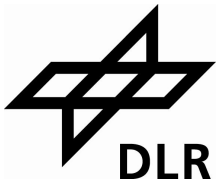

Bidirektionale Anbindung eines grafischen Editors an ein Systemmodell

BACHELORARBEIT

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der Dualen Hochschule Baden-Württemberg Mannheim

von

Juliane Müller

16.09.2013

Bearbeitungszeitraum:	12 Wochen
Matrikelnummer, Kurs:	5538716, TIT10AIN
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt, Braunschweig
Betreuer der Ausbildungsfirma:	Dipl.-Ing. V. Schaus
Gutachter der Dualen Hochschule:	Prof. Dr. R. Colgen

Ehrenwörtliche Erklärung

gemäß § 5 (2) der “Studien- und Prüfungsordnung DHBW Technik” vom 18. Mai 2009

Hiermit erkläre ich, Juliane Müller, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

_____

Braunschweig, den 13.09.2013

Abstract

The software Virtual Satellite (VirSat), created by the German Aerospace Centre, supports aerospace engineers by creating space projects. The software provides various features like creating individual components, defining parameters and integrating new calculations. This is done with the help of tabular editors, which reflect no correlations. For a more sophisticated visualization, a graphical editor has to be integrated into the VirSat, which represents the structure of the information. Changes of the data model are tracked and own changes can be written into the data model with this editor. For this purpose, the data model of VirSat has to be filtered according to the significant values. To integrate the graphical editor the Graphiti framework has been used. Using the layout framework KIELER the diagrams in the editor are clearly structured and the single elements are arranged in a sophisticated way. In summary, a graphical editor with a functioning two-way coupling of the editor with the data model and a well-arranged layout has been integrated into the VirSat. The graphical editor can be used as an additional way to manipulate the data.

Kurzfassung

Die Software Virtueller Satellit (VirSat), welche vom Deutschen Zentrum für Luft und Raumfahrt entwickelt wird, unterstützt Raumfahrtingenieure bei der Erstellung eines Raumfahrtprojektes. Dabei können Ingenieure Komponenten erstellen und den Komponenten Parameter und Berechnungen hinzufügen. Dies geschieht mit Hilfe von tabellarischen Editoren, welche keine Zusammenhänge wiederspiegeln. Zur besseren Veranschaulichung soll ein grafischer Editor in den VirSat integriert werden, welcher die Informationsstruktur darstellt. Über diesen Editor können Änderungen im Datenmodell des VirSats nachvollzogen und neue Änderungen ins Datenmodell geschrieben werden. Hierzu muss es nach den relevanten Werten gefiltert, welche im grafischen Editor dargestellt werden. Der grafische Editor wird mit Hilfe des Graphiti-Frameworks erstellt. Die Diagramme im Editor sollen klar strukturiert sein. Dafür soll das Layoutframework KIELER in die Software integriert werden. Zusammenfassend wurde ein grafischer Editor mit funktionierender bidirektionaler Kopplung des Editors mit dem Datenmodell und übersichtlichem Layout in den VirSat integriert. Der grafische Editor kann als zusätzliche Manipuliermöglichkeit von Daten genutzt werden.

Danksagung

Zunächst möchte ich mich an dieser Stelle bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelor-Arbeit unterstützt und motiviert haben. Ganz besonders gilt dieser Dank Herrn Dipl.-Ing. V. Schaus und Prof. Dr. R. Colgen, die meine Arbeit und somit auch mich betreut haben. Sie haben mich dazu gebracht, über meine Grenzen hinaus zu denken. Vielen Dank für die Geduld und Mühen. Daneben gilt mein Dank Michael Tiede, der in zahlreichen Stunden Korrektur gelesen hat. Er wies mich auf Schwächen hin und konnten als Fachinterner gute Ratschläge geben. Auch meine Vorgesetzten und Kollegen bei dem Deutschen Zentrum für Luft und Raumfahrt haben maßgeblich dazu beigetragen, dass diese Bachelorarbeit nun so vorliegt. Vielen Dank, dass Sie mir die Möglichkeit gegeben haben, bei Ihnen zu forschen und zu arbeiten. Nicht zuletzt gebührt meinen Eltern und meinem Bruder Dank, da Sie während des Studiums nicht nur finanziell, sondern vor allem auch emotional immer für mich da waren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AOCS	**A**ttitude and **O**rbit **C**ontrol **S**ystem
BDD	**B**lock**d**efinitions**d**iagramm
CEF	**C**oncurrent **E**ngineering **F**acility
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt
EMF	**E**clipse **M**odeling **F**ramework
FBD	**F**unktions**b**lock-**D**iagramm
GEF	**G**raphical **E**diting **F**ramework
Graphiti	**Graphi**cal **T**ooling **I**nfrastructure
IBD	**I**nternes **B**lock**d**iagramm
ICD	**I**nterface **C**ontrol **D**ocument
KIELER	**K**iel **I**ntegrated **E**nvironment for **L**ayout **E**clipse **R**ichClient
SC	**S**ystem**C**omponent
SVN	Apache **S**ub**v**ersio**n**
SysML	**Sys**tems **M**odeling **L**anguage
UML2.0	**U**nified **M**odeling **L**anguage **2.0**
VirSat	**Vir**tueller **Sat**ellit

1 Einleitung

> *Design ist nicht nur wie es aussieht und sich anfühlt. Design ist wie es funktioniert.*
>
> Steve Jobs, The Guts of a New Machine (NY Times, 1993)

Stellen Sie sich eine Applikation vor, welche die beste Funktionalität die man sich wünschen kann bietet, jedoch unübersichtlich und nicht intuitiv ist. Oder das Gegenteil, ein Tool mit einem perfekt durchgeplanten Aussehen, jedoch ohne jegliche Funktionalität. Würden Sie diese gern nutzen?
Der Schlüssel zum Erfolg liegt in einem makellosem Zusammenspiel beider. Dies erkannte auch Steve Jobs als er im Jahr 1976 die Firma Apple Inc. gründete. Er versuchte das äußere Design, wie auch den Aufbau des Betriebssystems, so intuitiv wie möglich zu gestalten.

1.1 Ausgangssituation und Motivation

Den gleichen Grundgedanken vom perfekten Zusammenspiel zwischen Funktionalität und Design versuchen die Entwickler der Software **Vir**tueller **Sat**ellit (VirSat) nachzugehen. Diese wird zur Planung von Raumfahrtstudien in einer Großraumforschungsanlage genutzt und ist das Grundgerüst dieser Arbeit. Es dient hierbei hauptsächlich als Datenzusammentragungsprogramm. Die Entwickler versuchen bei ihrer Implementierung nicht nur einwandfreie Features in die Software zu integrieren, sondern legen besonderen Wert auf die Nutzbarkeit des Programms. Dafür stehen sie im ständigen Kontakt mit den späteren Nutzern der Software - den Raumfahrtingenieuren, welche ihre Wünsche und Gedanken zu der Software mitteilen.
Die Software ist so aufgebaut, dass mehrere Ingenieure zeitgleich ihre erarbeiteten Informationen in verschiedene tabellarische Editoren eintragen können. Diese bieten ein einfaches Eintragen der Daten, jedoch aufgrund der Vielzahl der Editoren auch eine gewisse Unübersichtlichkeit. Weiterhin sind keine Zusammenhänge zwischen den Informationen in diesen Tabellen erkennbar. Diese muss der Nutzer stets selbst suchen. Auch bei einer späteren Abänderung der Informationen ist das Wiederfinden dieser insbesondere bei komplexen Komponenten mit Aufwand verbunden.
Weiterhin sind die Raumfahrtingenieure nach dem Zusammentragen der Daten für die

