- Systementwurf -

Use-Case Visualizer

Version: 1.0

Projektbezeichnung	Use-Case Visualizer	
Projektleiter	Nicolas Hanke	
Verantwortlich	Dennis Ruß	SW-Architekt
Erstellt am	16.05.2018	
Zuletzt geändert	24.05.2018 13:34	
Bearbeitungszustand		in Bearbeitung
	X	vorgelegt
		fertig gestellt
Dokumentablage	AI-SWP\02_Entwurf\Systementwurf(Use-Case_Visualizer).rtf	

Änderungsverzeichnis

Änderung			Geänderte Kapitel	Beschreibung der Änderung	Autor	Zustand
Nr.	Datum	Version				
1	16.05.18	1.0	Alle	Initiale Produkterstellung	D. Ruß	In Bearbeitung

Prüfverzeichnis

Die folgende Tabelle zeigt einen Überblick über alle Prüfungen – sowohl Eigenprüfungen wie auch Prüfungen durch eigenständige Qualitätssicherung – des vorliegenden Dokumentes.

Datum	Geprüfte Version	Anmerkungen	Prüfer	Neuer Produktzustand

Inhalt

1 Einleitung

Dieses Dokument soll ein Grundverständnis der Systemstruktur vermitteln ohne den Entwurf bis in letzte Einzelheiten darzulegen. Das Grundverständnis soll jedoch ausreichen, um sich ggf. anhand des Quellcodes in weitere Einzelheiten leicht einarbeiten zu können.

Kernthemen in diesem Dokument sind:

- Übersicht über die Zerlegung des Systems: Welche (größeren) Systemkomponenten gibt es? Wofür ist jede einzelne davon zuständig? Wie hängen diese Komponenten voneinander ab?
- Schnittstellenübersicht: Welche Schnittstellen stellt das System und jede Systemkomponente für seine/ihre Umgebung bereit?
- Systemkomponenten: Wie ist jede Systemkomponente aufgebaut?
- Designabsicherung: Zeigt für ausgewählte „architektur-relevante“ Use-Case-Szenarien, dass und wie diese mit dem gewählten Systementwurf realisierbar sind.

Der Systementwurf wird auf Grundlage der funktionalen und nicht-funktionalen Anforderungen sowie des konzeptuellen Datenmodells gewonnen, etwa indem man für ausgewählte „architektur-relevante“ Use-Case-Szenarien untersucht, welche Teile des Systems zur Realisierung in welcher Weise zusammenarbeiten müssen.

Die Gliederung dieses Dokuments orientiert sich grob am Aufbau der V-Modell-XT®[1]-Produkte „System-Architektur“ und „SW-Architektur“, ist jedoch für die Verwendung in der Veranstaltung **„Software-Projekte“** im Studiengang **„Angewandte Informatik“** der **OTH-Amberg-Weiden** verändert worden und nicht konform zum V-Modell-XT.

2 Architekturprinzipien und Entwurfsalternativen

Entwurfsalternativen:

- **Modulübergreifende Änderungen zum Datenmodell**

Die Klasse „UseCaseFile“ aus dem konzeptionellen Datenmodell wurde verworfen, da sie sich als unnötig erwiesen hatte. Die allgemeine „File“ - Klasse ist für das Einlesen ausreichend. Es muss keine gemeinsame Klasse für Dateien erstellt werden.

- **Übergabe der Daten vom XML-Parser zur internen Graph-Repräsentation**

Entwurfsalternativen:

1. Übergabe in einem Dictionary
   (mehrere „Alternative Flows“ können schlecht zugegriffen werden (gleicher Name in einem Dictionary nicht erlaubt/Key muss eindeutig sein…)
2. Übergabe als Liste von Tupeln
   (Datentypen von Tupeln können variieren (String, Tupel, etc.))
3. Übergabe der "einfachen" String Parameter über Properties und der Flows über Methoden

[1] V-Modell® ist eine geschützte Marke der Bundesrepublik Deutschland.

Ergebnis:
Es wird die Alternative 3 verwendet. Jeweilige Begründung siehe oben.

- **Weitergabe der Graph-Daten**

Rahmenbedingungen:
Es ist zu klären, wie andere Komponenten der Anwendung Daten von der UseCase-Klasse erhalten sollen, nachdem der Graph erstellt wurde.

Entwurfsalternativen:
1. Jede Klasse, die Daten braucht erhält eine Referenz auf den UseCase vom Controller
2. Der UseCase verbreitet seine Daten über Events an alle, die die Daten brauchen
   (jeder muss dafür eine Referenz auf den UseCase haben, also geht auch die Alternative 1)
3. Der UseCase verbreitet seine Daten über ein Messaging-System an alle, die die Daten brauchen
   (es existiert noch keine Notwendigkeit für ein globales Messaging-System)
4. Der UseCase kennt alle, die Daten brauchen und übergibt diese direkt
   (zu starke Kopplung (Klassenbindung))

Ergebnis:
Es wird die Alternative 1 verwendet. Begründung siehe oben.

- **Flow Repräsentation**

Rahmenbedingungen:
Es ist zu klären, ob es eine Klasse „Flow“ für jeden Typ von Flow gibt oder ob es eine gemeinsame Basisklasse gibt, von der jeder Flow-Typ abgeleitet ist.

Entwurfsalternativen:
1. Jeder Flow-Typ hat die gleiche Klasse (Flow), bei der teilweise Vorhersagen über die Struktur abhängig vom Typ getroffen werden können (Basic-Flow hat zum Beispiel keine Reference-Flows)
2. Jeder Flow Typ hat seine eigenen Klasse, die von Flow abgeleitet ist.

