Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung - Anwendungen 1

Parham Vasaiely

System Modellierung, Test und Simulation
mit UML/SysML und Modelica

Faculty of Engineering and Computer Science
Department Computer Science

Fakultät Techchnik und Informatik
Studiendepartment Informatik

Parham Vasaiely

Thema des Projektvorschlags

System Modellierung, Test und Simulation mit UML/SysML und Modelica

Stichworte

UML, SysML, Modelica, Simulation, Interaktive, System, Modell basierte Entwicklung, System Entwicklung, Test, Test Automation, Anforderungen, Eclipse

Kurzzusammenfassung

Das International Council on Systems Engineering (INCOSE), erkannte die Modellbasierte System Entwicklung als eine effektive und effiziente Schlüsseltechnik für die zukünftige Entwicklungen von Systemen. Die Simulation von Systemen wird meist zu Test-, Analyse-, Kommunikations- oder Einweisungs- Zwecken verwendet. In dieser Arbeit wird im Rahmen des Open Model-Driven Whole-Product Development and Simulation Environment (OPENPROD) Projektes ein Ansatz einer Umgebung zur Modellierung, zum Test, und zur Simulation von Systemen entwickelt. Dabei stehen, die Entwicklung auf Basis einer offenen Implementierung z.B. in Eclipse, die Modellierung in einer einheitlichen und generellen Grafischen Modellierungssprache wie UML/SysML und Simulation mit Unterstützung der System Programmiersprache Modelica im Vordergrund. Ebenso das Validieren und Verifizieren von Systemen durch Testen.

Title of the project proposal

System Modelling, Testing and Simulation with UML/SysML and Modelica

Keywords

UML, SysML, Modelica, Simulation, Interactive, System, Model based Engineering, Systems Engineering, Test, Test automation, Requirement, Eclipse

Abstract

The International Council on Systems Engineering (INCOSE) identified Model-Based Systems Engineering as a key driver for effective and efficient system development in the future. System simulation using models is widely used for analysis, communication or training purposes. This thesis presents an approach for an environment including modelling, testing and simulation of systems within the scope of the Open Model-Driven Whole-Product Development and Simulation Environment (OPENPROD) Project. As requested in OPENPROD the main focuses are developing based on a open product like the Eclipse environment, modelling in an unified and general usage graphical modelling language like UML/SysML and simulation with the next generation system programming language Modelica. In addition the validation and verification of systems with testing will be focused.

Inhaltsverzeichnis

I. Abbildungsverzeichnis

II. Glossar

INCOSE	International Council on Systems Engineering
MBSE	Model-Based Systems Engineering
OM	OpenModelica
OMG	Object Management Group
OMI	OpenModelica Interactive
SysML	Systems Modelling Language
U2TP	UML 2.0 Test Profile
UML	Unified Modelling Language
CAE	Computer Aided Engineering

1. Einleitung

1.1. Hintergrund

Das International Council on Systems Engineering (INCOSE) [13] identifizierte die Modellbasierte System Entwicklung (MBSE) [4] als eine wichtige Schlüsseltechnik zur effizienten und effektiven Entwicklung von Systemen in der Zukunft. Eine standardisierte Notation zur Beschreibung der System Anforderungen oder des System Designs an jedem beliebigen Punkt der Entwicklungsphase ist eines der wichtigsten Techniken in der MBSE. Ebenfalls, hat sich die Simulation von technischen Systemen, beispielsweise zu Analyse, Validierungs- und Verifikationszwecken, in der Praxis schnell als ein großer Vorteil des MBSE herausgestellt.
Während der Entwicklung von komplexen Systemen sind mehrere Ingenieursdisziplinen involviert, welche jede ihre eigenen Formalismen und Werkzeuge zur Entwicklung ihrer Teilkomponenten einsetzen. Beispiele für komplexe technische Systeme, welche speziell in dieser Arbeit betrachtet werden, kommen aus dem Automobil sowie der Flug und Raumfahrt Industrie.

Die Object Management Group (OMG) hat sich, unter anderem, die Standardisierung von Modellierungssprachen zur Unterstützung der Entwickler als Aufgabe gestellt. Bekanntestes Beispiel aus der Software-Entwicklung ist die Unified Modelling Language (UML) [6], welche Weltweit als grafische Standard Notation bei der Entwicklung von komplexen Software-Systemen zum Einsatz kommt. Ebenfalls von der OMG ist die sogenannte Systems Modelling Language (SysML) [7]. SysML ist ebenfalls eine grafische Modellierungssprache, welches ein, speziell an die Anforderungen und Bedürfnisse der System-Entwicklung angepasstes, Profil der UML ist. Modelica [2] bietet sich als Programmiersprache für die Modellbasierte Entwicklung von Systemen an, da die Sprache eigens für diese Disziplin entwickelt wurde. Die MBSE wird ebenfalls durch die ausgezeichneten Simulationseigenschaften von Modelica unterstützt. Ergänzend hierzu ist das Testen von Systemen eine wichtige Möglichkeit, welche maßgeblich für die erfolgreiche und qualitative Entwicklung eines Systems verantwortlich ist. Für die grafische Entwicklung von Tests unter Verwendung der UML, bietet sich das UML2 Test Profil (U2TP) [21] an, welches ebenfalls von OMG entwickelt wurde.

1.2. Aufgabenstellung und Motivation

Dieses Projekt findet im Rahmen des OPENPROD Projektes statt, welches ein von der Europäischen Union als ITEA2-Projekt [22], gefördertes und von führenden Industrie Unternehmen, Forschungs- Instituten und Universitäten angefragtes Forschungsprojekt ist.

Einer der bedeutenden Paradigmenwechsel, der sich im Bereich der System bzw. Produkt-Entwicklung vollzieht, ist die durchgängige Verwendung eines Modells entlang des Produktlebenszyklus. Diese Betrachtungsweise ermöglicht einen wesentlich effizienteren Entwicklungsprozess, da die zu entwickelnden Systeme über alle Entwicklungsphasen hinweg getestet und die kontinuierlich zum Modell ergänzten Informationen durchgängig verwendet werden können. Das OPENPROD Projekt setzt an diesem Ansatz an indem es einen ganzheitlichen Ansatz verfolgt, der eine Vielzahl unterschiedlicher Aspekte der Produktentwicklung berücksichtigt und in die angestrebte integrierte Entwicklungsumgebung einfließen lässt. OPENPROD ist ein Projekte zur Schaffung einer durchgängigen, ganzheitlichen modelgetriebenen Entwicklungsumgebung vom Geschäftsprozess über die Anforderungserfassung zu Plattform unabhängigen Modellen (PIM) und schließlich Plattform spezifischen Modellen (PSM).