Erstellung von **I**nterface **C**ontrol **D**ocuments zuständig. Diese beinhalten die Schnittstellendefinitionen der behandelten Komponente. Zur besseren Veranschaulichung werden hierbei **B**lock**d**efinitions**d**iagramme genutzt, welche die Ein- und Ausgänge der Komponente darstellt.
Zum Entlasten der Raumfahrtingenieure bei dem Eintragen und Wiederfinden von Informationen und dem Erstellen von BDD soll ein grafischer Editor in die Software VirSat integriert werden, welcher die Schnittstelleninformationen und die interne Struktur einer Komponente darstellt.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist das Erstellen und Integrieren eines grafischen Editors, welcher die erarbeiteten Informationen in verständlicher grafischer Form darstellt, in die Software **Vir**tueller **Sat**ellit. Die Informationen sollen in zwei verschiedenen Arten dargestellt werden: einmal in Anlehnung an das **B**lock**d**efinitions**d**iagramm und an das **I**nterne **B**lock**d**iagramm, wobei ersteres die Schnittstelleninformationen und zweiteres die Schnittstelleninformationen inklusive dem inneren Aufbau beinhaltet. Weiterhin soll der grafische Editor auf Benutzerinteraktionen reagieren. Hierbei müssen zunächst die Benutzerinteraktionen, wie Hinzufügen oder Editieren eines Elements definiert werden. Die Änderungen sollen nicht nur im grafischen Editor selbst angezeigt, sondern auch in den anderen tabellarischen Editoren angenommen werden. Bei den Interaktionen soll darauf geachtet werden, dass nicht jede Person alles verändern kann. Es muss überprüft werden, ob die Person auch die Zugriffsrechte für die jeweilige Eingabe besitzt.
Weiterhin sollen die Diagramme einen guten Aufbau besitzen. Dabei ist besonders auf das Layout, die Formatierung, die Farbgebung und die Informationsfilterung zu achten. Eine weitere Anforderung dieser Arbeit besteht in der Darstellung ingenieurstechnischer Besonderheiten bei Diagrammen. Dabei sollen die verschiedenen gängigen Darstellungsformen in die Diagramme integriert werden. Zusätzlich soll eine Möglichkeit entwickelt werden die Sonderformen beliebig zu erweitern.

1.3 Gliederung der Arbeit

Diese Arbeit befasst sich zunächst mit der Planung von Raumfahrtmissionen mit Hilfe der Software **Vir**tueller **Sat**ellit. Der VirSat ist das Grundgerüst dieser Arbeit, da der grafische Editor in diese Software integriert werden soll. in diesem Kapitel werden die Benutzung, der interne Aufbau und die Implementierungskonzepte des VirSats beschrieben. Diese sind zum Grundverständnis der weiterführenden Arbeit notwendig. Das Kapitel 3 beschreibt alle benötigten Konzepte und Frameworks, welche in dieser Arbeit angewendet wurden. Zusätzlich wird auch die Vorarbeit zu dieser in dem Kapitel dargestellt. Danach folgt das Kapitel des Entwurfs und Implementierung. Es befasst sich zunächst mit dem Entwurf der Diagramme und später mit dessen Umsetzung. Zum Schluss folgt das Fazit und der Ausblick der Arbeit, in dem die erledigte Arbeit zusammengefasst und kritisch betrachtet wird.

2 Planung von Raumfahrtmissionen mit Hilfe der Software Virtueller Satellit

Von dem ersten Gedanken an eine neue Raumfahrtmission bis hin zum Start vergehen circa fünf bis acht Jahre. Dieser Zeitraum wird in sechs Phasen unterteilt. Die erste Phase (Phase A) beschäftigt sich mit der Umsetzbarkeit des Projekts sowie der Auswahl eines geeigneten und kosteneffizienten Konzepts. Sie dauert meist über einen Zeitraum von acht bis zwölf Monaten an. In Phase B werden detaillierte Informationen zu den einzelnen Komponenten ausgearbeitet und zusammengetragen. Die Phasen C und D sind die längsten Phasen bei der Erstellung eines Satelliten. In ihnen werden die bisherigen Angaben zu den Bestandteilen angepasst sowie die Hardwarekomponenten anhand der Spezifikationen angefertigt und zusammengesetzt. In Phase E wird der Satellit in seinen Orbit befördert und tätigt dort seine Aufgabe und Phase F impliziert die Entsorgung des Satelliten, welche entweder als kontrollierter Wiedereintritt und Verglühen in der Atmsphäre oder als Eintreten in einen Parkorbit erfolgt. [Mül13, FSS11]
Ein Satellit wird immer für einen bestimmten Zweck, wie beispielsweise das Sammeln von Luftaufnahmen, konzipiert. Um einen Satelliten planen zu können, ist das Wissen von Ingenieuren unterschiedlicher Fachrichtungen notwendig (siehe Abbildung 1). Ein Ingenieur ist hierbei zum Beispiel für die Energiegewinnung im Satelliten, welche meist durch Solarpanele konzipiert wird, zuständig. Ein anderer regelt die Stromversorgung durch eine Batterie , welche von den Solarpanelen mit Energie versorgt wird. Weiterhin sollen Daten gesammelt werden, wie beispielsweise die Luftaufnahmen eines Planeten. Dafür ist der Kamera- und Missionsspezialist zuständig. Ein weiterer Ingenieur kümmert sich um die Kommunikation des Satelliten mit der Erde, so dass die gesammelten Luftaufnahmen übertragen werden und neue Befehle erhalten werden können. Ein solcher Befehl ist beispielsweise die Änderung der Flugbahn.
Die einzelnen Erkenntnisse der Planungsmitglieder sind von großer Bedeutung. Dimensioniert der Spezialist für Batterie und Massenspeicher die Anforderungen falsch oder bietet die Batterie nicht ausreichend Speicher um eine große Datenmenge in kurzer Zeit abzuspeichern, so kann das Missionsziel gefährdet sein. Grund hierfür sind die multidisziplinären Abhängigkeiten, welche die Komponenten untereinander verbinden.
Damit die Ergebnisse aller Ingenieure besser zusammengetragen und auch im Zusammenspiel simuliert werden können, arbeiten sie nach dem Concurrent Enginee-

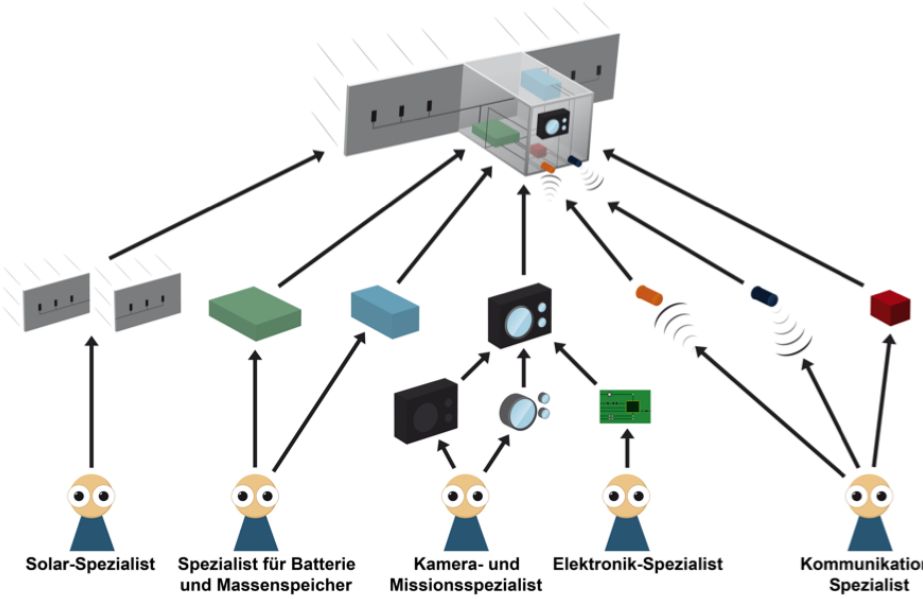

Abbildung 1: Zuständigkeit der Ingenieure für die einzelnen Teilbereiche eines Raumfahrtprojekts [Tie13]

ring Prozess. Das Concurrent Engineering ist eine effiziente Arbeitsweise, da eine direkte verbale und mediale Kommunikation zwischen den Ingenieuren bestehen und sie durch Diskussionen schnelles Feedback und Problemerkennung bekommen. Hierbei führen die Ingenieure innerhalb von ein bis zwei Wochen eine Machbarkeitsstudie unter Leitung eines Projektmanagers durch. Insbesondere in Phase A kommt dieses Arbeitsprinzip zum Einsatz. Für diesen Arbeitsschritt wurde eine Großforschungsanlage, **C**oncurrent **E**ngineering **F**acility (CEF) genannt, im **D**eutsches Zentrum für **L**uft- und **R**aumfahrt (DLR)[1] am Standort Bremen errichtet. Auf Abbildung 2 sieht man ei-

