation des Modells in andere Sprachen als auch das darstellen von diversen Sachverhalten. Bei der Transformation ist hierbei zu beachten, dass die verwendeten Konzepte beider Sprachen möglichst gleich sein müssen, um eine annähernd vollständige Konvertierung zu ermöglichen. Zur Abbildung diverser Sachverhalte enthält eine gute Operationalisierbarkeit die Abbildungsmöglichkeit von Funktionen, Bedingungen, Ressourcen, Objekten und Ereignissen. Deswegen müssen Konzepte zur Planung und Analyse des domänenspezifischen Einsatzes enthalten sein [2, S. 95 f.].

3.3.4 Überprüfbarkeit

Die Überprüfbarkeit eines Modells einer Modellierungssprache bezieht sich auf die Realitätsnähe eines abzubildenden Sachverhalts. Es ist also ein Maß der Abweichung eines Modells, bezogen auf die Realität welches es abbilden soll. Ein Modell sollte also immer möglichst genau, überprüfbar und nachvollziehbar an der Realität übergreifend für die Betrachter liegen. Dabei muss jedoch berücksichtigt werden, dass objektive Verfahrensweisen für ein Modell nicht vollumfänglich anwendbar sind, da es sich meist um geplante und nicht faktischen Realitäten handelt. Dadurch gegeben obliegt diese Einschätzung dann den Betrachter des Modells. Sollten diese einen Konsens über die geplanten Erwartungen erreichen, so kann ein Modell als realistisch eingeschätzt werden [4, S. 3]. Bei der objektive Überprüfung ist die Semantik einer Sprache zu prüfen, dabei ist zwischen den formalen und informalen Sprachen zu unterscheiden. Da ein Modell eine abstrakte Abbildung eines Sachverhalts darstellt, werden in ihm nur die wesentlichen Eigenschaften berücksichtigt und nicht Sachbezogene Eigenschaften ausgelassen. Abbildung 29 zeigt die Beziehung zwischen Modell und abzubildender Sachverhalt. Sollte ein Modell nicht

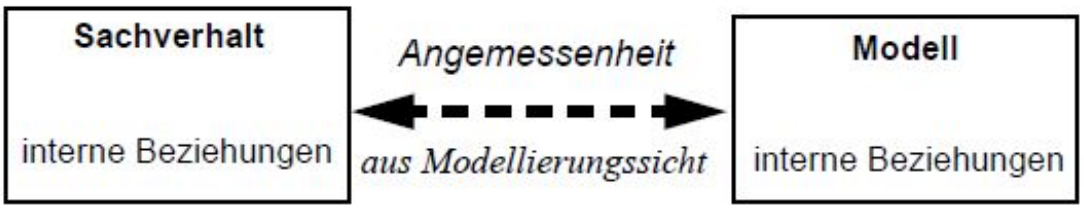

Abbildung 3.1: Beziehung zwischen Modell und zu modellierendem Sachverhalt, Quelle : [11, S. 35]

formal definiert sein, so ist die einzige Möglichkeit, den Inhalt dessen zu erfassen und mit dem gegebenen Sachverhalt entgegen zu prüfen. Je nachdem wie exakt und präzise ein Modell interpretiert werden kann, desto leichter kann es mit dem abzubildenden Sachverhalt verglichen werden. Es ist zwar prinzipiell nicht zu entscheiden, ob ein unpräziser Sachverhalt und ein präzises Modell adäquat sind, jedoch hilft die Präzession und Genauigkeit eines Modells bei dem Vergleich des abzubildenden Sachverhalts. Eine Modellierungssprache, die es ermöglicht, Modelle exakt und eindeutig zu beschreiben und zu interpretieren, erleichtert daher die Überprüfung des Modells wesentlich. Ist das Modell eindeutig, so ist auch ein umfängliches übergreifendes Verständnis möglich. Dies bedeutet aber auch, das jedes Sprachkonstrukt und Sprachverhalten eindeutig interpretiert werden muss. Ansonsten kann dies wegen Interpretationsschwierigkeiten zu Missverständnissen

führen. Deswegen muss eine abstrakten Syntax und Semantik vorausgesetzt werden, um die Interpretation der Modelle zu vereinheitlichen [11, 35F].