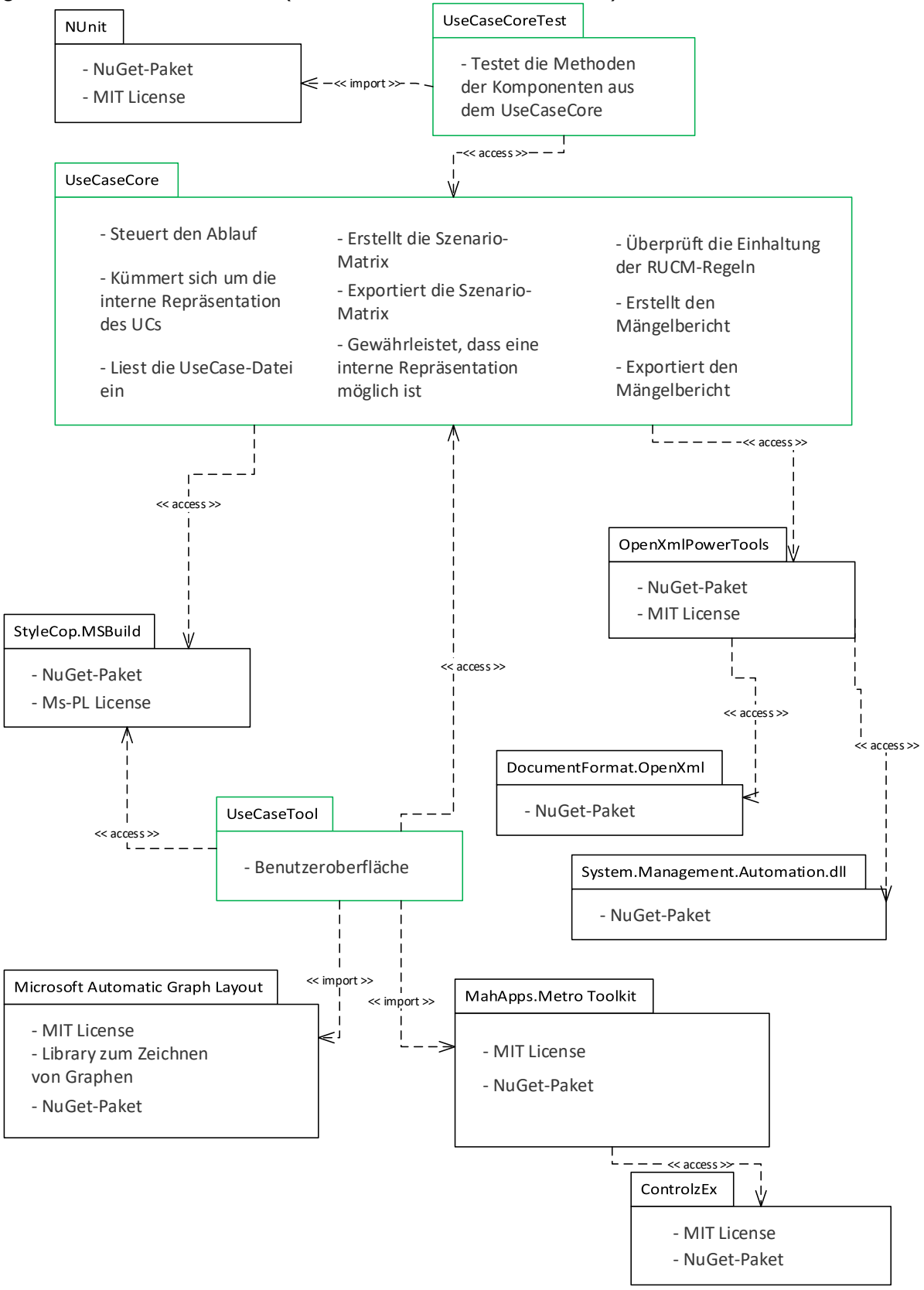
Ergebnis:
Es wird die Alternative 1 verwendet. Die Entscheidung wurde getroffen, da der Typ nur die Belegung einzelner Eigenschaften variiert, die für alle Flow Typen allerdings haben können. Weiter reduziert diese Alternative den Entwicklungsaufwand.

Architekturprinzipien

- Oberfläche (*subscriber*) „abonniert“ Controller (*publisher*) (observer pattern)

3 Übersicht über die Zerlegung des Systems

Neben den beiden Hauptpaketen („UseCaseCore“ und „UseCaseTool“) werden folgende Pakete verwendet (GRÜN bedeutet selbsterstellt):



4 Schnittstellenübersicht

Hier folgt eine kurze Schnittstellenbeschreibung einer jeden Komponente. Methoden, die eine Klasse einer anderen Klasse derselben Komponente zur Verfügung stellt, werden nicht berücksichtigt. Auch werden hier keine Methoden aus den Klassendiagrammen berücksichtigt, die mit „Get“ bzw. „Set“ beginnen, da diese später als Properties umgesetzt werden.

Controller	
CurrentXmlFilePath(string): void	Can be called to tell the system path and file name of the XML UseCase file. The string parameter contains the path to the new file.
MatrixFilePath(string): void	Can be called to tell the system path and file name for the export of the scenario matrix and to trigger the export. The parameter string contains the path under which the new file should be stored.
ReportFilePath(string): void	Can be called to tell the system path and file name for the export of the defect report and to trigger this. The parameter string contains the path under which the new file should be stored.
ChangeCycleDepth(uint): void	Can be called to tell the system a new cycle depth for the scenario matrix. The scenario matrix is then redrawn. The uint parameter specifies the cycle depth to be used.

XMLStructureParser	
LoadXmlFile(path : string) : bool	Loads external (word) xml file, which is stored on a storage medium. The absolute path to the file must be passed as parameter. Returns true if file was read sucessfully. Returns false if file was not read sucessfully.
GetError() : string	If error (return value = false) has occurred at function “LoadXmlFile()” or “TryToFixMalformedXml()”, the error text can be read out.
TryToFixMalformedXml() : bool	If external xml file contains easy structural errors (for example missing bracket on line 50), it can be tried to repair automatically. Attention: Source file on storage medium will be overwritten! Returns true if file was repaired successfully. Returns false if file was not repaired sucessfully.

ParseXmlFile(out UseCase : UseCase) : bool	Analyzes the previously read xml file. Returns true if file was analyzed sucessfully. Returns false if file was not analyzed sucessfully.

IRucmRuleValidator	
Validate(flowToCheck : Flow, referencedBasicFlow : Flow):bool	Validates one flow against all the RUCM-Rules. For each violation an IError is added to the ErrorReport. If the flowToCheck is the „Basic Flow“ no „Reference Flow“ has to be passed. If the flowToCheck is an „Alternative Flow“, the referencedBasicFlow has to be passed in ordert o validate it properly. Returns true if there was no violation found, otherwise false.
AddExternalError (errorToAdd : string) : void	Method can be called to add an external GeneralError to the ErrorReport, e.g. if the XML file could not be read. The errorToAdd contains the message that should be added.
GetErrorReport(): ErrorReport	Can be used to get the actual ErrorReport. Returns the existing ErrorReport instance.
Export(path : string) : bool	Can be called to export the ErrorReport in a CSV-file. The path parameter specifies the path where to put the file. Returns true if the ErrorReport was exported without any problems, otherwise false.