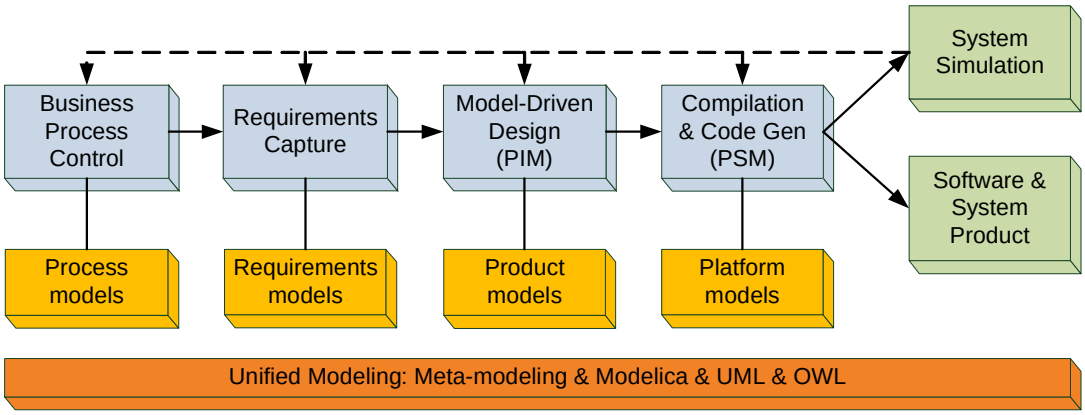

Abbildung 1-1: Idee des OPENPROD

Ein genereller Trend in der Systementwicklung sind die zunehmende Komplexität technischer Systeme sowie die verteilten Standorte, in denen diese entwickelt und hergestellt werden. Die Komplexität technischer Systeme ergibt sich aus dem erhöhten Grad der Automation, der Vielzahl der beteiligten Technologien sowie der Tatsache, dass Software und Hardware einer engen Kopplung unterliegen. Komponentenhersteller müssen daher zunehmend in den Entwicklungsprozess eingebunden werden.

Komponentenhersteller sollen dabei in der Lage sein, die Funktion der zu liefernden Komponente im Systemkontext bereits zur Entwurfszeit verifizieren zu können, lange bevor das Endprodukt hergestellt wird.

Die IT-Landschaft im Bereich der System-Entwicklung ist durch kommerzielle Computer Aided Engineering (CAE) - Werkzeuge geprägt. Die Strategie der Anbieter zielt auf die Etablierung ihrer Produkte entlang des gesamten Produktlebenszyklus ab. Dieses bietet dem Anwender einen hohen Komfort hinsichtlich der Durchgängigkeit und des Datenaustausches zwischen den Werkzeugen, bringt ihn jedoch auch in eine Abhängigkeit, da die zugrunde liegenden Managementsysteme auf Proprietären Datenformaten basieren. Somit stehen die Anwender von Engineering Tools unter anderem vor folgenden Herausforderungen:

- Interoperabilität und Datenaustausch verlangen zusätzliche Investitionen in eine teure IT-Infrastruktur.
- Die Durchgängigkeit der Toolkette ist nur innerhalb der Produktlandschaft des Toolherstellers gewährleistet und führt bei Informationsaustausch zwischen Tools von verschiedenen Herstellern zwangsläufig zu Informationsverlust.
- Offene Standards, sofern vorhanden, werden nur unzureichend unterstützt, um nicht in Konkurrenz zum eigenen Datenformat zu treten.

Im Bereich der Softwareentwicklung haben sich mit der UML als grafische Modellierungssprache und den vielfältigen Programmiersprachen offene Standards bezüglich Spezifikation, Entwurf und Implementierung etabliert. Modelica und SysML zeigen großes Potential, diese Lücke auch im Bereich der Systementwicklung zu schließen. Das Durchsetzungsvermögen dieser Standards hängt in hohem Maße von der Verfügbarkeit und Benutzbarkeit entsprechender Modellierungs- und Simulationsumgebungen ab. Dieses Vorhaben wird durch die Bereitstellung von frei Verfügbaren OpenSource Tools, deren Modellbeschreibung auf offenen Standards basiert unterstützt. Eine frei verfügbare, integrierte Entwicklungsumgebung ermöglicht es dem Systemhersteller und den Komponentenherstellern ihre Kommunikation und Interaktion, basierend auf einem ganzheitlichen Systemmodell und ohne erwähnte Beschränkungen von Tool Herstellern zu führen.

1.3. Ziele dieser Arbeit

Da das OPENPROD Projekt eine Vielzahl verschiedener Aufgaben der ganzheitlichen Systementwicklung behandelt, fokussiert diese Arbeit nur die folgenden Probleme und Aufgabenstellungen, welche bei der Entwicklung technischer Systeme entstehen:

- Standardisierte (Grafische) Modellierung der Systeme und deren Teilkomponenten, unter Verwendung von UML/SysML und Modelica.
- Analyse, Verifikation und Validierung des Systems und der Teilkomponenten durch Simulation der Selben.
- Verifikation, Validierung und Unterstützung bzw. Verbesserung der Qualitätssicherung durch Testen des Systems und der Teilkomponenten.

Die erarbeiteten Grundlagen gilt es durch die technische Umsetzung in Form von Prototypischen SW- Werkzeugen zugänglich zu machen. Die technischen Ziele lassen sich wie folgt zusammenfassen:

- Erarbeitung und Erweiterung offener standardisierter Beschreibungssprachen und Zwischenformate.
- Integration der Methoden und Verfahren in frei verfügbare Engineering- und Simulationswerkzeuge.
- Nachweis der Einsetzbarkeit der Methoden und Werkzeuge anhand von Demonstratoren.