4 Bewertung der Eignung und Uneignung der Modellierungssprachen

In diesem Kapitel werden die Modellierungssprachen aus Abschnitt 2.1, 2.2 und 2.3 anhand der definierten Kriterien aus Kapitel 3 auf ihre Eignung bewertet. Das Ergebnis soll Anwendern unterstützen eine Entscheidung bei Ihrer Auswahl anhand der Eignung bzw. Uneignung der Modellierungssprache zu treffen.

4.1 UML

UML ist heute eine der dominierenden Sprachen für die Modellierung von betrieblichen Anwendungs- bzw. Softwaresystemen. Der erste Kontakt zu UML besteht häufig darin, dass UML-Diagramme im Rahmen von Softwareprojekten zu erstellen, zu verstehen oder zu beurteilen sind. UML-Diagramme gelten als Standard bei objektorientierter Modellierung.

4.1.1 Eignung

Aus meiner persönlichen Sicht liegen die wesentlichen Vorteile bei dem Einsatz von UML in der breiten Unterstützung sämtlicher objektorientierter Grundsätze. Man kann an vielen Stellen ein und dasselbe Problem auf unterschiedliche Art und Weise lösen, ohne sich dabei durch die Vorgaben der Sprache eingeengt zu fühlen. Die Sprache lässt also den Softwareentwicklern den Freiraum, nach eigenen Vorstellungen zu modellieren. Besonders Vorteilhaft erscheint die Tatsache, dass UML auch die Erweiterungsmöglichkeiten durch eigene Sprachkonstrukte vorsieht (Metamodell), und damit wirklich jedem das Recht anbietet, den eigenen Notationsrahmen zu erschaffen. Diese Möglichkeit sollte aber nur in äußersten Notfällen (wenn es nicht anders geht) verwendet werden, da man sonst der Gefahr, sich vom Standard zu entfernen, entgegenläuft .
Weiterer Vorteil liegt in der Standardisierung der UML. Die UML bietet eine fast perfekte Grundlage für die Kommunikation der verschiedenen Entwicklungsteams miteinander. Diagramme können nun nicht nur von den Fachleuten der Software –Firma , sondern auch von den außerhalb stehenden verstanden und verbessert werden. Lästiges Einarbeiten in die verschiedenen Notationen entfällt, falls sich alle grundsätzlich an die UML halten. Weiterer Vorteil liegt in der zunehmenden Verbreitung der Unified Modeling Language. Die exakten Zahlen sind zwar noch schwer abzuschätzen, es zeichnet sich jedoch ein wachsender Trend für den Einsatz der UML ab.
Man kann die Vorteile von UML in vier wichtigste Punkte zusammenfassen wie folge:

- Die Vereinheitlichung der Terminologie und die Standardisierung der Notation führen zu einer massiven Erleichterung der Verständigung zwischen allen Beteiligten.
- Die UML wächst mit Ihren Anforderungen an die Modellierung. Sie können mit der Erstellung einfacher Modelle beginnen, aber auch sehr komplexe Sachverhalte im Detail modellieren, da die UML eine mächtige Modellierungssprache ist.
- Die UML baut auf bewährten und weit verbreiteten Ansätzen auf. Die UML wurde nicht im Elfenbeinturm erstellt, sondern hat sich zu grossen Teilen aus der Praxis und

aus bestehenden Modellierungssprachen heraus entwickelt. Das gewährleistet die Einsatzfähigkeit und Praxisnähe der UML.