Abbildung 2: Zusammenarbeit mehrerer Raumfahrtingenieure in der CEF in Bremen

[1] Das DLR ist eine der zentralen Einrichtungen für Forschung in den Bereichen der Luftfahrt, Raumfahrt, Energie, Verkehr und Sicherheit in Deutschland.

ne solche Arbeitsweise in der CEF. Dabei sitzen die unterschiedlichen Ingenieure im Halbkreis um den Projektleiter. Zur Präsentation der erarbeiteten Daten werden diese auf den Hauptmonitor abgespiegelt. Auf diesen hat jeder der Spezialisten eine gute Sicht und ideale Grundvoraussetzungen für eine erfolgreiche Diskussion sind erstellt. Ein weiterer Vorteil der Großraumforschungsanlage ist das Arbeiten aller Mitglieder mit einem Datensatz. Die Software, welche in der CEF zur Informationszusammentragung durch mehrere Raumfahrtingenieure eingesetzt wird, trägt den Namen VirSat.
Im Unterkapitel 2.1 wird die Software des Virtuellen Satelliten in seiner Nutzbarkeit vorgestellt. Darüber hinaus wird in diesem Kapitel die Notwendigkeit eines grafischen Editors aufgezeigt. Das Unterkapitel 2.2 beschreibt den internen Aufbau der Software inklusive der zugrundeliegenden Konzepte der Entwicklung.

2.1 Die Software Virtueller Satellit im Überblick

Der VirSat ist eine von der Einrichtung Simulations- und Softwaretechnik des DLR erstellte Software, welche hauptsächlich zur Unterstützung der CEF in Bremen entwickelt wird. Sie ist speziell an die Ansprüche und Wünsche der Raumfahrtingenieure für die Planung einer Raumfahrtmission angepasst und konzipiert worden. Eine Perspektive des Programms ist in Abbildung 3 zu sehen. Der Repository Manager (im unteren Teil der Abbildung), ist für das Laden des Repositories[2] zuständig. In ihm werden alle bereits erarbeiteten Ergebnisse abgespeichert. Dieses Repository kann immer wieder geschlossen und neu geöffnet werden und besitzt dabei stets den gleichen Inhalt.
Die *Study Navigator*-Ansicht beschreibt die hierarschische Dekomposition des Raumfahrtzeuges mit all seinen Einstellungen. In der Abbildung 3 ist die Raumfahrtstudie mit einem Satelliten namens AHAB zu sehen, welche 2011 in der CEF durchgeführt wurde. In der Ansicht sieht man im Study Navigator Part, dass der AHAB-Satellit aus Komponenten besteht, die aus weiteren Subkomponenten bestehen können. Eine Subkomponente in dieser Ansicht ist die Lageregelung AOCS. Diese besteht aus weiteren Subkomponenten. Weiterhin sieht man den Antrieb (Propulsion), die Sensoren und die Komponente der Struktur der Lageregelung. Hierbei sind die unterschiedlichen Ingenieure auch für die einzelnen Subkomponenten zuständig. Weiterhin werden die Berechtigungen der Ingenieure in der Studie unter dem Rollenmanagement

[2] "*Die grundlegenden Funktionen eines Repository bestehen in der Speicherung, Versionskontrolle und Unterstützung beim Abrufen der gespeicherten Modelle und deren Bestandteilen.*" [PDLS13]

(Role Management) verwaltet, bei dem jedem Ingenieur sein Verantwortungsbereich zugeteilt wird. Dies ist notwendig, da ansonsten jeder Ingenieur alles bearbeiten kann, was zu Fehlern und Irritationen führt. Korrekt soll jeder Experte nur die ihm zugewiesene Komponente bearbeiten können. Die unterschiedlichen, für die Studie erforderlichen, Einheiten sind im Einheitenmanagement (Unit Management) festgehalten. Die

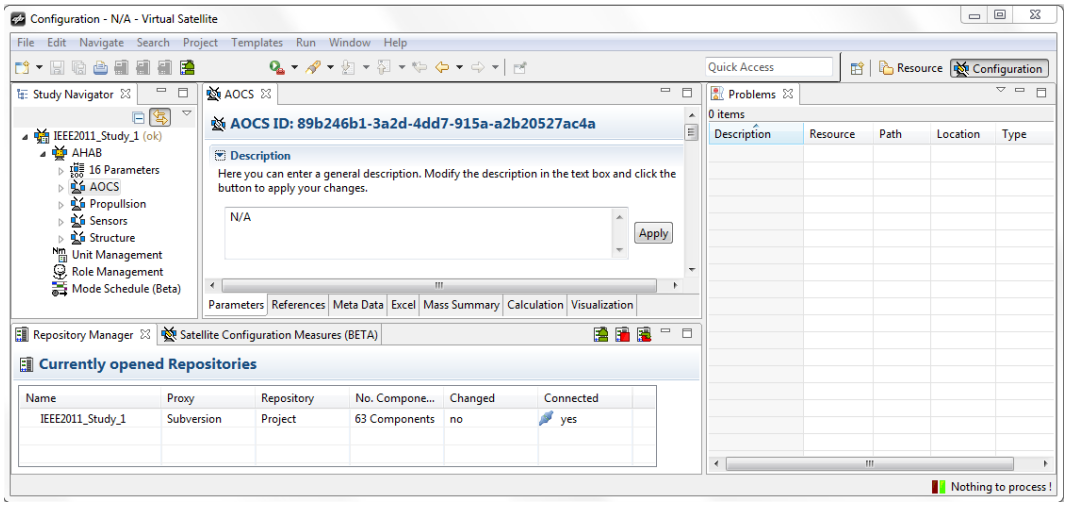

Abbildung 3: Grafische Oberfläche des Virtuellen Satelliten

Ingenieure können ihre erarbeiteten Informationen in verschiedenen Editoren abspeichern. Im Editor Part der Abbildung oder in der gesamten Abbildung 4 ist das Editorfenster zu sehen. In ihm können die Parameter und Berechnungen zu der ausgewählten Komponente erstellt, editiert oder gelöscht werden. Der Editor besteht, wie man in Tabulatoren der Abbildung 3 sehen kann aus den für diese Arbeit relevanten Tabulatoren *Parameters*, *Excel* und *Calculation*. Es ist möglich in einer Komponente mehrere Werte zu berechnen. Dafür gibt es drei verschieden Möglichkeiten Berechnungen zu erstellen Der Parameters-Tab ist für das Editieren von den Parametern und den dort erstellten Berechnungen zuständig. Diese Berechnungen sind einfache Funktionen wie eine Zuweisung oder Addition. Bis zum konsequenten Einsatz des Virtuellen Satelliten in der CEF waren es die Ingenieure gewohnt, die Raumfahrtstudien mit Hilfe der Software *Microsoft Excel* zu erstellen. Um ihnen den Übergang zwischen den beiden Tools zu erleichtern wurde der Excel-Tab in den VirSat integriert. Darin können komplexe Berechnungen mit den vorhandenen Parametern erstellt werden. Der Calculation-Tab bietet die Möglichkeit Berechnungen mit den internen Parametern in mathematischer Form[3] anzufertigen.