UseCase	
SetBasicFlow(steps : List<string>, postcondition : string) : void	Sets a basic flow for the use-case. It consists of a list of steps numbered by their position in the list and a postcondition as string.
AddSpecificAlternativeFlow(id:int, steps:List<string>, postcondition:string, referenceStep : ReferenceStep)	Adds a specific alternative flow with an unique id (unique for all specific alternative flows added to this use-case), a list of steps numbered by their position in the list, a postcondition and a reference step that describes the step from which

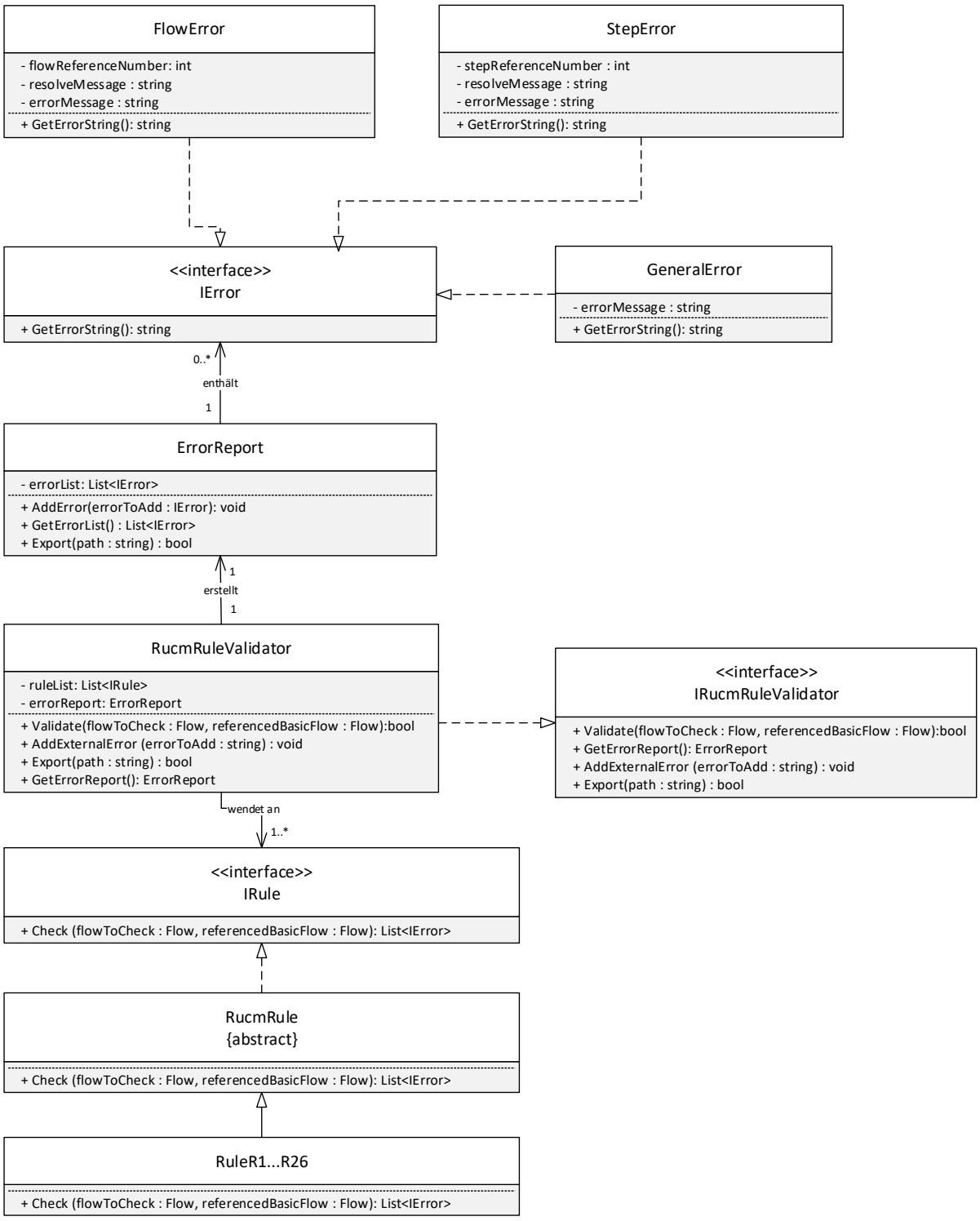
	this alternative flow may start.
AddGlobalAlternativeFlow(id : int, steps : List<string>, postcondition : string)	Adds a global alternative flow with an unique id (unique for all global alternative flows added to this use-case), a list of steps numbered by their position in the list and a postcondition.
AddBoundedAlternativeFlow(id : int, steps : list<string>, postcondition : string, referenceSteps: List<ReferenceStep>) : void	Adds a bounded alternative flow with an unique id (unique for all bounded alternative flows added to this use-case), alist of steps numbered by their position in the list, a postcondition and a list of reference steps that describe the steps from which this alternative flow may start.
BuildGraph() : void	Signales the end of the description of the use-case and builds the internal graph-representation. Before calling BuildGraph only the setters and methods starting with "Set" and "Add" are allowed to be called. After using these methods to describe the use-case, BuildGraph is called and the given description is used to build a graph representation. After calling BuildGraph only the getters are allowed to be called.

ScenarioMatrix	
ChangeCycleDepth(newCycleDepth:int):void	Changes the max. allowed cycledepth for scenarios. When changed the list of scenarios gets refreshed
Export(path:string):bool	Exports the Scenario Matrix as a Word-Document to a given path. Returns true if successfull.
Initialize(useCase:UseCase):bool	Starts the initial calculation of scenarios by giving a reference to a Use-Case object.
GetScenarios():List<Scenario>	Returns the list of possible scenarios for the Use Case. Returns an empty list if no scenarios found.