4.1.2 Uneignung

Die Nachteile lassen sich nun wiederum aus dem Umfang der Sprache ableiten. UML ist sehr vielfältig, so dass auch am Anfang sehr viel Aufwand für das Aneignen und Verstehen sämtlicher Sprachkonstrukte aufgebracht werden muss.
Außerdem wird man in der Regel feststellen, dass viele Elemente sehr selten zum Einsatz kommen und damit eher als Ballast der Sprache angesehen werden. So habe ich in der Darstellung der UML Notation auf die Erläuterung der OCL Konstrukte (Object Constraint Language) verzichtet, obwohl OCL als formale Sprache zur Erweiterung der Semantik der UML Notation herangezogen werden kann. Damit lassen sich zwar Zusicherungen, Invarianten, Vor –und Nachbedingungen, Navigationspfade etc. kurz und aussagekräftig darstellen, aber man kann diese im begrenzten Maße auch durch UML Basiselemente beschreiben.
Speziell für die Entwicklung der Software für eingebettete Systeme lässt sich anmerken, dass UML eine objektorientierte Modellierungssprache ist. Sämtliche Konzepte von UML können nur dann ausgenutzt werden, wenn wirklich objektorientiert programmiert wird. Es macht wenig Sinn, objektorientiert zu modellieren , wenn anschließend kein objektorientierter Code verwendet wird. Steht kein objektorientierter Compiler zur Verfügung, sollte man sich über den Einsatz anderer Werkzeuge und Spezifikationssprachen Gedanken machen. Man kann zwar im begrenzten Maße sämtliche objektorientierte Ansätze in den „strukturierten“ Code konvertieren (also z.B. von C++ nach C überführen), das Ergebnis lässt jedoch zu wünschen übrig, insbesondere leidet die Code- Qualität / Lesbarkeit darunter.

Man kann die Nachteile von UML in vier wichtigste Punkte zusammenfassen wie folge:
- Die neue Notation muss zuerst erlernt werden. Es fallen somit Schulungskosten an und die Mitarbeiter müssen dafür Zeit aufwenden können.
- Im Bereich der Software-Unterstützung ist noch einiges zu tun, bis wirklich benutzerfreundlich mit UML gearbeitet werden kann.
- Durch die Vereinheitlichung der Modellierungssprachen geht die Vielfalt und Kreativität verloren. Kleine, vielleicht gute Modelle gehen verloren.
- Der Wettbewerb wird durch die Monopolisierung zerstört.

4.1.3 Fazit

Wir versuchen in diesem Abschnitt die Vorteile und Nachteile von UML zusammenfassen und einen Überblick darüber geben.

Eignung	Uneignung
Standardisierte, weit verbreitete Methode	Grundsätzliches IT-Wissen erforderlich
Ergebnisse der Modellierung in UML können zur Umsetzung in Software-Code verwendet werden. Es ist eine automatisierte Erzeugung eines Coderahmen aus den Diagrammen möglich wodurch der Programmier-Aufwand reduziert wird	Sehr großer Umfang und Komplexität der Sprache (beispielsweise umfasst die Sprachspezifikation mehr als 1200 Seiten) verursacht Schwierigkeiten bei der Benutzung und erfordert einen hohen Einarbeitungsaufwand
Einfacher Modellaustausch möglich und damit bessere Modellwiederverwendung	Die UML wird hinsichtlich semantischer Inkonsistenzen, Konstruktmehrdeutigkeiten, inadäquaten Notationen und kognitiven Unzugänglichkeiten kritisiert
Einheitliche Toolunterstützung	Die UML wurde ursprünglich für die Entwicklung von Softwaresystemen konzipiert, daher sind die relevanten Beschreibungs- und Darstellungsmittel weniger auf die Prozessmodellierung ausgerichtet
Modellierungsmittel für die meisten Probleme im Software Engineering	Konzeptionelle Schwächen, im Requirements Engineering insbesondere bei der Formulierung von Anwendungsfällen
Breites Angebot an Werkzeugen, Büchern und Schulung	UML-Modelle sind Sammlungen von Einzelmodellen: Konsistenzprobleme, Problem des Zusammensuchens relevanter Information
einheitliche Notation für alle Aspekte	
Referenzen und Beziehungen möglich	
Erweiterungsmöglichkeit mittels Profile (Stereotypen/Tags) möglich	