[3]Unter einer mathematischen Form versteht man die Darstellung von Gleichungen mit mathematischen Zeichen. Ein Beispiel wäre: $massOfSatellite = massOfComponent1 + massOfComponent2$

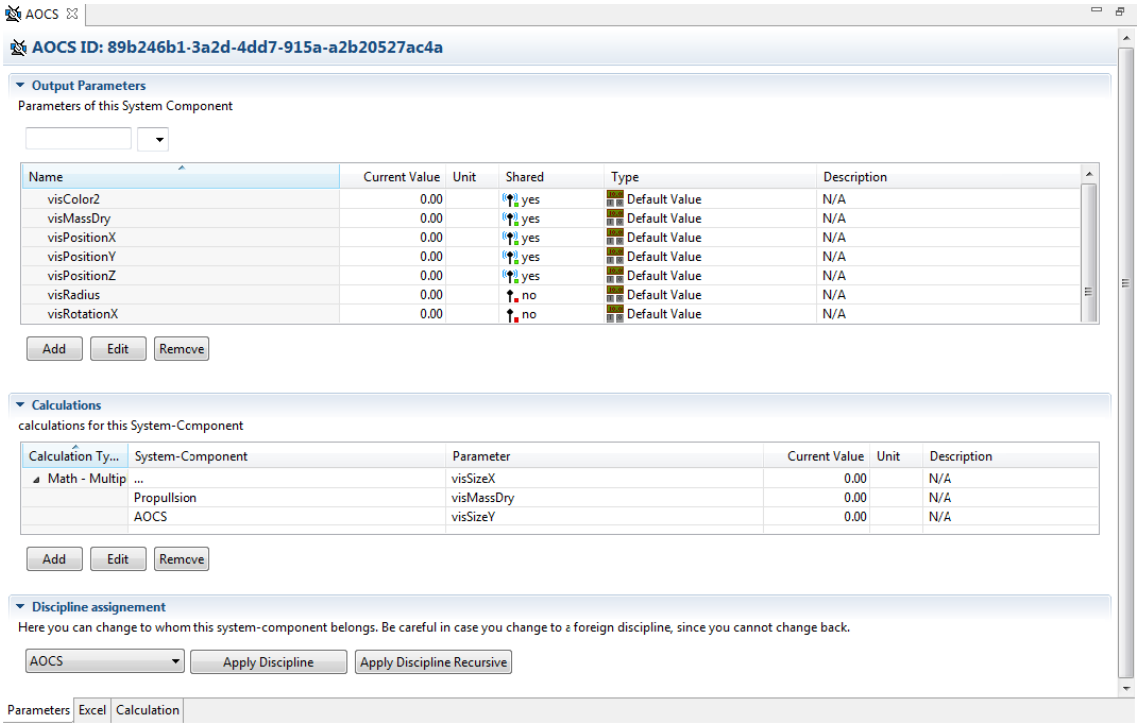

Abbildung 4: Darstellung des Komponenteneditors im Virtuellen Satelliten

Die drei verschiedenen Berechnungsarten und die Verkettung von Berechnungen einer Art können schnell zu Unübersichtlichkeit und Verwirrung führen. Soll ein Raumfahrtingenieur beispielsweise die Berechnungen einer anderen Komponente oder nach einem längeren Zeitraum seine früheren Gedankengänge nachvollziehen, so verliert er schnell den Überblick. Bisher gibt es keine Ansicht, die alle internen Berechnungen, Berechnungsarten und die Verwendung der einzelnen Parameter aufzeigt. Eine Übersicht all dieser Elemente würde den Ingenieuren ein besseres und schnelleres Verständnis der zusammengetragenen Daten bringen.
Die meisten Berechnungen werden derzeit mit dem im Parameters-Tab vorhandenen Calculations-Part erstellt. Diese Ansicht besitzt jedoch den Nachteil, dass es doppelte Scrollrichtungen und eine Art Aufzuklappen gibt (siehe Abbildung 5). In einem Gespräch mit einigen Raumfahrtingenieuren der CEF wurde festgestellt, dass diese Ansicht nur ungern genutzt wird. Besser wäre in diesem Fall ein grafischer Editor, welcher alle Berechnungen in intuitiver Form aufzeigt, editieren oder auch neue Berechnungen hinzufügen lässt.
Das Löschen eines Parameters im VirSat ist nur dann möglich, wenn er nicht anderweitig für eine Berechnung verwendet wird oder berechnet wurde. Bisher müssen die Ingenieure, im Falle der Nutzung des Parameters, alle Berechnungen nach dessen

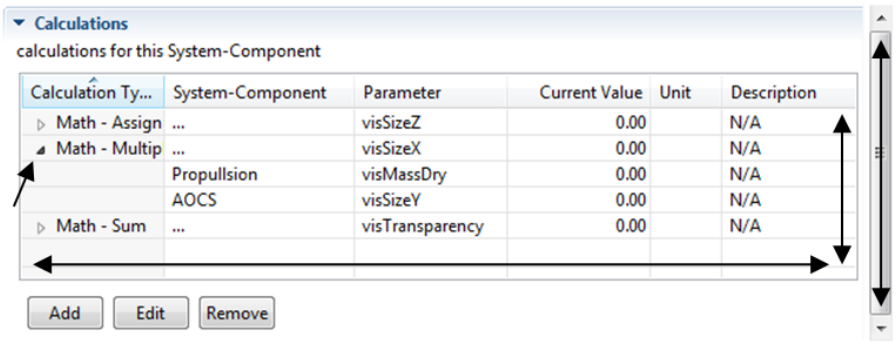

Abbildung 5: Scroll-Darstellung im Berechnungseditor

Verwendung prüfen. Es gibt keine Möglichkeit diese sofort zu erkennen. Betrachtet man Abbildung 6, so wird deutlich, dass die Parameter der component3 und component9 nicht gelöscht werden können, da sie in den Berechnungen zwei und drei Verwendung finden. Eine grafische Übersicht, wie die in Abbildung 6, würde die Arbeit der Raumfahrtingenieure beträchtlich vereinfachen.

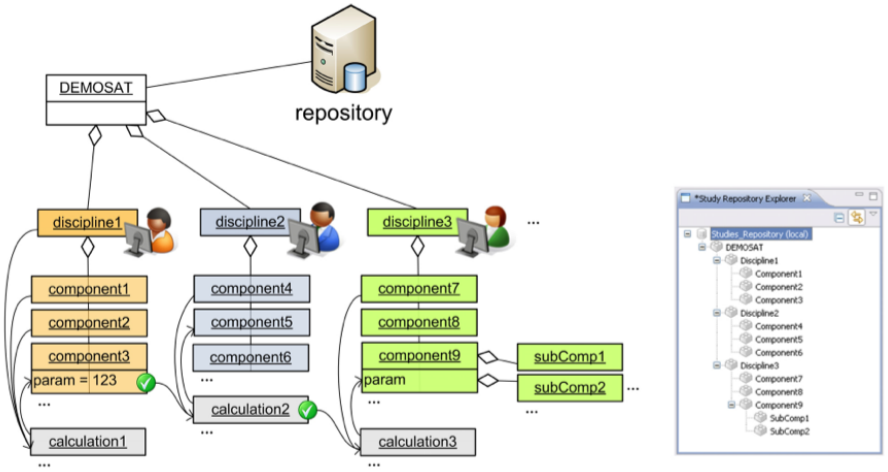

Abbildung 6: Schematische Darstellung einer Studie im Virtuellen Satelliten [SFL+10]