Folgende Wissenschaftliche Erkenntnisse werden erwartet:

- Ansatz für eine durchgängige Methodik basierend auf einer standardisierten Notation (UML/SysML, Modelica) und integrierten, erweiterbaren Werkzeugen (Eclipse) zur Modellierung und Simulation heterogener Systeme.
- Methoden zur formalen Beschreibung von Anforderungen und deren Überführung in mathematisch auswertbare Formen.
- Entwicklung und Implementierung von Methoden zur semi-automatischen Qualitätssicherung für Modelle.

2. Stand der Wissenschaft und Technik

2.1. Modellierungssprachen

Ein Modell ist ein vereinfachtes und auf die Wesentlichen, für die entsprechende Entwicklungsphase, abstrahiertes Abbild eines realen Systems oder einer Teilkomponente. Modelle werden zur Veranschaulichung, Kommunikation, Interaktion und Analyse verwendet. Eine Modellierungssprache hilft dem Entwickler die Anforderungen an ein System und dessen Struktur in einer abstrakteren Ebene darzustellen.

2.1.1. Modelica

Modelica [5] ist eine frei verfügbare, objektorientierte Sprache für die Modellierung von großen, komplexen und heterogenen physikalischen Systemen. Modelica wird von der Modelica Association [15] spezifiziert und veröffentlicht. Sie ist geeignet für Multi-Domain-Modellierung wie z.B. mechatronische Modelle in der Robotik, Automobil und Aerospace-Anwendungen mit deren mechanischen, elektrischen und hydraulischen Systemen sowie deren Steuerungssystemen, aber auch für prozessorientierte Anwendungen und die Anwendung im Bereich Erzeugung und Verteilung elektrischer Energie. Modelle in Modelica werden durch Differentialgleichungen und diskrete Gleichungen mathematisch beschrieben. Keine Variable muss von Hand gelöst werden, da ein Modelica- Werkzeug genügend Informationen besitzt, dies automatisch zu entscheiden. Modelica ist so konzipiert, dass verfügbare, spezielle Algorithmen genutzt werden können, um die effiziente Bearbeitung von großen Modellen mit mehreren hunderttausend Gleichungen zu gewährleisten. Es eignet sich für Hardware-in-the-Loop-Simulationen und Embedded-Systeme und wird in diesen Bereichen eingesetzt.

2.1.2. UML

Die UML hat sich als umfassende, standardisierte Notation für die Entwicklung von Softwaresystemen etabliert. Die UML wird von der OMG spezifiziert und stellt einen weltweiten Standard dar. Im Bereich des System Engineering existiert derzeit kein solcher weltweiter Standard, darum ist vor allem bei interdisziplinären Projekten mit einem Informationsverlust zu rechnen, wenn Informationen in Form von Modellen ausgetauscht werden. INCOSE hat es sich 2001 zum Ziel gesetzt, die UML zum Standard für das System Engineering zu machen. Die UML bietet durch ihren Erweiterungsmechanismus die Möglichkeit, die UML an verschiedene Disziplinen und Domänen anzupassen.

2.1.3. SysML

Im Bereich des System- Engineering hat sich derzeit kein entsprechender Standard weltweit durchgesetzt, darum wurde mit der SysML (Version 1.0 2006, aktuelle Version 1.1 von Dez. 2008) ein UML- Profil von der OMG und dem INCOSE standardisiert, das den Anspruch hat, System Engineering durchgängig und domänenübergreifend zu unterstützen. Die SysML erweitert die UML um Anforderungsdiagramme und unterstützt so die Nachverfolgbarkeit von Anforderungen durch den Entwicklungsprozess. Über Blockdiagramme und - als Spezialisierung - Parametrische Diagramme werden Komponenten und ihre Schnittstellen in SysML beschrieben und in ein Systemmodell integriert, die nicht mit der UML entworfen werden. SysML lässt jedoch an vielen Stellen einen Interpretations- und Auslegungsspielraum zu, um Domain-spezifische Anpassungen zu erleichtern. Heterogene Systemmodelle, beschrieben in der SysML, ermöglichen noch keine Simulation des Systemverhaltens, denn neben der bewusst informellen Semantik und der softwarezentrierten Sicht auf das Systemverhalten fehlen Attribute, um unterschiedliche Ausführungs- und Kommunikationsarten (Models of Computation and Communication, MoCC) präzise zu spezifizieren. Die meisten Grundkonzepte in SysML basieren auf UML und spiegeln somit stark die Softwareentwicklungssicht wieder. In der jetzigen Version ist SysML nur bedingt geeignet für die physikalische System Modellierung (z.B. Modellierung und Simulation von hybriden Systemen). Die Anwendung von SysML in der Systementwicklung ist somit beschränkt, was das Nichtvorhandensein von allgemein etablierten Vorgehensweisen für modellbasierte Systementwicklung in der Industrie erklärt [10].

2.2. Simulation

„Simulation ist die Nachbildung eines dynamischen Prozesses in einem „realen“ Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind“ (Definition nach DIN 3633). Der Erkenntnisgewinn wird durch Experimente gewonnen, welche auf ein Mathematisches Modell durchgeführt werden und maßgeblich zur Validierung und Verifikation der dynamischen Teile eines entwickelten Modells beitragen.

2.2.1. Notwendigkeit und Gefahren der Simulation

Es gibt eine Vielzahl von Gründen ein Experiment durch eine Simulation auf einem Modell durchzuführen, anstatt Experimente auf realen Systemen. Hier einige davon:

- Reale Experimente benötigen ein physikalisches Modell beispielsweise ein Prototyp. Diese sind meist sehr teuer und werden durch die Experimente evtl. selbst beschädigt oder könnten sogar Personen beschädigen.
- Die Zeit welche für ein reales Experiment benötigt wird, könnte viele Jahre betragen und die Menschliche Zeit somit weit übersteigen. Eine Simulation hingegen kann ebenfalls die Zeit simulieren und die Ergebnisse in kürzester Zeit darstellen.
- Nicht alle Eigenschaften eines realen Systems lassen sich messen, da sie beispielsweise nicht zugänglich sind, in einer Simulation hingegen sind die Werte verfügbar.