5 Systemkomponenten

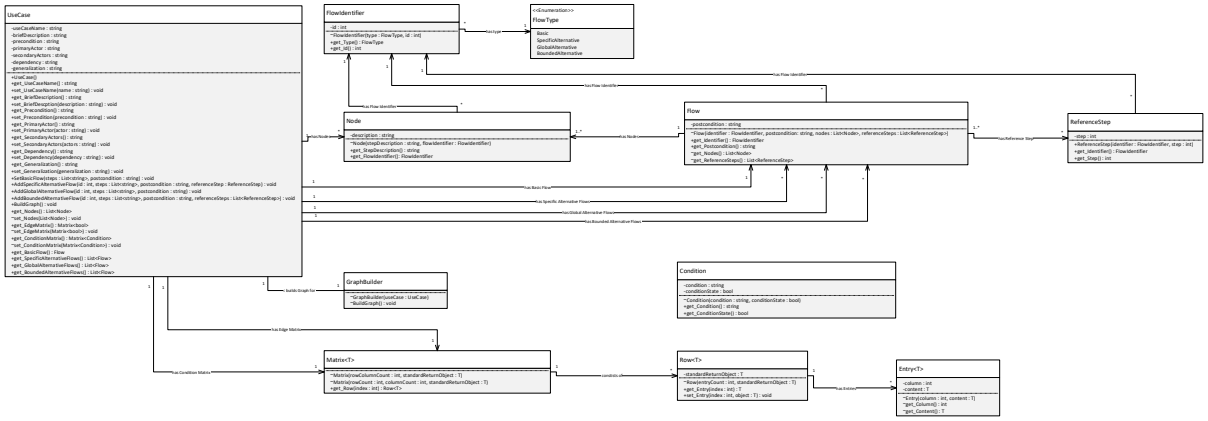
- Komponente: RuleValidation

Als Schnittstelle zu anderen Modulen dient das Interface IRucmRuleValidator.

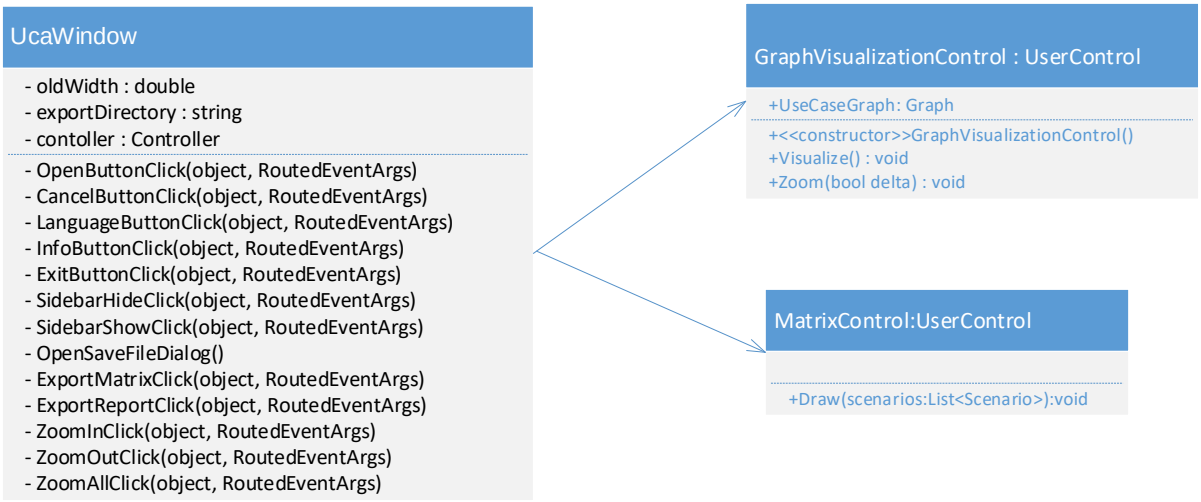


- Komponente: UC intern

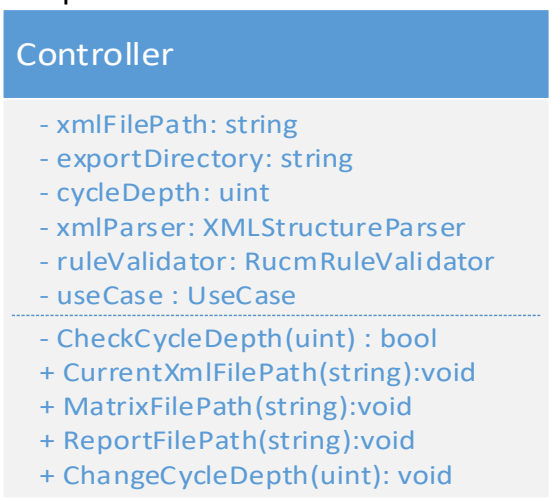
Das Original ist zu finden unter:
AI-SWP\02_Entwurf\UC-Intern\Klassendiagramm_UC-Intern.vsdx