Jeder Ingenieur kann seine Daten mittels des Versionsverwaltungstools SVN in einem gemeinsamen Repository ablegen. Die dort abgelegten Informationen können von den anderen Personen wieder verwendet werden. So arbeiten alle Ingenieure bei stetiger Aktualisierung mit den gleichen Datensatz.
Zusätzlich zu den bisher genannten Möglichkeiten den VirSat zu nutzen, bietet er das Feature der dreidimensionalen Visualisierung der Raumfahrtzeuge im VirSat direkt oder das Exportieren eines CATScriptes zur Erstellung der Raumfahrtmission in

Computer Aided Three-Dimensional Interactive Application (CATIA) V5[4]. Dabei werden wiederum die Informationen des Datenmodells ausgelesen und die jeweiligen Elemente oder das CATScript anhand der Informationen erstellt. Die Visualisierungen durchgeführten AHAB-Studie im VirSat und in CATIA ist in Abbildung 7 zu sehen. Ein weiteres grafisches Hilfsmittel im VirSat ist die Simulation der Flugbahn des

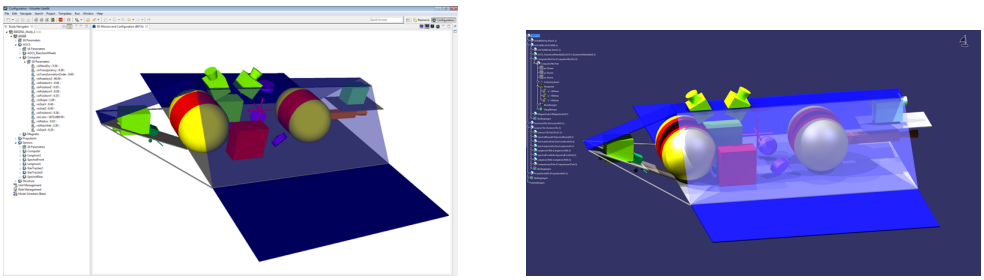

Abbildung 7: AHAB-Mission visualisiert im Virtuellen Satelliten und in CATIA

Raumfahrtprojektes. Hierbei kann man sich entweder im fortlaufenden Modus oder zu einer bestimmten Zeit den Ort der Raumfahrtmission definieren und/oder anzeigen lassen.

2.2 Implementierungskonzepte im Virtuellen Satelliten

Der Virtuelle Satellit ist eine auf der Open-Source-Entwicklungsumgebung *Eclipse* basierende Applikation. IBM begann das Eclipse-Framework im Jahr 2001 zu entwickeln und seit 2004 ist das Framework eine non-profit-Gesellschaft. Hinter dieser Umgebung steht eine sehr große Community, welche einerseits ständig versucht Eclipse intuitiver und besser zu gestalten und andererseits mit ihrem Wissen die Fragen der Anwender zu beantworten. Weiterhin beschäftigt sich die Community damit immer neue Features für die Entwicklungsumgebung zu erstellen. [Fou13] Die objektorientierte Programmiersprache Java wird im VirSat verwendet.
Entwickelt wird die Software nach dem DLR-Entwicklungsprozess. Dieser ist stark an das Scrum-Prinzip angelehnt. Ein wesentliches Leitprinzip des Scrum ist, dass nur im Moment benötigte Features entwickelt werden. Ein weiteres Leitprinzip sind immer wiederkehrende Meetings nach einem festgelegten Zeitraum. Diese dienen zur

[4]CATIA V5 ist eine prozessorientierte Konstruktionssoftware um dreidimensionale Draht-, Flächen- und Volumenmodelle darzustellen und daraus zweidimensionale Entwürfe abzuleiten.

Verbreitung des derzeitigen Implementierungsstandes der Software, zur Aufnahme neuer Ideen und zur Hilfestellung bei eventuellen Problemen. In der Projektgruppe des Virtuellen Satelliten findet ein solches Treffen wöchentlich statt.
Intern besitzt der VirSat einen modularen aus Plug-Ins[5] bestehenden Aufbau. Durch diese Struktur ist er leicht nach den Wünschen des Kunden zusammensetzbar und erweiterbar. Zu den einzelnen Plug-Ins, mit dem in ihnen befindlichen Quellcode, gibt es jeweils ein Plug-In-Fragment[6], welches die JUnit-Tests zum Testen der Funktionalitäten des Plug-Ins in sich beinhaltet. JUnit ist ein einfaches Framework um wiederholbare Tests für die Programmiersprache Java zu kreieren.
Damit aus den einzelnen Plug-Ins eine funktionsfähige Desktop-Anwendung wird, wird der VirSat nach dem Eclipse RCP (Rich Client Plattform) entwickelt. Dieses stellt ein Grundgerüst bereit, das um eigene Anwendungsfunktionalitäten erweitert werden kann. Dabei werden die alten Strukturen des Datenmodells mit Hilfe des Kompatibilität Layer in die neueren Eclipse Plattformen integriert. [Ebe11]
Damit mehrere Personen am Quellcode des VirSats arbeiten können steht er unter ständiger Versionskontrolle mittels Apache **S**ub**v**ersio**n**. Durch das Build-Management Tool Maven kann die Software kompiliert und somit lauffähig gemacht werden. Nebenbei werden die einzelnen Module getestet. Mit Hilfe der Continues-Integration-Applikation Jenkins werden alle für die Software notwendigen Bibliotheken und Frameworks heruntergeladen, die Software gebaut und alle Tests durchlaufen. Die Entwickler erhalten nach Beendigung einen Bericht über die Stabilität der aktuellen Version der Software.
Alle benutzerdefinierten Einstellungen, Erkenntnisse und Informationen werden in einem zentralen Datenmodell gespeichert. Das Datenmodell wird als Ecore Metamodell dargestellt. Das Ecore-Metamodell (das vorher definierte Metamodell) in Verbindung mit dem Gen-Metamodell (das generierte Metamodell) bilden das Eclipse Modeling Framework (EMF). Dieses beschreibt den inneren Aufbau inklusive der Zusammenhänge der einzelnen Komponenten eines Datenmodells untereinander. Das vereinfachte Datenmodell des VirSats im Vergleich mit dem StudyNavigator View im VirSat ist in Abbildung 8 zu sehen. Die Oberkomponente des Datenmodells im VirSat ist ein *StudyRepository*. Im StudyNavigator wird es mit IEEE_Study_1 bezeichnet. Das

[5] Ein PlugIn in Eclipse ist eine Komponente, die eine bestimmte Art von Service in der Eclipse-Umgebung bietet. Die Eclipse Runtime bietet die Möglichkeit die Aktivierung solcher Plug-Ins zu verwalten. [Com03]

[6] Ein Plug-In-Fragment ist ein Unter-Plug-In, welches von einem Haupt-Plug-In abhängt. Es kann auf die Funktionalitäten des Haupt-Plug-Ins zugreifen, ist aber allein nicht lauffähig. Genutzt wird es vor allem zum Entwickeln von Plug-In-Tests.