Allerdings birgt eine Simulation auch Gefahren in sich, welche meist vernachlässigt werden und im schlimmsten Fall zu einer Katastrophe führen könnten:

- Wie erwähnt ist ein Modell nur eine Abstraktion der Realität. Diese sind somit keineswegs äquivalent zu einander. Da die Realität viele Faktoren beinhaltet welche nicht auf ein Modell abgebildet werden können.
- Ebenfalls können nicht alle Eigenschaften der Realität in Bedingungen eines Modells eingebunden werden. Da diese meist selbst einen komplexen Hintergrund haben. Somit ist dies erneut eine Abstraktion der Realität.
- Das Maß der Fehlerfreiheit eines Modells muss beachtet werden und daraus resultierend die Interpretation der Ergebnisse eines Simulation.

2.2.2. Arten von Simulationen

In diesem Kontext gibt es Grundsätzlich zwei verschiedene Arten von Simulationen. Diese Unterscheiden sich einerseits im Zeitverhalten und andrerseits in der Möglichkeit mit einem Benutzer oder anderen System zu Interagieren. welche hier kurz erwähnt und beschrieben werden sollen:

- Nicht Interaktive Simulation: Die nicht interaktive Simulation zeichnet sich durch ihre Eigenschaft aus ein dynamisches Modell über ein fixes Zeitintervall (Start, Stopp) zu Simulieren. Die Simulationszeit ist somit asynchron zur realen Weltzeit, was zum Vorteil hat zeitintensive Simulationen in sehr kurzer Zeit durchzuführen. Es bietet keinerlei Interaktion mit einem Benutzer. Gewünschte individuelle Eingaben müssen durch Angabe einer festen Zeit durchgeführt werden. Diese Art der Simulation eignet sich zur schnellen Erkenntnisgewinnung. Ebenfalls eignet es sich auch um dynamische Tests durchzuführen, dazu mehr in Kapitel 2.3.

- Interaktive Simulation: Die Simulationszeit ist synchron zur realen Weltzeit Die interaktive Simulation bindet ihren Takt an die reale Zeit und läuft somit parallel zu dieser ab. Dies ermöglicht eine Interaktion mit einem Benutzer oder einem anderen System. Erst hierdurch wird eine „Live Demonstration“ des Systems, beispielsweise zum Verständnis der Funktionalität für alle Beteiligten, ermöglicht.

2.3. Testen

Das Testen ist eines der wichtigsten Methoden zur strukturellen Analyse und anschließenden Auswertung der Ergebnisse eines Modells hinsichtlich seiner statischen und dynamischen Struktur. Ein Systemtest ist ein Test, welcher während der Systementwicklung durchgeführt wird. Die Notwendigkeit von Tests ist durch folgende Begründungen gegeben:

- Fehlerwirkungen sollten noch möglichst vor ihrer Wirkung in der Realenwelt erkannt und ihre Fehlerzustände beseitigt werden.
- Das Testen von Komponenten und deren Integrationstest als Gesamtsystem ist eines der wichtigen Möglichkeiten zur Qualitätssicherung.
- Ebenfalls kann die Verifikation und Validierung des Systems und der Komponenten mit Hilfe von Tests durchgeführt werden.

2.3.1. Textuelle Anforderungen

Anforderungen, geschrieben in natürlicher Sprache, stellen weiterhin eine große Herausforderung in der Systementwicklung dar. Natürliche Sprache ist nicht eindeutig, sie ist nur begrenzt Computer-Verarbeitbar und verursacht somit intensive manuelle Pflege von textuellen Anforderungen. Dies führt oft zu Inkonsistenzen und Missinterpretationen von Anforderungen, was den Einsatz im Test Kontext erschwert.
Abhilfe schafft hier die „Formale Anforderungsmodellierung“ durch vorhandene Verhaltensdiagramme der UML, welche keine Interpretation der Spezifizierten Eigenschaften mehr zulassen, dazu mehr in Kapitel 2.3.3.

2.3.2. Testarten

Da diese Arbeit, unter anderem, das Thema der System Simulation behandelt, sind zwei wichtige Testarten zu beachten, der funktionale und nicht funktionale Test. Der funktionale Test prüft ein System hinsichtlich spezifizierter Leistungsmerkmale. Wo hingegeben der Nicht-funktionale Test anhand von Software- und Systemmerkmalen, prüft wie das System arbeitet (Bsp.: Performanztest, Lasttest).

Ebenfalls geprägt durch die Simulation wird auf Statische Tests verzichtet, welche das Modell und den erzeugten Code nicht zur Ausführung bringen, sondern deren richtige Semantik und Interpretation prüfen. Anders als beim dynamischen Test, welches das Modell mit Hilfe von Testkomponenten innerhalb einer Simulation zur Ausführung bringt.
Beim dynamischen Test sind grundsätzlich zwei Sichten bzw. Informationsgrundlagen des Modells zu beachten. Beim Black-Box Verfahren verfügt man beim Test über keinerlei Informationen des Inneren Aufbaus, beobachtet wird hier nur das Verhalten des Systems von außen. Wo hingegen beim White-Box Verfahren, die innere Struktur bekannt ist und somit mit in die Tests einbezogen werden kann.

2.3.3. Modellbasiertes Testen

Modellbasiertes Testen ist ein Verfahren, um Testfälle automatisch zu erzeugen.
Es ermöglicht ein systematisches und effektives Testen bei relativ geringen Kosten.
Mit Hilfe formaler Techniken lassen sich Anforderungen präzise formulieren, so dass eine Überprüfung auf Konsistenz und Vollständigkeit mit Hilfe mathematischer Modelle und Werkzeuge möglich ist. Die formale Modellierung erlaubt beispielsweise Testfälle werkzeuggestützt direkt aus den Anforderungen abzuleiten oder Beweise zu führen, welche die Konsistenz von Anforderungen nachweisen. Der Aufwand für die formale Anforderungsmodellierung durch die vorhandenen Verhaltensdiagramme wird durch die Möglichkeit der vollständigen und frühen Verifikation kompensiert. In diesem Zusammenhang muss evtl. auch der Ansatz von Visuellen Kontrakten beachtet werden. Visuelle Kontrakte basieren auf der Theorie von Graph-Transformationen und beschreiben das Verhalten anhand formaler Vor- und Nachbedingungen, die durch UML grafisch spezifiziert werden.