Abbildung 4.1: Eignung und Uneignung von UML

4.2 SDL

SDL ist eine alte und häufig verwendete Modellierungssprache zum spezifizieren und beschreiben von verteilten Kommunikationssystemen. Es schließt die Lücke zwischen Spezifikation und Implementierung, da sie Modellierung auf einem abstraktem Level ermöglicht und eine detaillierte Beschreibung der Implementation liefert. SDL verwendet ein explizites Nachrichtenkonzept, welches zwar einen erhöhten Aufwand bei Modellerstellung und -pflege erfordert, dafür aber fehlerrobuster ist.

4.2.1 Eignung/Uneignung von SDL

Da SDL eine formale, objektorientierte Modellierungssprache ist, deckt sie somit die Formalen Kriterien der Korrektheit aus **??**, sowie die der Strukturierbarkeit aus **??** ab. Sowohl die Syntax als auch die Semantik der einzelnen Konstrukte und Konzepte werden präzise

definiert und bieten somit keinen Interpretationsfreiraum. Es sichert Konsistenz und Klarheit über das Modell und eignet sich daher perfekt für kritische Kommunikationssysteme in denen harte Anforderungen gestellt werden. Ihre Strukturierbarkeit verdankt SDL vor allem den Neuerungen aus Version SDL-2000, welche die Sprache um objektorientierte Aspekte erweitert hat.
Durch die Konzepte der Objektorientierung wie Kapselung, Vererbung, Polymorphismus und dynamischem Binden, kann das Modell in Teilmodelle modularisiert werden und außerdem die Wiederverwendbarkeit als auch die Wartbarkeit gesteigert werden. Dadurch eignet sich SDL seitdem auch für die Entwicklung in mehreren größeren Teams.
Die Vollständigkeit von SDL wird durch ihre Spezifikation gezeigt, in der für Ersteller und Leser alle vorkommenden Sprachbausteine und Konzepte vollumfänglich beschrieben werden. Was im Umkehrschluss bedeutet, dass keines der Konzepte nicht beschrieben wurde. Vollständigkeit sollte im Allgemeinen und besonders in Kommunikationssystemen als kritische Voraussetzung zur Eignung betrachtet werden, da eine nicht Beschreibung immer zu Inkonsistenz in der Interpretation führt.
Das Kriterium der Einheitlichkeit ist ebenso vorhanden, wie das der Redundanzfreiheit. Ersteres kann man anhand der Definition der einzelnen Konstrukte in der Spezifikation nachvollziehen und letzteres is nach betrachten der Spezifikation augenscheinlich redundanzfrei. Es sind also keine Doppeldefinition für verschiedene Konstrukte enthalten.
Die frage nach der Einfachheit und Erlernbarkeit von SDL ergibt sich aus der unseres Erachtens geringen Anzahl an Notationselementen und Begriffen, die in der Sprachdefinition spezifiziert sind. So definiert SDL seine Agenten als gewisse Typen, gängige in der Fachterminologie verwendete Abstrakte Datentypen, Signale und Kanäle für die Kommunikation, sowie Sprachkonzepte der Objektorientierung. Dadurch lässt sich die Sprache mit verhältnismäßig wenig Aufwand erlernen und mögliche Schulungsmaßnahmen können gering gehalten werden.
SDLs Sprachkonzept bedient sich an vielen technischen Bezeichnungen aus der Telekommunikation und bezieht somit seinen Sprachraum aus dem informatisch, technischen Bereich. Dies ist dem geschuldet, da SDL von Telekommunikationsfirmen entwickelt und verabschiedet wurde um fachliche Kontexte aus der Telekommunikationsindustrie zu vereinheitlichen. Da wir nicht sehen, dass ein unfachlicher Sprachumfang für eine Fachperson als verständlich angesehen werden kann, so kann auch die Modellierungssprache als verständlich angesehen werden. SDL eignet sich nicht für Fachfremde Benutzer, da für diese die Konzepte und Sprachmittel der Sprache nicht verständlich sein dürften.
Die Konstrukte welche SDL in der Spezifikation erfasst sind, haben eine strukturierte Gliederung und heben sich untereinander durch ihre grafische Darstellung ab. Sie sind einfach gehalten und nur auf das nötigste begrenzt. Sie bestehen aus 2-Dimensionalen Piktogrammen und haben jeweils ihre eigene Form. Durch die Verwendung von objektorientierten Konzepten und der hierarchischen Gliederung wird die Lesbarkeit stark gefördert.
SDL eignet sich aufgrund seiner Herkunft und durch die Personen, welche an der Spezifikation beteiligt waren hervorragend für die Modellierung von Problemfeldern in der Telekommunikationsbranche, da sich die Fachterminologie der Branche mit den Konzeptbezeichnungen der Modellierungssprache decken. Sie eignet sich eher nicht für die allgemeine Definition von Modellierungen in anderen Bereichen wie beispielsweise der Datenmodellierung in Datenbankensystemen.
Da die SDL eine formale Modellierungssprache mit abstrakter Syntax und kontextfrei ist, kann sie in der Chomsky-Hierarchie als eine Typ-2 kontextfreie Grammatik eingestuft werden. Dies würde die Mächtigkeit ihrer Sprache in der Automatentheorie beschreiben. Es ist also nicht entscheidbar ob ein Konstrukt der Sprache Eindeutig oder Mehrdeutig