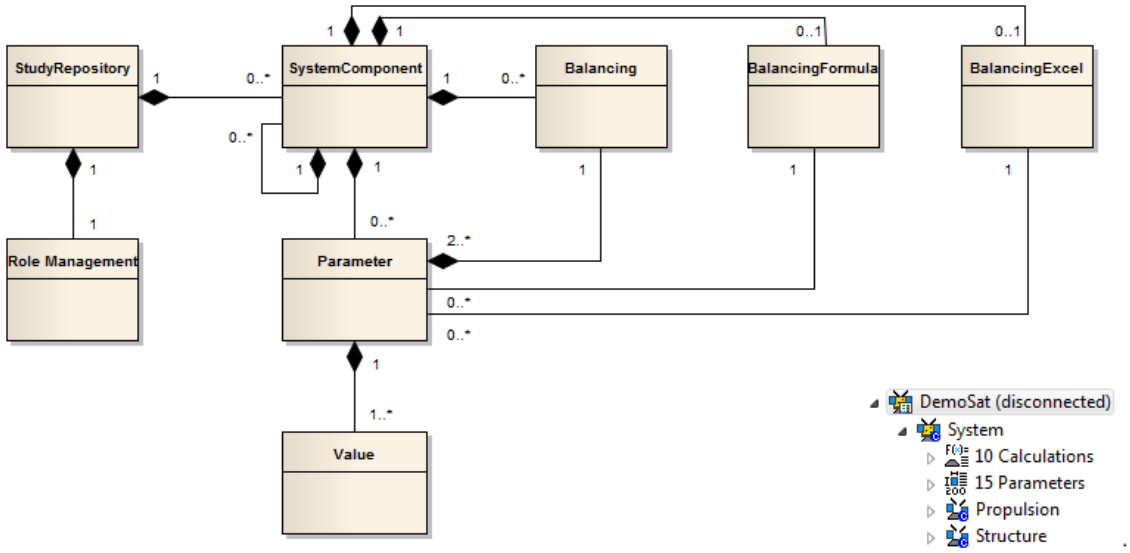

Abbildung 8: Vereinfachtes Datenmodell des Virtuellen Satelliten mit dem Aufbau einer Studie

StudyRepository soll die ganze Studie abbilden. Ein StudyRepository kann mehrere *SystemComponents* in sich beinhalten. Eine **S**ystem**C**omponent (SC) spiegelt hierbei die Komponenten und Subkomponenten, wie AHAB und AOCS, wieder. Diese SCs können wiederum einige SCs enthalten. Sie werden im StudyNavigator View als Unterkomponenten, wie Computer zu AOCS, dargestellt. Eine SC beinhaltet weiterhin mehrere Parameter und Berechnungen aus dem Parameters-Tabulator, hier *Balancing* genannt (Vergleich Unterkapitel 2.1). All jene werden auch im StudyNavigator als Unterpunkte zu den Komponenten, wie beispielsweise bei AOCS, angezeigt. Eine Komponente kann weiterhin höchstens eine Referenz auf ein ExcelSheet (*BalancingExcel*) und eine auf eine Berechnung mit Hilfe des Calculation-Tabs (*BalancingFormula*) besitzen. Eine solche Berechnung kann aus mehreren Einzelschritten bestehen. Sie werden nicht im StudyNavigator View angezeigt. Die Parameter der Komponenten können für alle drei Berechnungsarten als Input genutzt oder als Output berechnet werden. Im VirSat ist es weiterhin möglich die eigenen Parameter für andere Komponenten sichtbar und verwendbar zu machen. Jedes Hauptkomponente (Vergleich AHAB in Abbildung 8) besitzt genau ein Rollenmanagement. Im Datenmodell sind noch weitere Einstellungen und Elemente abgespeichert, aber sie haben für diese Arbeit keine weitere Bedeutung.
Im VirSat reagieren alle Editoren auf Veränderungen im Datenmodell. Wird beispielsweise ein Parameter im einem der drei Berechnungstypen gesetzt, so wird sein neuer Wert automatisch in den Parameters-Part eingetragen. Diese Änderungen sind

selbstverständlich auch im Datenmodell abgespeichert.
Entwickelt wird die Software nach dem DLR-Entwicklungsprozess. Dieser ist stark an das Scrum-Prinzip angelehnt. Ein wesentliches Leitprinzip des Scrum ist, dass nur im Moment benötigte Features entwickelt werden. Ein weiteres Leitprinzip sind immer wiederkehrende Meetings nach einem festgelegten Zeitraum. Diese dienen zur Verbreitung des derzeitigen Implementierungsstandes der Software, zur Aufnahme neuer Ideen und zur Hilfestellung bei eventuellen Problemen. In der Projektgruppe des Virtuellen Satelliten findet ein solches Treffen wöchentlich statt.

3 Stand der Forschung

In den Ingenieurswissenschaften werden oftmals noch nie dagewesene Konzepte umgesetzt. Diese Konzepte werden meist in einer vorangehenden Planungsphase modelliert. Auch in dieser Arbeit wird ein Konzept umgesetzt, welches so noch nicht vorhanden ist. Die Modellierung in dieser Arbeit besteht aus der Verbesserung einer bereits bestehenden grafischen Anzeige von Elementen und zur Umwandlung dieser Anzeige in einen grafischen Editor. Dabei werden bereits existierende Tools eingesetzt. In diesem Kapitel werden einige grafischen Darstellungsformen im Ingenieurstechnischen Hintergrund erklärt, sich mit diesen Aufgaben beschäftigenden Frameworks vorgestellt und die für die Arbeit relevanten Grundkenntnisse vermittelt.
Der Abschnitt 3.1 beschreibt hierzu den Einsatz von Diagrammen im wissenschaftlichen und ingenieurstechnischem Bereich. Insbesondere einzelne Frameworks zur Erstellung solcher Elemente werden hierbei genauer betrachtet. Die Kapitel 3.3 und 3.4 erläutern die Funktionsweise, Architektur und Verwendung des grafischen Modellierungsframeworks Graphiti und des Layoutframeworks **K**iel **I**ntegrated **E**nvironment for **L**ayout **E**clipse **R**ichClient (KIELER). Der Aufbau grafischer Editoren im Satellitendesign wird in Kapitel 3.5 näher erläutert. Dabei wird gesondert auf die Standards **B**lock**d**efinitions**d**iagramm (BDD) und **I**nternes **B**lock**d**iagramm (IBD) eingegangen. Im Abschnitt 3.5 wird die Ausgangssituation anhand der vorgehenden Arbeiten zusammengefasst.

3.1 Diagrammarten im wissenschaftlichen und ingenieurstechnischen Bereich

Informationen sind häufig besser verständlich, wenn diese in einfacher Form grafisch dargestellt werden. Durch die Visualisierung können sie zudem oft schneller erfasst werden. Dieser Effekt wird auch oft in der Werbebranche genutzt, da der Mensch eine einfache Abbildung mit kurzen Texten schneller verinnerlicht, als einen anspruchsvollen Text ohne Abbildung. Deshalb werden grafische Elemente auch im wissenschaftlichen und ingenieurstechnischen Bereich gern genutzt. Ein Beispiel dafür sind die Schaltpläne. Abbildung 9 zeigt den Unterschied zwischen einer textuellen Beschreibung eines Schaltplans und einer visuellen Form eines Schaltplans. Der Text erfasst hierbei die gleichen Informationen wie die Grafik, jedoch in einer andern Form. Zur

besseren Verdeutlichung ist Abbildung 9 zu betrachten. Hierbei wird deutlich, dass die

Der Stromkreis besteht aus einem Verbraucher, einem Taster und einem Akkumulator, welche in Reihe geschalten sind. Eine Buchse ist mit dem Akku parallel geschalten

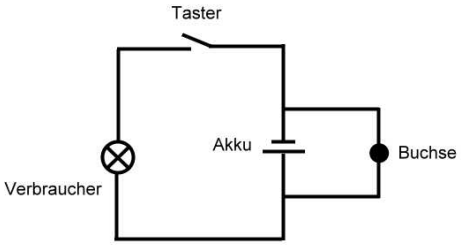

Abbildung 9: Beschreibung eines Schaltplans in textueller und in grafischer Form

Anordnung von Elementen mittels einer grafischen Darstellung leichter zu verstehen ist, als in textueller Form.
Ein weiteres Beispiel für eine Diagrammart im wissenschaftlichen Bereich ist das Datenflussdiagramm. Es beschreibt den Fluss, die Veränderung und die Bereitstellung von Daten innerhalb eines Abschnitts. Hierbei wird der Ablauf der einzelnen Vorgänge mit berücksichtigt. Ein Beispiel eines Datenflussdiagramms ist in Abbildung 10 zu sehen. Dieses stellt den Datenfluss in Anlehnung an die Raketengleichung von Ziolkowki dar. Die Raktengleichung besagt, dass aufgrund des definierten Orbits das