2.3.4. UML 2 Test Profil

Das UML2.0- Test Profil (U2TP) basiert auf UML2.0 und erweitert die UML um die Fähigkeit ein Testsystem zu Entwerfen, Visualisieren, Spezifizieren, Analysieren, Konstruieren und zu Dokumentieren. U2TP erweitert UML2.0 beispielsweise um testspezifische Konzepten wie Testkomponenten, Testergebnisse, Testfälle, Test Suite, Default- Verhalten. Da U2TP als UML Profil definiert ist, ist es nahtlos in UML integriert. Da es auf dem UML2.0- Metamodell basiert und die UML2.0-Syntax verwendet kann es, mit Einschränkungen, auch in Verbindung mit anderen UML2.0- Profilen verwendet werden.

3. Weitere Schritte

3.1. Eclipse als Basis Technologie

Ein Ziel des OPENPROD ist es offene und damit wieder verwendbare und nicht kommerzielle Lösungen zu Entwickeln. Hierfür bietet sich die Entwicklung von Werkzeugen und Profilen unter Verwendung der Eclipsetechnologie [24] an. Nachfolgend sind alle Identifizierten Eclipse Erweiterungen als sogenannte „Eclipse Plug-ins“ aufgeführt und werden kurz erläutert.

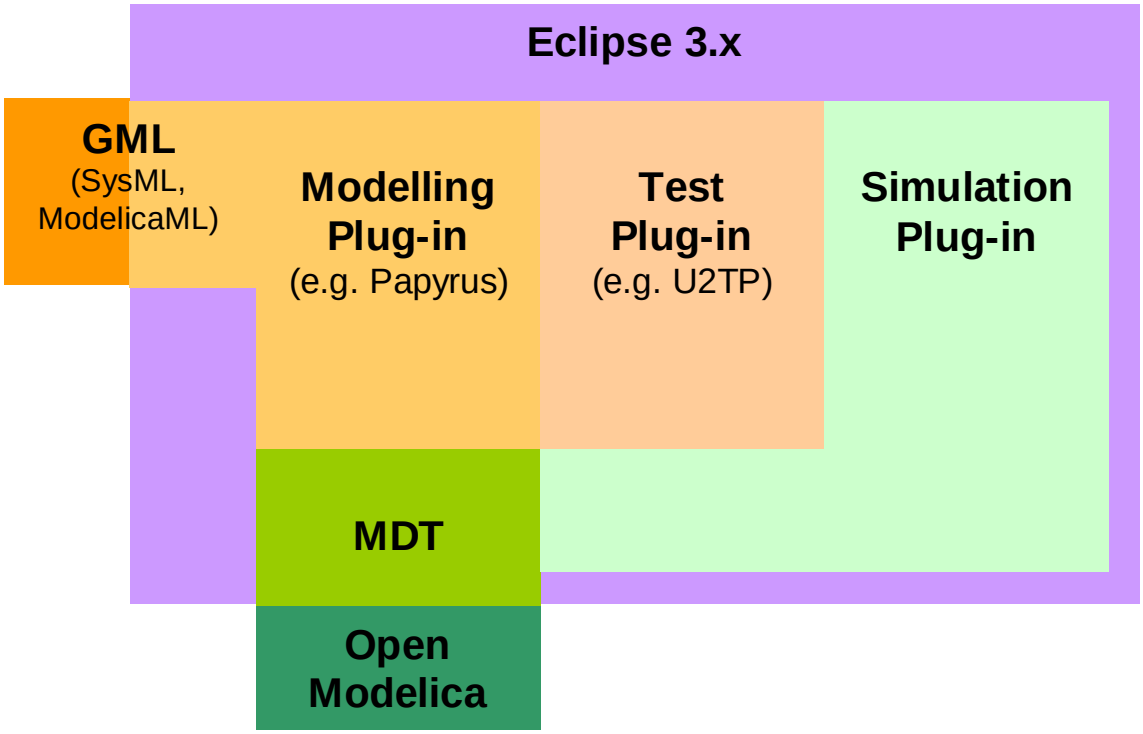

Abbildung 3-1: Identifizierte, von dem Projekt verwendete Informations- Komponenten und Komponenten basierend auf Eclipse Plug-ins.

GML: Eine auf UML2 basierende grafische Modellierungssprache, welche verwendet wird um mit Hilfe einer Modellierungsumgebung ein System zu modellieren. Diese Sprache kann SysML, ModelicaML (siehe Kapitel 3.1.1) oder eine Andere sein. Die reine Spezifikation einer solchen Sprache ist nicht Teil von Eclipse und ist somit außerhalb der Eclipse Umgebung zu realisieren.

Modelling Plug-in: Dieses Plug-in verwendet die GML und bietet eine grafische Modellierungsumgebung, in der der Benutzer die System Architektur spezifizieren und das

Design grafisch erstellen kann. Als Plug-in würde sich eine bereits vorhandene und Quell offene UML Modellierungsumgebung namens Papyrus [25] anbieten.

Simulation Plug-in: Um ein entwickeltes System zu simulieren und die Resultate zu analysieren wird ein Simulationsumgebung benötigt. Diese sollte die Nicht Interaktive wie die Interaktive Simulation unterstützen. Ebenfalls sollte die Umgebung den Benutzer bei der Darstellung der Ergebnisse unterstützen und eine Interaktionsschnittstelle zwischen Simulationsobjekt und dem Benutzer anbieten.

Test Plug-in: Dieses Plug-in verwendet die, unter der Modellierungsumgebung erstellten, formalen Anforderungsmodelle und das erstellte UML Modell um Testfälle zu generieren. Es unterstützt den Benutzer bei der Durchführung der Tests und bietet somit Teile einer Testautomatisierung an. Hierfür sollte das UML2TP verwendet werden, welches ein effizienteres Testen ermöglicht.

OpenModelica: OpenModelica (OM) ist ein Quell offenes Modelica Tool, entwickelt von der Linköping University [19] und des Open Source Modelica Consortium (OSMC) [18]. Die OpenModelica Umgebung setzt sich aus vielen Teilen zusammen. Hauptbestandteil ist ein Modelica Compiler welcher mit Unterstützung von Gleichungslösern eine ausführbare Applikation zur Simulation des Modells anbietet [8].