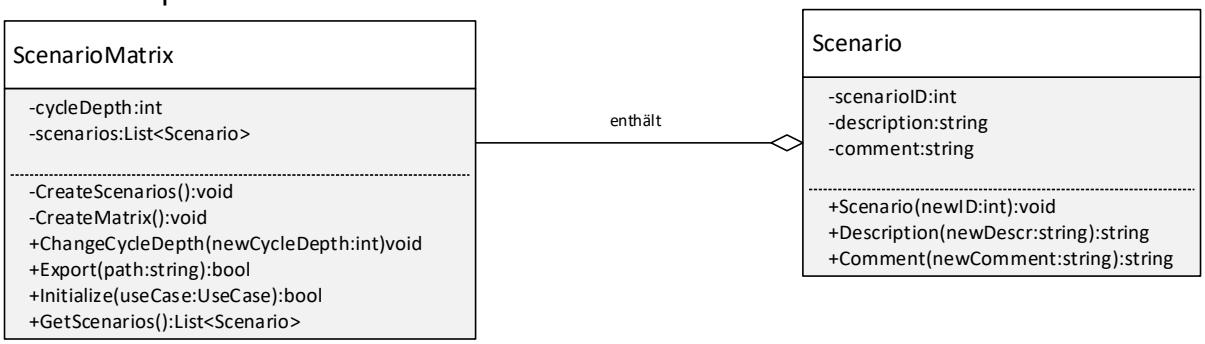
- Komponente: Benutzeroberfläche



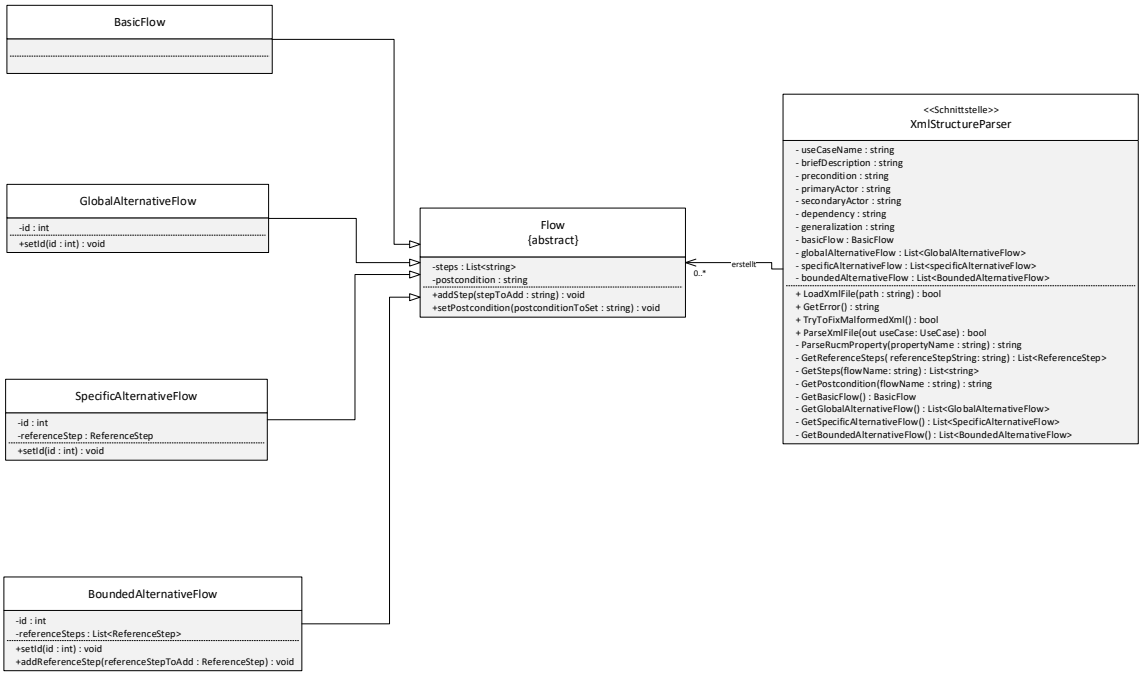
- Komponente: Controller



- Komponente: Szenario-Matirx



- Komponente: UC-Einlesen



6 Designabsicherung

Use-Case: UC-Beschreibungsdatei einlesen UND Use-Case: UC validieren

Auch hier befinden sich die Original-Dateien unter dem jeweils angegebenen Pfad

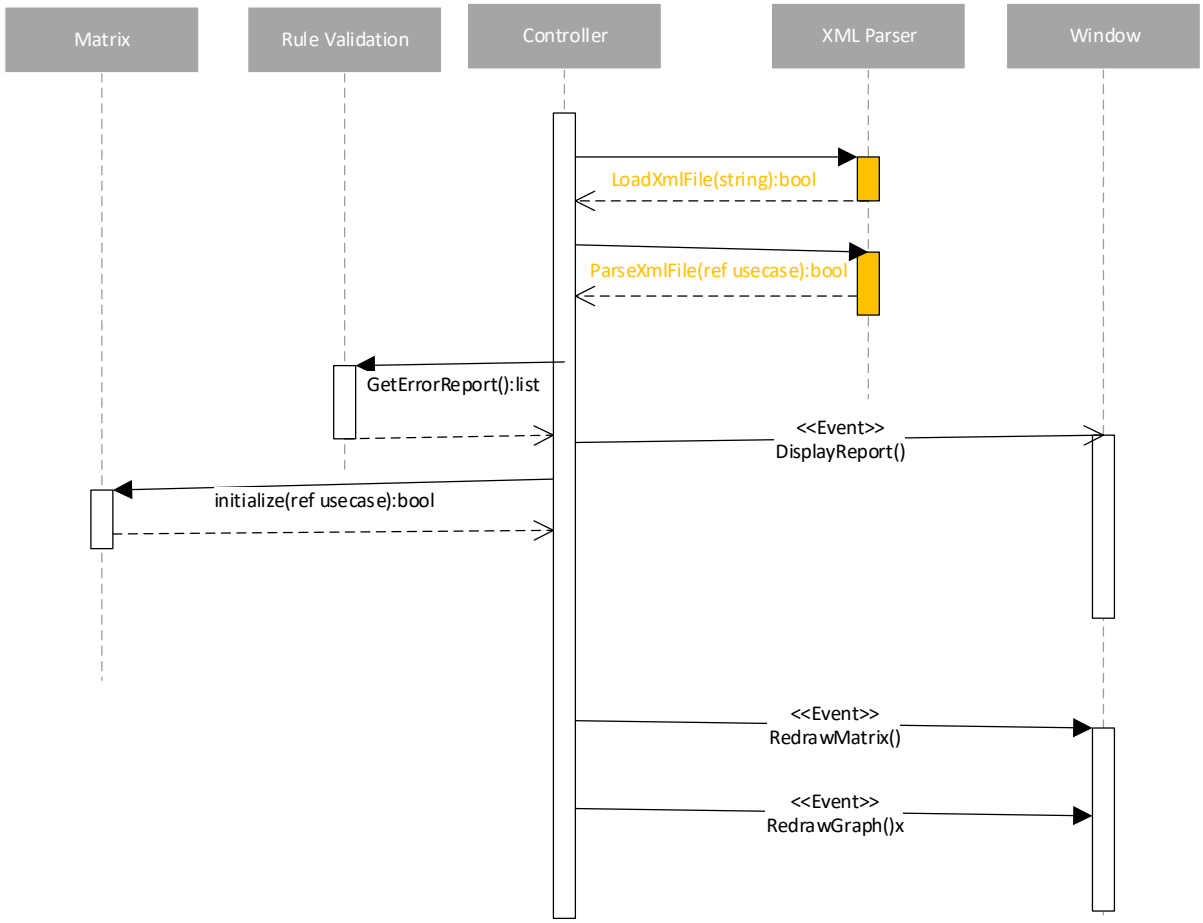


Abbildung 1: AI-SWP\02_Entwurf\Controller_View\Sequenzdiagramme Felix.vsdx

Im Folgenden wird der Lade- und Parse-Vorgang (Kennz. ORANGE) genauer dargestellt.