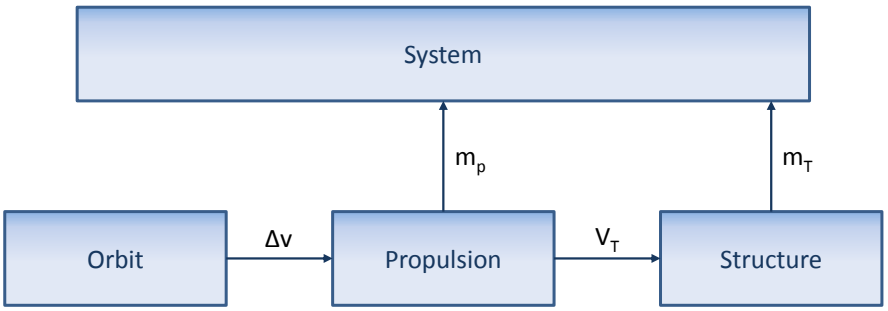

Abbildung 10: Datenflussdiagramm der Gleichung von Ziolkowski

benötigte Volumen des Treibstoffs festgelegt ist. Daran kann der zuständige Raumfahrtingenieur für den Antrieb (Propulsion) den geeignetsten Treibstoff ermitteln und das Volumen des Tanks bestimmen. Mit Hilfe der Spezifikation des Volumens des Tanks ist der Raumfahrtingenieur der Struktur (Structure) in der Lage eine geeignete Form und Art des Tanks zu bestimmen. Die Massen des Treibstoffs und des Tanks werden der Gesamtmasse des Satelliten hinzugefügt. [SFQG12]

3.2 Tools zur Visualisierung von Informationen

Ingenieure aller Fachrichtungen arbeiten oftmals mit großen Datenmengen. Diese können meist nur mittels Grafiken oder Diagrammen veranschaulicht werden. Zur Erstellung solcher Grafiken und Diagramme gibt es mehrere Tools. Eines, was meist bei den Datenflussdiagrammen verwendet wird ist beispielsweise der yEd Graph-Editor in Verbindung mit yFiles. Beispielhaft fand es in [Wel08] zur Darstellung des strukturellen Aufbaus von Clusterbäumen in der Biochemie Anwendung.
Diagramme im Software Engineering können auch mit dem Eclipse-basierten Modell-Entwicklungs-Programm Papyrus erstellt werden. 2012 kam Papyrus beispielsweise zur Erstellung von SysML-Diagrammen im Flugzeugwesen zum Einsatz. [DIL+12]
Auch wurden bereits Diagramme in Anlehnung an SysML mit Hilfe des Graphiti- Frameworks erstellt. Graphiti ist, wie Papyrus, ein Eclipse-basiertes Grafik-Framework. Es ermöglicht die schnelle Entwicklung von Diagrammeditoren für Domain-Modelle. So wird Graphiti beispielsweise zur Konkretisierung einer Modellierungssprache eingesetzt. [Kle12, Mül13]

3.3 Graphiti Framework zur Erstellung eines grafischen Editors

Der Quellcode des grafischen Modellierungsframeworks ***Graphi**cal **T**ooling **I**nfrastructure* wurde 2009 von der Firma SAP an die Eclipse-Communitiy übergeben und wird seitdem durch diese weiterentwickelt. Das Framework vereinfacht die Erstellung homogener grafischer Editoren aus einem bereits bestehenden Datenmodell. Weiterhin dient es zur Verbergung der Komplexität des **G**raphical **E**diting **F**ramework (GEF) und zur Integration des **E**clipse **M**odeling **F**ramework (EMF) mit GEF[7]. Der VirSat basiert auf Eclipse und in ihm wird EMF (und GEF) genutzt. Daher ist das Framework ideal um eine vereinfachte Darstellung des Datenmodells zu erstellen.

3.3.1 Architektur des Graphiti-Frameworks

Grundlage des Frameworks ist eine Eclipse-basierte Anwendung und Implementierung mit Hilfe der Programmiersprache Java. Der VirSat wurde auf Grundlage dieser

[7] GEF stellt eine Technologie zur Erstellung von grafischen Editoren in einer Eclipse-Umgebung bereit. EMF ist ein Modellierungsframework zur Erstellung von Applikationen basierend auf einem Datenmodell.

Frameworks entwickelt und man kann sie somit gut kombinieren. (siehe Kapitel 2.2). [The13, Kle12]
Die Architektur des Graphiti-Frameworks ist in Abbildung 11 zu finden. Die Hauptelemente sind hierbei die Komponenten *Rendering Engine*, die *Interaction Component* und der *Diagram Type Agent*.

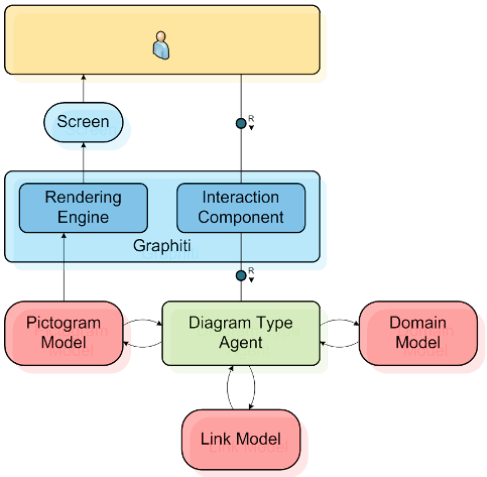

Abbildung 11: Vereinfachte Darstellung der Architektur des Graphiti-Frameworks [The13]

Die Graphiti-Runtime stellt den Kern des Frameworks da. Es besteht aus der Rendering Engine und der Interaction Component. Die Interaction Component ist für die Verarbeitung aller Nutzereingaben im Editor zuständig. Dazu gehören Mausklicks, Anpassung der Größe und Drag-and-Drop, sowie das Erstellen und Löschen von Elementen. Die Rendering Engine ist für das Darstellen der aktuellen Daten auf dem Bildschirm zuständig.
Der Diagram Type Agent ist Datenmodell-spezifisch und wird vom Entwickler erstellt und angepasst. Angesprochen wird der Agent über eine standardisierte Schnittstelle durch die Interaction Component. Darin sind alle Funktionalitäten des Editors definiert. Dazu gehören beispielsweise das Erstellen neuer Elemente im Datenmodell und im Diagramm und das Verschieben oder Löschen von Elementen.
Im FeatureProvider werden die Aktionen, wie zum Beispiel Hinzufügen und Entfernen von Elementen, definiert. Dabei wird unterschieden, ob es sich um das Hinzufügen/Entfernen von Elementen im Diagramm (Add-/RemoveFeature) oder vom Datenmodell und Diagramm (Create-/DeleteFeature) handelt. Bei der Integration des Frameworks muss jedes Hinzufügen, Löschen usw. von unterschiedlichen Elementen auch in einzelnen Features definiert werden. Im ToolBehaviorProvider sind die

Interaktionen des Benutzers mit der Maus, wie beispielsweise der ToolTip oder der Doppelklick, festgelegt. Die Rendering Engine ist mit dem FeatureProvider und dem ToolBehaviorProvider gekoppelt. Auch diese können je nach Bedarf selbst definiert werden. Wenn keine eigene Definition der Features vorhanden is, dann wird der Standard des Frameworks genutzt.
Bei allen Interaktionen mit dem Diagram Type Agent werden die Modellierungsdaten des Editors modifiziert. Zu ihnen gehören das Pictogram Model (Piktogrammmodell), inklusive dem Link Model (Verbindungsmodell), und das Domain Model (Wirkungsmodell) (siehe Abbildung 12).