definiert ist. Da aber durch Testverfahren beim vergleichen der jeweiligen Konstrukte keine Mehrfachdefinition gefunden wurde, kann dies ausgeschlossen werden. Relativ zur Anwendungsdomäne betrachtet deckt die Spezifikation unter anderem Protokollspezifikationen in Kommunikationssystemen ab. Auch hier ist aber wieder die Mächtigkeit in Bezug auf eine beliebig andere Anwendungsdomäne nicht gegeben, da hier die nötigen Sprachkonstrukte und Vorschriften zur Abbildung fehlen.

4.2.2 Fazit

Kategorie	Eignung	Uneignung
Formale Kriterien		
Anwenderbezogene Kriterien	Verwendung einer schwer verständlichen formalen axiomatischen Notation.	
Anwendungsbezogene Kriterien	Aufwendige aber fehlerrobuste Modellierung von Kommunikation durch Nachrichtenkanäle.	

4.3 MSC

MSC wurde von derselben Arbeitsgruppe wie SDL spezifiziert und wird häufig in Verbindung mit anderen Modellierungssprachen verwendet. Dort wird sie unter zur Anforderungsspezifikation, Testfallspezifikation und Dokumentation verwendet. Durch ihre Flexibilität, kann sie in den unterschiedlichsten Anwendungsdomänen verwendet werden, da sie nicht explizit für eine einzige zugeschnitten wurde. Hauptvorteil von MSC ist dessen grafisches Aussehen, welches intuitiv die Beschreibung des Systemverhaltens verständlich macht. Der Hauptaugenmerk liegt dabei bei dem Austausch von Nachrichten über Kommunikationseinheiten wie verteilten Systemen.

4.3.1 Eignung

MSC ist wie SDL eine formale Modellierungssprache und deckt damit das Formale Kriterium der Korrektheit aus Abschnitt 25 ab.