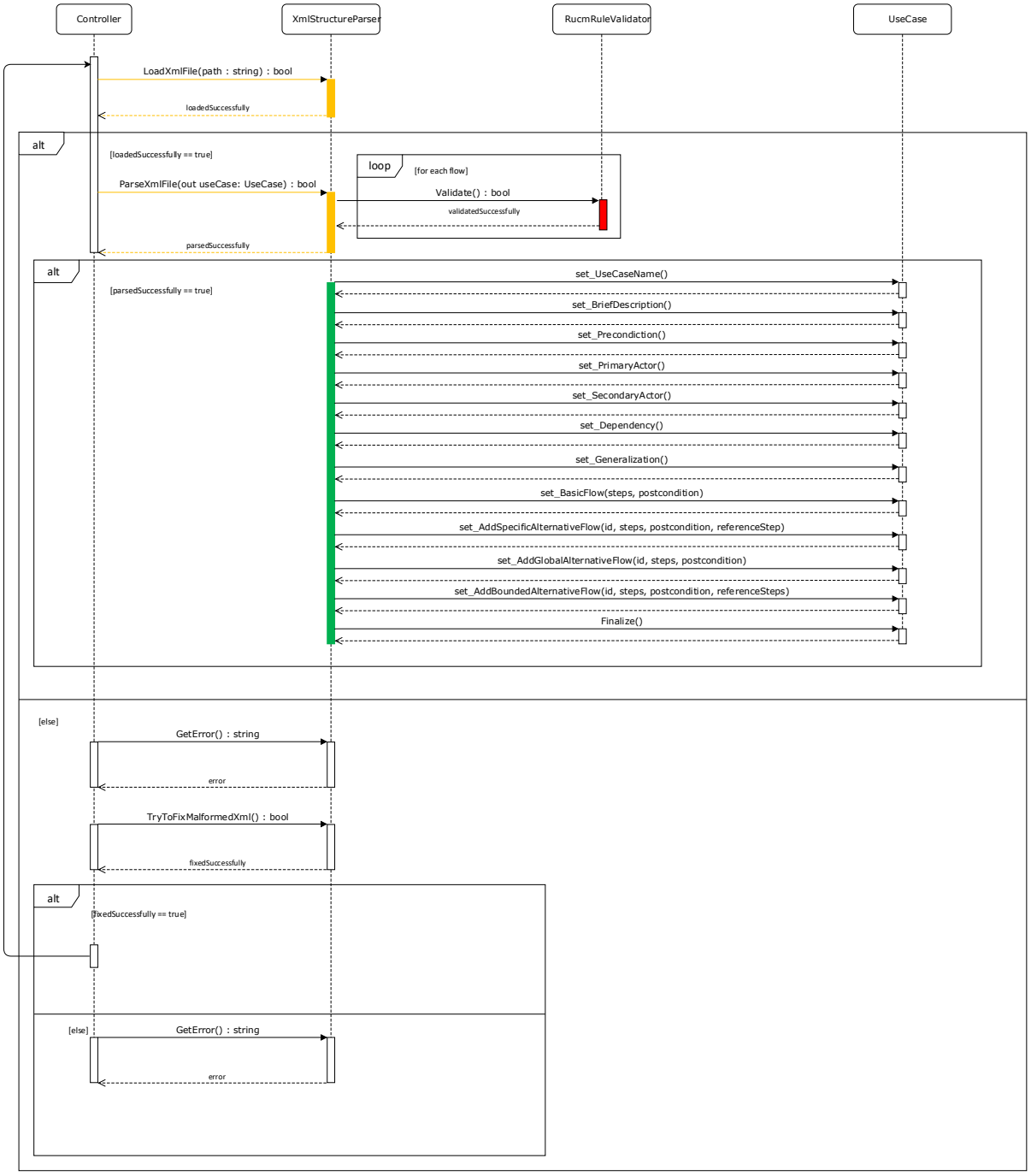


Abbildung 2: AI-SWP\02_Entwurf\UC_Einlesen\Sequenzdiagramm_UseCaseEinlesen_Validieren.vsdx

Im Folgenden wird der Validierungsvorgang (Kennz. ROT) genauer dargestellt.

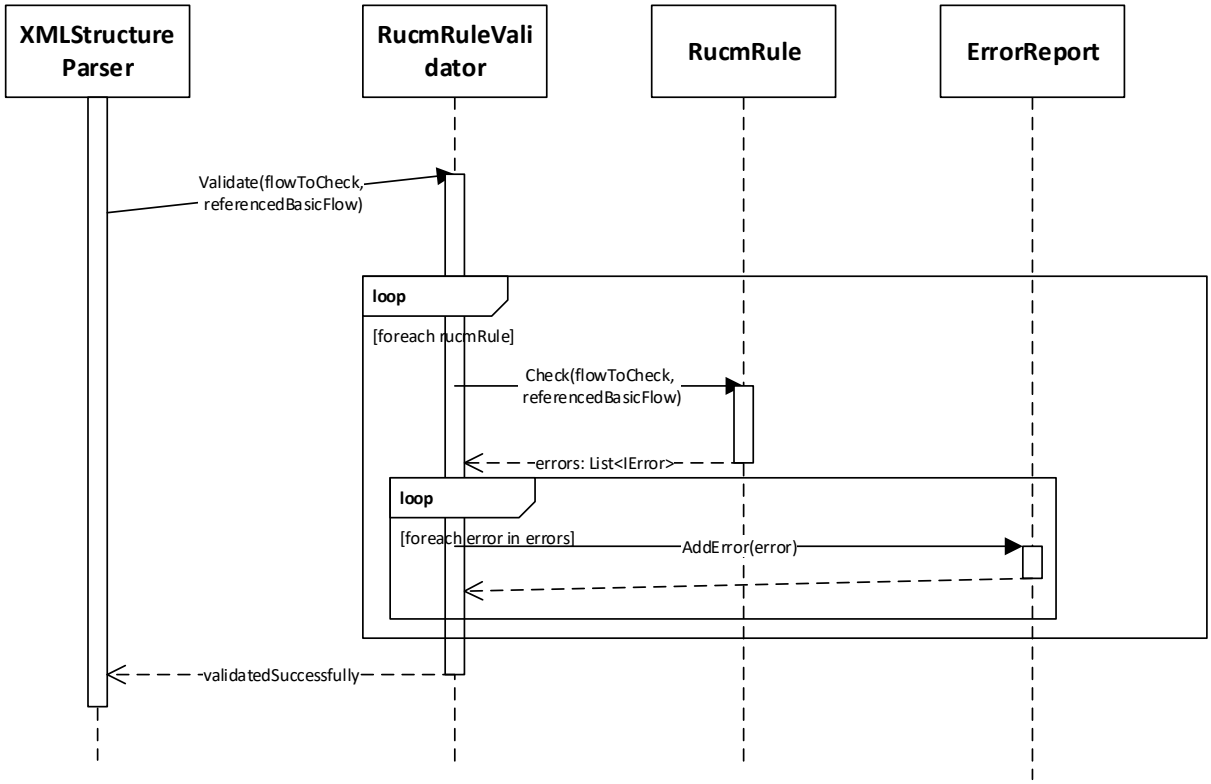


Abbildung 3: AI-SWP\02_Entwurf\RuleValidation\Sequenz_Validierung.vsdx

Im Folgenden wird interne Darstellungsvorgang (Kennz. GRÜN) genauer dargestelllt.