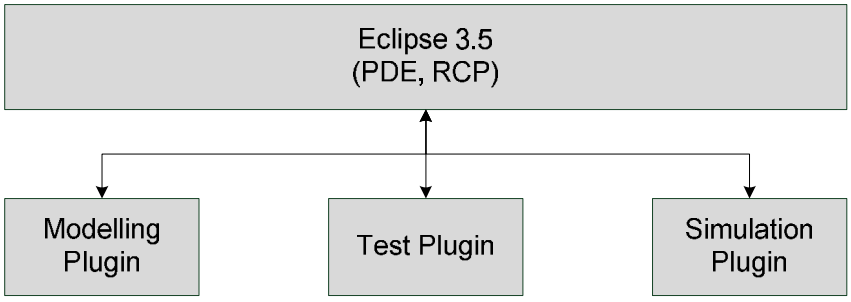

Abbildung 3-2: Übersicht der zu implementierenden Eclipse Plug-ins

3.1.1. Die Modellierungssprache und Modellierungsumgebung

Die grafische Repräsentation von Systemen ist durch die steigende Komplexität derselben nicht weg zu denken.

Eine standardisierte und offene System- Modellierungssprachen muss folgende Grundeigenschaften besitzen:

- Sie muss Unternehmensübergreifend einsetzbar sein, so dass Modelle ausgetauscht werden können ohne diese noch mal umwandeln zu müssen, was in den meisten Fällen zu Informationsverlust führt.
- Ein offener Standard, welcher die unkomplizierte und unkommerzielle Verwendung ermöglich.
- Nicht Anwendungsfall spezifische Merkmale besitzen, um den Einsatz während der gesamten Entwicklungsstadien und eine weite Verbreitung zu gewährleisten.
- Sie muss Domänen übergreifend einsetzbar sein, welches ebenfalls die Verwendung im gesamten Entwicklungszyklus unterstützt und die Kommunikation fördert.

Modelica ist als textuelle Syntax definiert und bietet nur eine beschränkte graphische Modellierungsunterstützung verglichen mit UML/SysML. Beide Modellierungssprachen bieten spezifische Vorteile für die Multidomänen-Modellierung und Simulation. Mit ModelicaML [23] wurde daher bereits ein UML- Profil zur Integration der beiden Notationen vorgeschlagen. Ansätze sollten daher speziell bei ModelicaML gesucht werden.

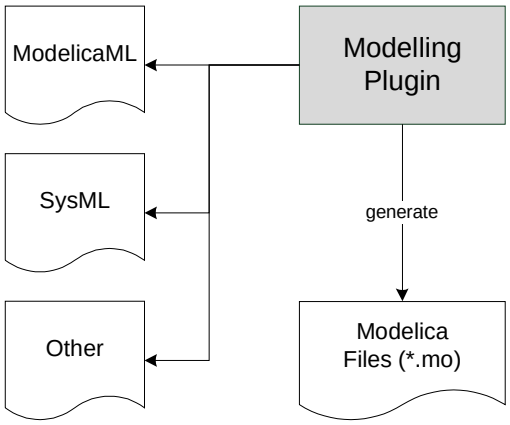

Abbildung 3-3: Identifizierte Komponenten der Modellierungsumgebung

Die Entwicklung einer Modellierungsumgebung ist nicht Hauptbestandteil dieser Arbeit. Allerdings muss eine Prototypische Entwicklungsumgebung für die Ableitung von Tests und der Simulation vorhanden sein. Hierfür bietet sich wie erwähnt Papyrus an.

3.1.2. Die Simulationsumgebung

Die Simulationsumgebung muss dem Benutzer die Nicht Interaktive sowie die Interaktive Simulation eines in UML2 erstellten Systems ermöglichen. Die Schnittstelle zu OM wird zur Erzeugung einer Simulationslaufzeit verwendet, welche als lauffähige Applikation das

System und Gleichungslöser (Solver) zur Simulation beinhaltet. Ebenfalls sollte eine Interaktionschnittstelle zwischen Simulation und Benutzer, zur Kontrolle einer Interaktiven Simulation in Form von Start, Stopp und Pause Befehlen sowieso die visuelle Änderung von System Parametern, ermöglicht werden.

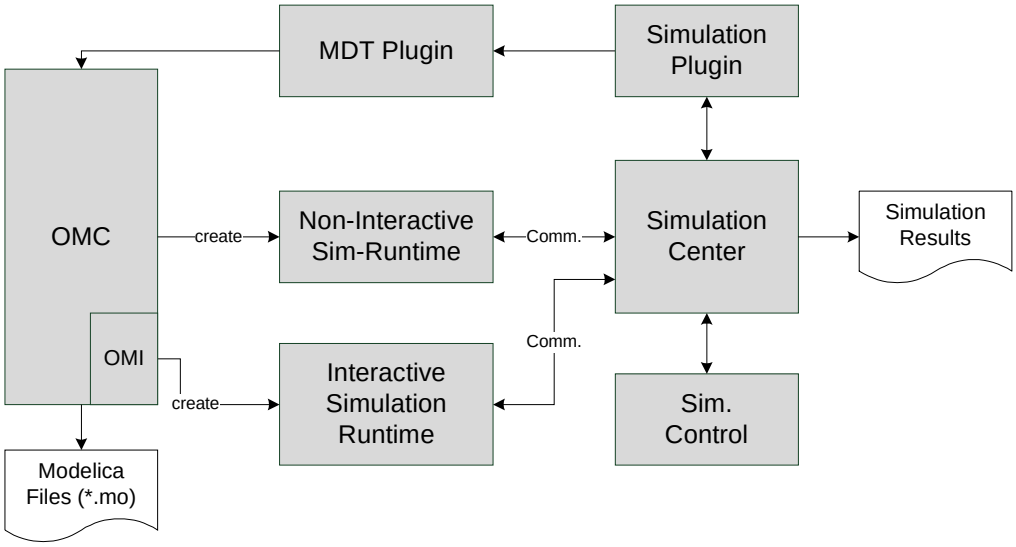

Abbildung 3-4: Identifizierte Komponenten der Simulationsumgebung

Resultate sollten dem Benutzer grafisch dargestellt werden. Hierfür sollten vordefinierte Views in Form von Graphendiagrammen verwendet werden.