4.3.2 Fazit

MSC wird in folgenden Feldern verwendet

5 Fazit

Eine strukturierte Liste von Beurteilungskriterien lässt sich daraus schwer ableiten. Dazu sind Evaluationen von Modellierungssprachen ein schwieriges Unterfangen. Dies liegt zum einen in der Komplexität des Untersuchungsgegenstandes begründet. Zum anderen ist es problematisch, objektiv bewertbare Qualitätskriterien zu identifizieren: Qualitätskriterien für Modellierungssprachen machen sich eben nicht allein an einer formalen Syntax und Semantik oder einer Notation fest. Stattdessen ist es von entscheidender Bedeutung, das Verhältnis der Sprache zu den Modellierern und dem Modellierungszweck zu bewerten. Dies bedeutet, dass die Bewertungskriterien von Fall zu Fall unterschiedlich stark ins Gewicht fallen und dass die Bewertung der einzelnen Kriterien von Fall zu Fall ebenfalls unterschiedlich ist. Daraus folgt, dass die Evaluation kaum zu 100 Prozent objektiv sein kann.
Dies muss sie aber auch nicht sein: Eine Modellierungssprache muss „lediglich“ – dies ist aber bereits schwer genug zu erreichen – zu den Modellierern und dem Modellierungszweck passen. Es schadet für die Praxis nicht, persönliche Präferenzen – und diese fließen fast zwangsläufig in die praktische Evaluation ein – zu beachten. Den Modellierern muss die Sprache gefallen, denn sie müssen damit kreativ und sicher umgehen können.
Dieser Bezugsrahmen kann und soll nicht den Anspruch auf Vollständigkeit erheben. Dennoch stellt er eine geeignete – durchaus durch eigene weitere Kriterien erweiterbare – Grundlage für ein Evaluationsprojekt dar, denn es findet sich in diesem Bezugsrahmen eine große Anzahl von Anforderungen an Modellierungssprachen, die in einer Evaluation Beachtung finden sollten. Die Evaluation sollte jedoch von erfahrenen Modellierern durchgeführt werden, die dazu in der Lage sind zu bewerten, in wie fern die Anforderungen erfüllt sind, die die Anforderungen in ihrer Wichtigkeit bewerten, und gegebenenfalls eine Auswahl treffen, welche Kriterien innerhalb eines Evaluierungsprojekts zu untersuchen sind, und welchen für einen gegeben Modellierungszweck und Modellierungsumfang weniger Beachtung geschenkt werden muss.
Mit dem Bezugsrahmen ist aber nicht nur ein Beitrag zur Evaluation von Modellierungssprachen geleistet, sondern auch zur Entwicklung oder Weiterentwicklung derselben. Die Anforderungen können sukzessive durchgearbeitet und geeignete Konzepte in die Modellierungssprache aufgenommen werden: Dieser Bezugsrahmen soll damit durchaus auch die Kreativität der Entwickler von Modellierungssprachen fördern.

Literatur

[1] Thomas Allweyer. *Maßgeschneiderter Methodeneinsatz für die Entwicklung betriebswirtschaftlicher Software*. Springer Berlin Heidelberg, 2005.

[2] Jane Fröming. *Ein Konzept zur Simulation wissensintensiver Aktivitäten in Geschäftsprozessen*. GITO-Verlag, Dez. 2009. ISBN: 978-3940019851.

[3] Ekkart Rudolph und Michael Schmitt Jens Grabowski. *Die Spezifikationssprachen MSC und SDL. Teil 1: Message Sequence Chart (MSC)*. 2001. URL: https://www.swe.informatik.uni-goettingen.de/sites/default/files/publications/at-sdl-0202.pdf,.

[4] Oliver Vering Jörg Becker Wolfgang Probandt. *Grundsätze ordnungsgemäßer Modellierung*. Springer -Verlag Berlin Heidelberg, 2012. ISBN: 978-3-642-30411-8.