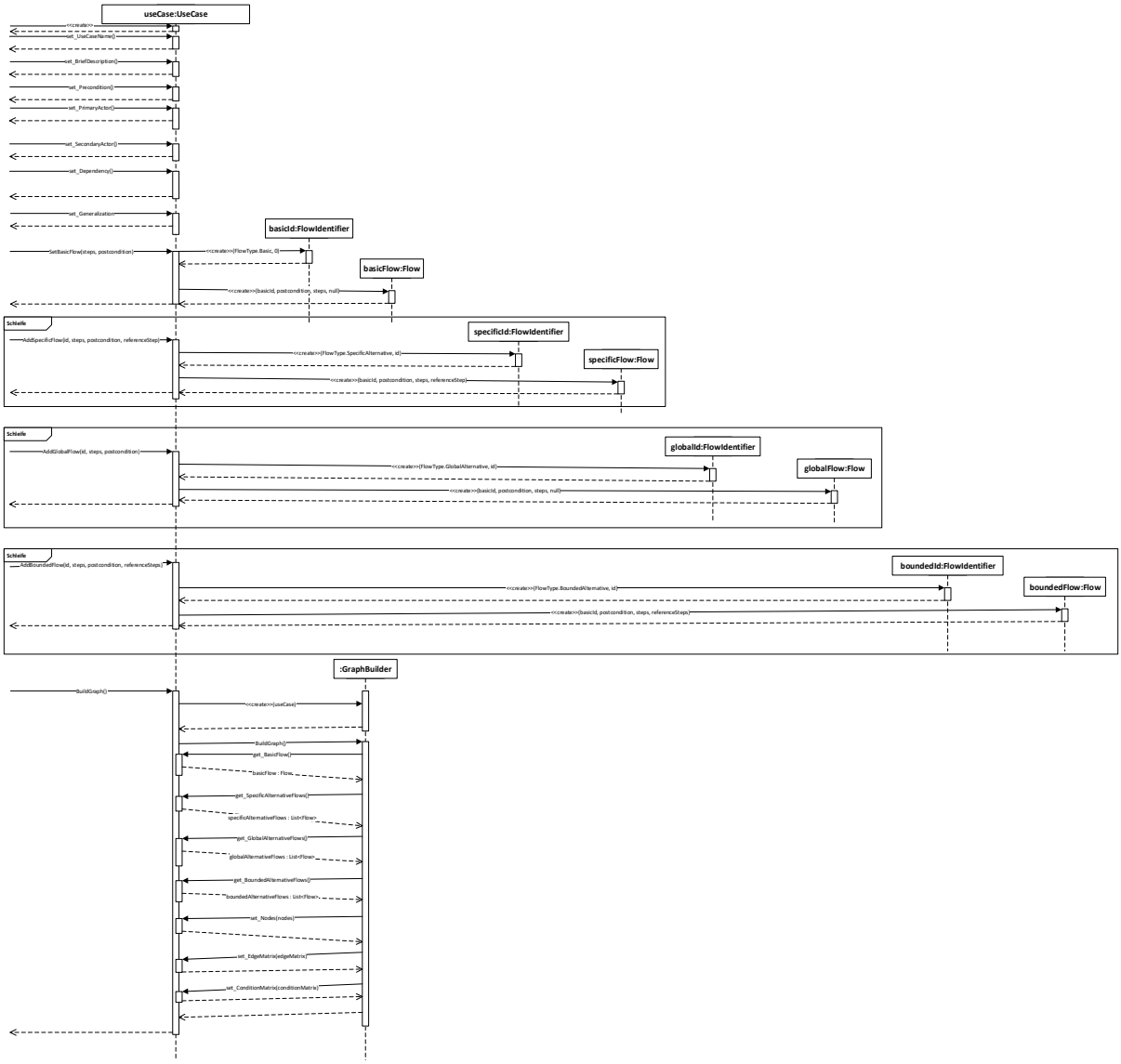


Abbildung 4: AI-SWP\02_Entwurf\UC-Intern\Sequenzdiagramm_UC_Validieren_UC-Intern.vsdx