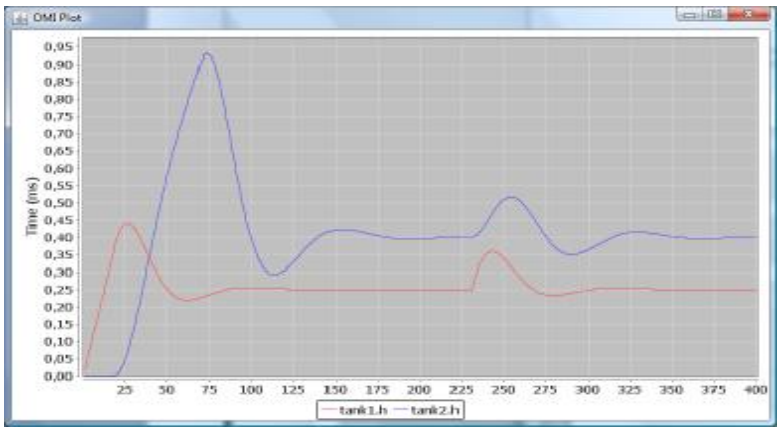

Abbildung 3-5: Mögliche Grafische Darstellung von Werten

Um eine erweiterte und Benutzerdefinierte Visualisierung zu ermöglichen sollte ebenfalls eine Schnittstelle zur Anbindung eigener Views angeboten werden.

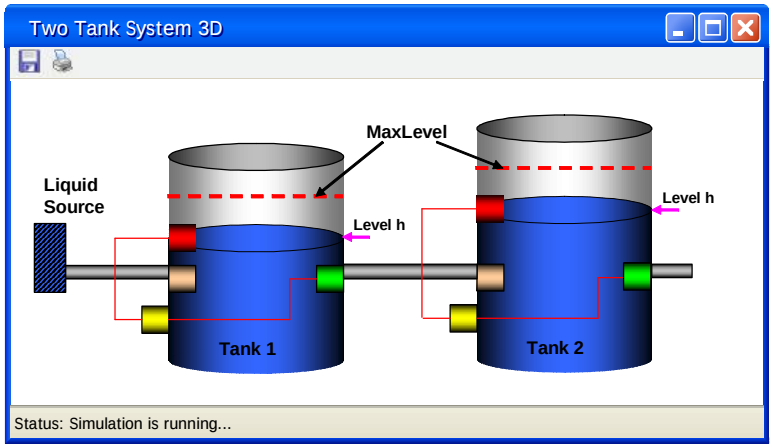

Abbildung 3-6: Beispiel für eine Benutzerdefinierte Darstellung von Ergebnissen

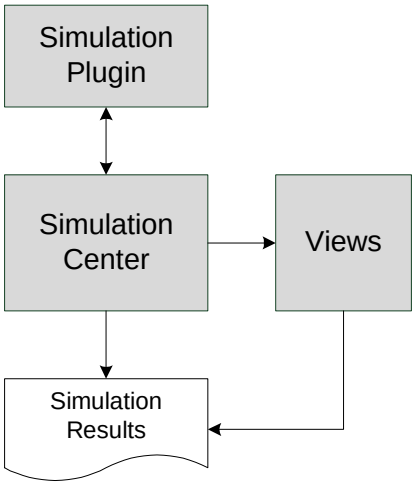

Abbildung 3-7: Identifizierte Komponenten der Grafischen Darstellung von Ergebnissen

Das Originale OM unterstützt keine Interaktive Simulation. Ein Ansatz für eine Interaktive Simulation von Modelica Modellen unter Verwendung von OpenModelica würde bereits in der Bachelorarbeit von Parham Vasaiely „Interactive Simulation of SysML Modells using Modelica“ [11] erforscht. Die Arbeitet bieten ebenfalls ausgearbeitete Grundlagen für den praktische Einsatz dieser Technik.

3.1.3. Die Testumgebung

Die Entwicklung eines Testverfahrens mit UML/Modelica und die damit verbundene Entwicklung einer Testumgebung zur Realisierung und Automatisierung von Tests, ist neben der Simulation des Systems ein Hauptforschungsbestandteil dieser Arbeit. Behandelt wird die Funktionale Testweise, welche anhand der funktionalen Anforderungen spezifiziert wird. Die Verwendung von spezifikationsbasierten Testentwurfsverfahren ermöglicht, Testendekriterien und Testfälle aus der Funktionalität des Systems herzuleiten

und die Vollständigkeit der Prüfung (Überdeckungsgrad) zu bewerten. Die Überdeckungskriterien bieten dabei eine heuristische herangehensweise an das Auswahlproblem von Testfällen. Da die Testmenge, beliebig groß sein kann, muss eine gewisse Auswahl getroffen werden. An eine Auswahl werden Anforderungen gestellt, die die Testfälle erfüllen müssen. Dabei unterscheidet man zwischen den Überdeckungskriterien für Modelle und den Anforderungen. Der Benutzer sollte dabei Unterstützt werden diese zu definieren. Die Tests müssen dann mit Hilfe der Simulation ausgeführt werden. Wobei die Analyse und Auswertung der Ergebnisse nicht Bestandteil der Arbeit sein sollte.