[5] Christopher Lobe. *Bewertung der Eignung von Modellierungssprachen zur ebenenübergreifenden Prozessdarstellung im Managementhandbuch*. Otto-von-Guericke-Universität Magdeburg, 2015. URL: http://bauhaus.cs.uni-magdeburg.de:8080/miscms.nsf/FEA8C8150500AA14C1257449004F79A9/A066435FF04D9DBCC125820B0068A184/$FILE/Masterarbeit%20Christopher%20Lobe.pdf,.

[6] Christopher Lobe. „Bewertung der Eignung von Modellierungssprachen zur ebenenübergreifenden Prozessdarstellung im Managementhandbuch“. Magisterarb. Otto-von-Guericke-Universität Magdeburg, Nov. 2015.

[7] Prozessmanagement in MV. *Ausgewählte Modellierungs-Notationen im Überblick*. DVZ Datenverarbeitungszentrum Mecklenburg-Vorpommern GmbH, 2011. URL: http://www.cio.m-v.de/static/CIO/Dateien/KE/Prozessmanagement/Anlage_Modellierungsnotationen_im_Ueberblick_V03.pdf,.

[8] *OMG® Unified Modeling Language® (OMG UML®) Version 2.5.1*. Techn. Ber. OMG® Unified Modeling Language®, Dez. 2017. URL: https://www.omg.org/spec/UML/2.5.1/PDF.

[9] Fleischmann Albert Obermeier Stefan Fischer Herbert und Dirndorfer Max. *Geschäftsprozesse realisieren*. Springer Vieweg, 2014. ISBN: 978-3-8348-2303-8.

[10] Roger Süttenbach und Jürgen Ebert. *A Booch Metamodel*. Universität Koblenz-Landau, 1997. URL: http://www.uni-koblenz.de/~ist/documents/Suettenbach1997ABM.pdf,.

[11] Michael Prasse Ulrich Frank. *Bezugsrahmen zur Beurteilung objektorientierter Modellierungssprachen- veranschaulicht am Beispiel von OML und UML*. Techn. Ber. Institut für Wirtschaftsinformatik - Universität Koblenz Landau, 1997. URL: https://www.wi-inf.uni-duisburg-essen.de/FGFrank/documents/Arbeitsberichte_Koblenz/Nr6.pdf.

[12] International Telecommunication Union. *Message Sequence Chart (MSC)*. Techn. Ber. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, Feb. 2011. URL: https://www.itu.int/rec/T-REC-Z.120-201102-I/en.

[13] International Telecommunication Union. *Specification and Description Language - Overview of SDL-2010*. Techn. Ber. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, Apr. 2016. URL: https://www.itu.int/rec/T-REC-Z.101/en.

[14] International Telecommunication Union. *Specification and Description Language – Basic SDL–2010*. Techn. Ber. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, Apr. 2016. URL: https://www.itu.int/rec/T-REC-Z.100.

[15] International Telecommunication Union. *Specification and Description Language – Comprehensive SDL-2010*. Techn. Ber. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, Apr. 2016. URL: https://www.itu.int/rec/T-REC-Z.102/en.

[16] International Telecommunication Union. *Specification and Description Language – Data and action language in SDL-2010*. Techn. Ber. TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU, Apr. 2016. URL: https://www.itu.int/rec/T-REC-Z.104/en.

[17] Christian Weber. „Entwurf und Implementierung eines konfigurierbaren SDL Transpilers für eine C++ Laufzeitumgebung“. Magisterarb. Technische Universität Kaiserslautern, Sep. 2008. URL: https://vs.informatik.uni-kl.de/en/publications/2005/We05/ConTraST.pdf.

Abkürzungsverzeichnis

ADT Abstract Data Type

ASN1 Abstract Syntax Notation One

EFSM Extended Finite State Machine

FIFO First In First Out

ITU International Telecommunication Union

ITU-T ITU Telecommunication Standardization Sector

MSC Message Sequence Chart

SDL Specification and Description Language

TTCN3 Testing and Test Control Notation Three

UML Unified Modelling Language

Abbildungsverzeichnis

Tabellenverzeichnis

Listings