Use-Case: Mängelbericht exportieren

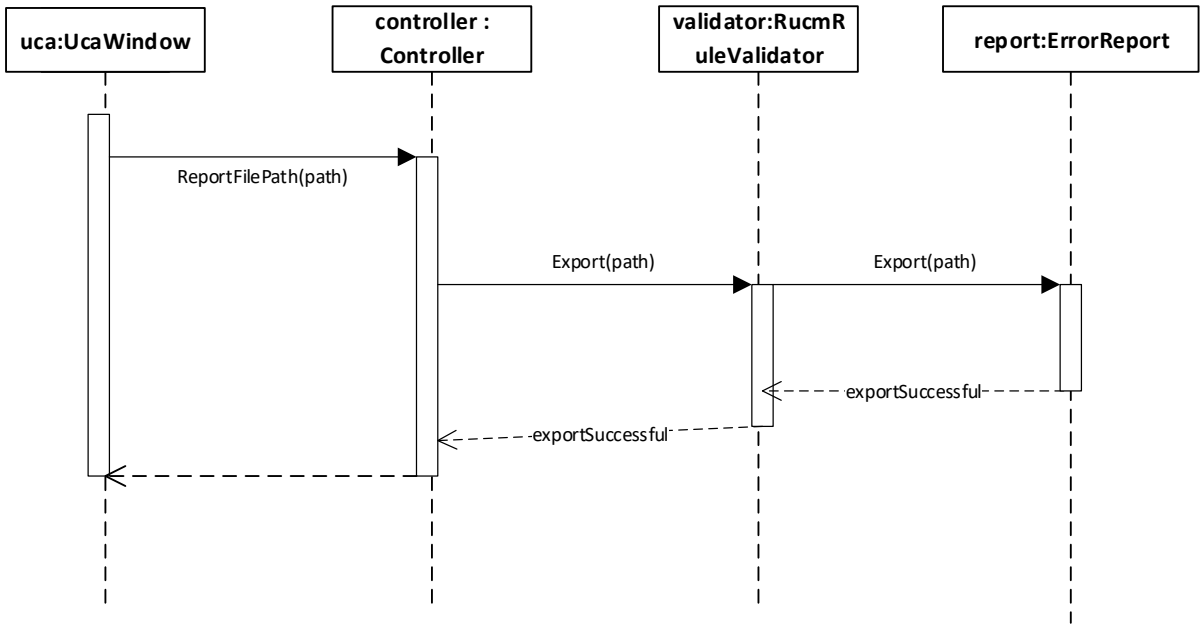


Abbildung 5: AI-SWP\02_Entwurf\RuleValidation\Sequenz_Export.vsdx

Use-Case: Szenariomatrix exportieren

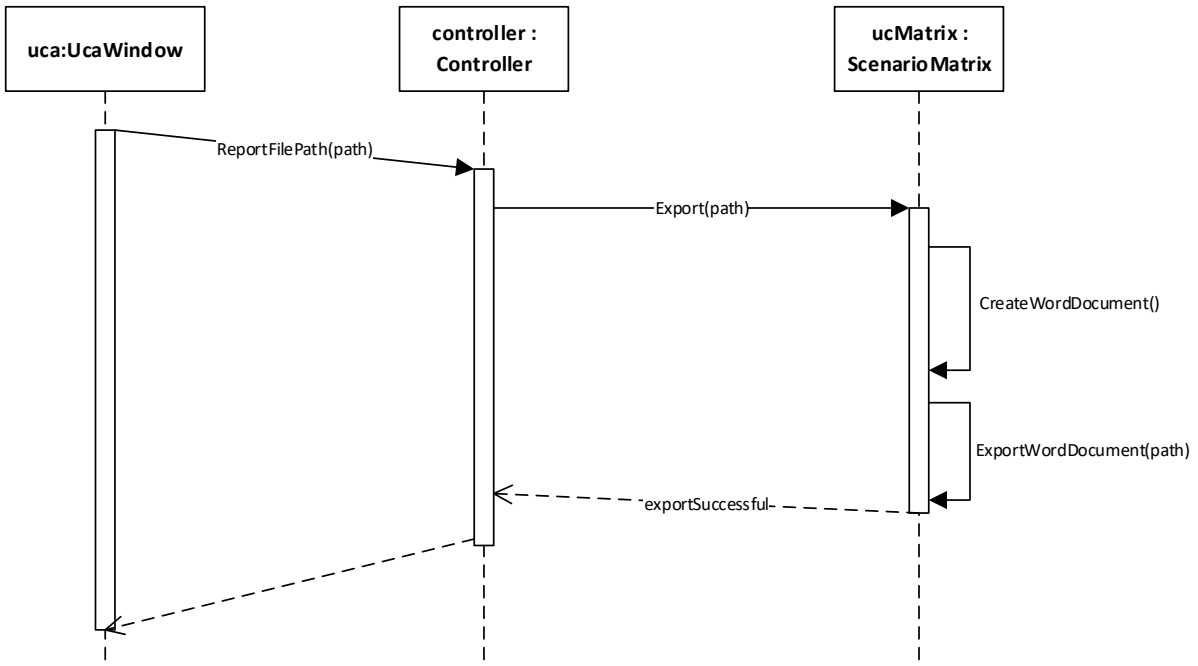


Abbildung 6: AI-SWP\02_Entwurf\SzenarioMatrix\SzenarioMatrix_Export.vsdx

Use-Case: Szenario-Matrix konfigurieren

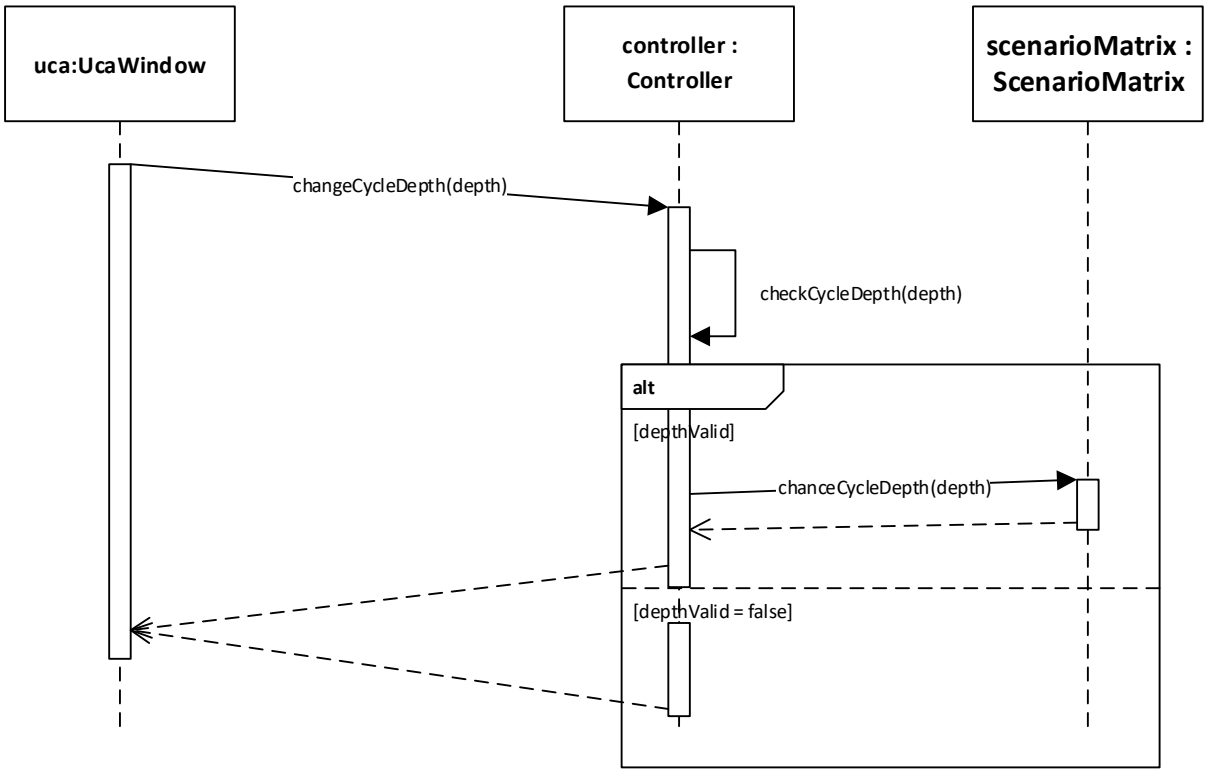


Abbildung 7: AI-SWP\02_Entwurf\SzenarioMatrix\SzenarioMatrix_Zyklustiefe.vsdx

7 Abkürzungsverzeichnis

Abkürzung	Erklärung
UC	Use Case
GUI	Graphical User Interface
MIT	Massachusetts Institute of Technology
RUCM	Restricted Use-Case-Modeling
Ms-PL	Microsoft Public License
XML	Extensible Markup Language

8 Literaturverzeichnis

9 Abbildungsverzeichnis