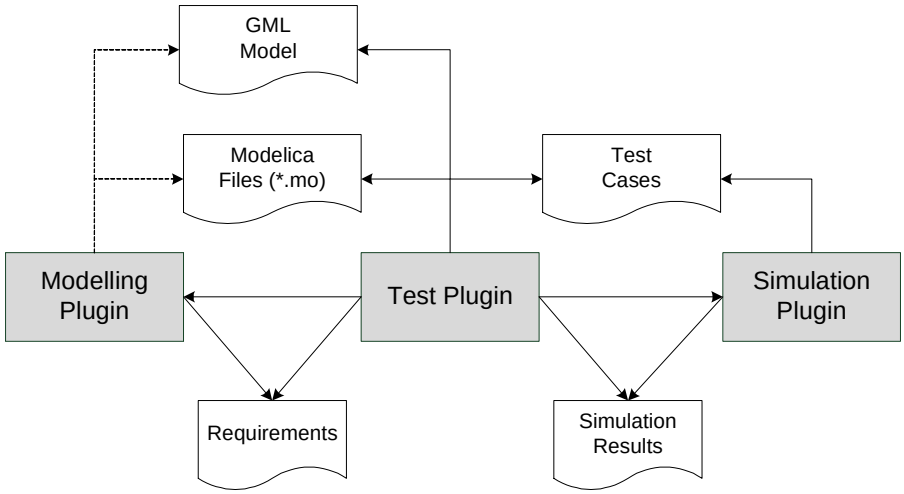

Abbildung 3-8: Identifizierte Komponenten einer Testumgebung

Die Test Automation sollte daher folgende Schritte behandeln:

1. Erzeugung Funktionaler Tests
2. Test Überdeckung
3. Test Ausführung

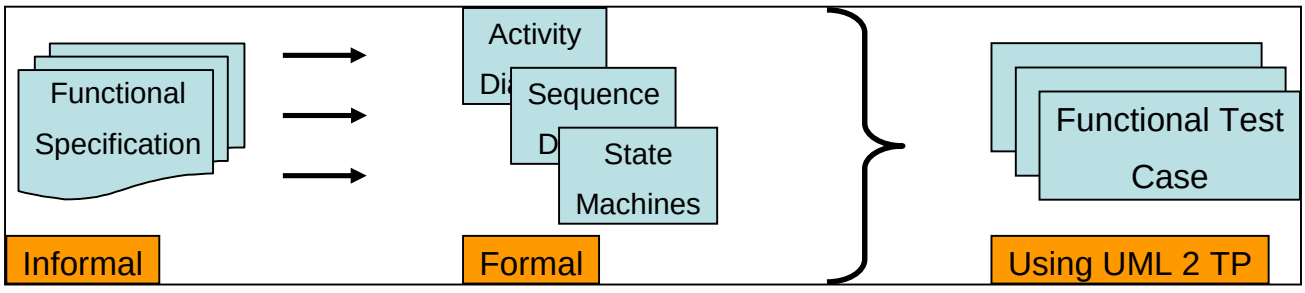

Abbildung 3-9: Erster Schritt zur Erzeugung von Testfällen bei der Test Automation ist die formale Anforderungsspezifikation

III. Referenzen

[1] Sanford Friedenthal, Alan Moore and Rick Steiner, 2008, Practical Guide to SysML: The Systems Modeling Language, Morgan Kaufmann.

[2] Fritzson Peter, 2004, Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, Wiley-IEEE Press.

[3] Ralf Reussner und Wilhelm Hasselbring, 2006, Handbuch der Software-Architektur, dpunkt Verlag.

[4] Friedenthal, Sanford, Greigo, Regina, and Mark Sampson, INCOSE MBSE Roadmap, in “INCOSE Model Based Systems Engineering (MBSE) Workshop Outbrief” (Presentation Slides), presented at INCOSE International Workshop 2008, Albuquerque, NM, pg. 6, Jan. 26, 2008.

[5] Modelica Association, 2005, "Modelica Language Specification Version 3.0", http://www.modelica.org/documents/ModelicaSpec30.pdf, September 5, 2007.

[6] Object Management Group UML, “OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2”, http://www.omg.org/docs/formal/07-11-02.pdf, November 2007.

[7] Object Management Group SysML, “OMG Systems Modeling Language (OMG SysML™) Specification”, http://www.omg.org/docs/formal/08-11-02.pdf, November 2008.

[8] PELAB, Peter Fritzson, “OpenModelica System Documentation, Version, 2008-01-27 for OpenModelica1.4.5”, http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/releases/1.4.5/doc/OpenModelicaSystem.pdf, January 2009.

[9] PELAB, Peter Fritzson, “OpenModelica Users Guide, Version 2009-01-27 for OpenModelica 1.4.5”, http://www.ida.liu.se/labs/pelab/modelica/OpenModelica/releases/1.4.5/doc/OpenModelicaUsersGuide.pdf, January 2009.

[10] Jeff A Estefan, “Survey of Model Based Systems Engineering Methodologies”. http://omgsysml.org/MBSE_Methodology_Survey_RevB.pdf, May 2008

[11] EADS Innovation Works, HAW Hamburg, Parham Vasaiely, Bachelor Thesis: “Interactive Simulation of SysML Models using Modelica.pdf”, August 2009.

[12] Computing and Mathematics Research Division Lawrence Livermore National Laboratory, Petzold, Linda R., http://www.netlib.org/ode/ddassl.f, December 12 2006.

[13] The International Council on Systems Engineering (INCOSE), Last Accessed: 2009 http://www.incose.org/

[14] Object Management Group (OMG) Systems Modelling Language, Last Accessed: 2009 http://www.omgsysml.org/

[15] Modelica and the Modelica Association, Last Accessed: 2009 http://www.modelica.org/

[16] Modelica and the Modelica Association, Modelica Libraries, Last Accessed: 2009 http://www.modelica.org/libraries

[17] The OpenModelica Project, Last Accessed: 2009 http://www.ida.liu.se/~pelab/modelica/OpenModelica.html

[18] Open Source Modelica Consortium, Last Accessed: 2009 http://www.ida.liu.se/labs/pelab/modelica/OpenSourceModelicaConsortium.html

[19] Linköping University, Last Accessed: 2009 http://www.liu.se

[20] OpenModelica source code version 1.4.5 from Subversion repository, http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html#Download

[21] Object Management Group UML 2 Test Profile, “UML 2 TP Specification”, http://www.omg.org/cgi-bin/doc?formal/05-07-07, July 2007.

[22] Information Technologie for European Advancement ITEA2, Last Accessed: 2009 http://www.itea2.org/

[23] Wladimir Schamai, Modelica Modeling Language (ModelicaML), Last Accessed: 2009 http://www.ida.liu.se/~pelab/modelica/OpenModelica/MDT/ModelicaML/

[24] Eclipse Platform and Eclipsetechnologie, Last Accessed: 2009 www.eclipse.org

[25] Papyrus for UML, Last Accessed: 2009 http://www.papyrusuml.org/