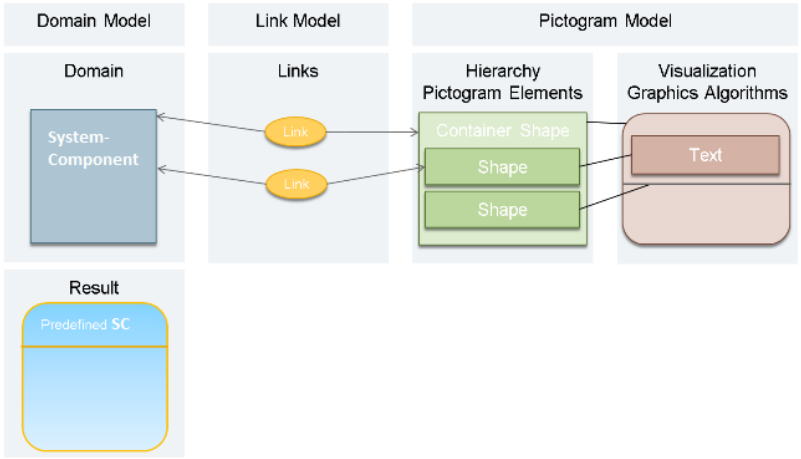

Abbildung 12: Zusammenspiel zwischen dem Pictogram-Model, Link-Model und Domain-Model [The13]

Das Piktogrammmodel enthält alle Informationen, die im Diagramm angezeigt werden. Ein Piktogrammelement wird genau dann als Containershape bezeichnet, wenn es mehrere Piktogramme in sich beinhaltet (siehe Abbildung 12). Das Wirkungsmodell beschreibt das Element aus dem Datenmodell, welches durch das Piktogrammelement dargestellt wird. Das Verbindungsmodell beinhaltet die Verknüpfung zwischen den beiden Modellen. Das Result der obigen Abbildung stellt eine so definierte Komponente im grafischen Editor dar. Es besteht im Piktogrammmodell aus den einzelnen grafischen Elementen, welche als Komplettpaket mit einem Objekt des Wirkungsmodells gekoppelt sind. In diesem Beispiel ist das Objekt eine **S**ystem-**C**omponent (siehe Kapitel 2.2).
Mit Hilfe des Frameworks kann entweder das gesamte Datenmodell oder auch nur Teilaspekte abgebildet werden. Abbildung 13 beinhaltet die Darstellung eines Datenmodells in unterschiedlichen Diagrammen. Im oberen Teil der Abbildung ist dabei das zugrunde liegende Datenmodell zu sehen und im unteren Abschnitt sind die dazu-

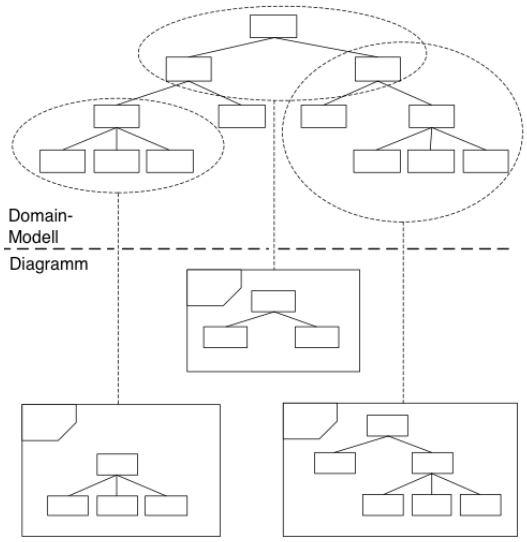

Abbildung 13: Darstellung von Teilmengen des Datenmodells mit Hilfe des Graphiti-Frameworks (Bildquelle: [Kle12])

gehörigen Diagramme abgebildet. Die einzelnen Elemente des Modells können in verschiedenen Diagrammen dargestellt werden.
Insgesamt bietet des Framework eine Laufzeit-orientierte Architektur, eine zentralisierte und nach Features aufgeteilte Logik und einen standardisierten Aufbau des Editors. Die Integration des Frameworks in eine Applikation erfolgt über eine Implementierung gegen dessen Schnittstelle. [The13]

3.3.2 Verwendung des Graphiti-Frameworks

Das Graphiti-Framework wurde erst vor vier Jahren zur Verfügung gestellt und wird mittlerweile in einer Vielzahl von Anwendungsgebieten verwenden. Ein Beispiel ist das Erstellen eines Workflowmanagementsystems in der Bioinformatik. [AAC+13] Unter einem Workflow versteht man in diesem Zusammenhang die Automatisierung eines Prozesses, bei dem Dokumente, Informationen oder Aufgaben zwischen den Aktivitäten, aus denen der Prozess aufgebaut ist, ausgetauscht werden. Um den korrekten Ablauf zu kontrollieren, wird der Datenfluss mit Regeln gesteuert. Diese Regeln wurden auch bei der Integration des Graphiti-Frameworks beachtet.
Ein weiteres Beispiel für die Verwendung des Graphiti-Frameworks ist in [BMJ12] zu sehen. In diesem technischen Bericht wurde das Framework aufgrund des Unverständnisses gegenüber dem GEF genutzt. Es dient hierbei zur Darstellung des Work-

flows für Anwendungen, welche sich mit dem Zugriff und der Verwaltung von verteilten Daten beschäftigen.
Im Eclipse-Projekt *eTrice* kommt das Visualisierungsframework Graphiti auch zum Einsatz. eTrice ist eine Implementierung der **R**eal Time **O**bject **O**riented **M**odeling (ROOM)-Modellierungssprache für ereignisgesteuerte, verteilte, sowie eingebettete Systeme. Graphiti dient hierbei zur Erstellung grafischer Editoren für die ROOM-Modelle. Das Graphiti-Framework bietet standardmäßig keine Integration der automatischen Anordnung von Elementen. Hierfür gibt es mehrere Ansätze. Ein Ausgangspunkt ist die Implementierung von Layoutalgorithmen in die Software. Dies ist jedoch sehr umständlich und es gibt nicht genügend Auswahl von unterschiedlichen Layouts. Eine bessere Möglichkeit ist die Integration des KIELER-Layoutframeworks. Dieses wurde ebenfalls im eTrice-Projekt integriert. [Ecl13]

3.4 KIELER-Framework zur Integration von Layoutalgorithmen

Das KIELER-Framwork ist ein Forschungsprojekt, welches zur Verbesserung grafischer Modellierungsdesigns und komplexen Systemen dient. Die Grundidee in diesem Projekt ist die konsequente Umsetzung des automatischen Layouts, wobei auch die Modellierungsumgebung mit beachtet wird. Es ist auch auf Diagramme, erstellt mit Hilfe des grafischen Modellierungsframeworks Graphiti, anwendbar.
Der Ablauf der Anwendung des Layoutalgorithmus auf den Inhalt des grafischen Editors ist in Abbildung 14 zu sehen. Zunächst wird das bestehende Diagramm ausgelesen und in eine für KIELER verständliche Form gebracht - das Notierungsmodell (Notation Model). Dies besteht aus einem KGraphen, welcher wiederum aus den Elementen KNode, KEdge und KPort gebildet wird. Bei der Umwandlung kommt der Diagram Layout Manager zum Einsatz. Dieser bestimmt, bei welchem Element es sich um eine KNode, KEdge oder KPort handelt und fügt diese dem Notationsmodell hinzu. Zur Veranschaulichung der Elemente wurde die Abbildung 15 entworfen. Eine KNode repräsentiert hierbei einen Knotenpunkt im Graphen. Dieser kann mehrere KPorts beinhalten. Sie repräsentieren die Anfangs- und Endpunkte der Verbindungslinien (KEdges). Dabei kann eine KNode wiederum mehrere KNodes in sich beinhalten. Eine KEdge muss immer genau mit zwei KPorts verbunden sein.
Mit Hilfe der Layout View kann der Benutzer den gewünschten Algorithmus zur Berechnung des Layouts auswählen und nach seinen belieben abändern. So ist es dort auch beispielsweise möglich die Breite zwischen der neuen Positionierung zu definie-

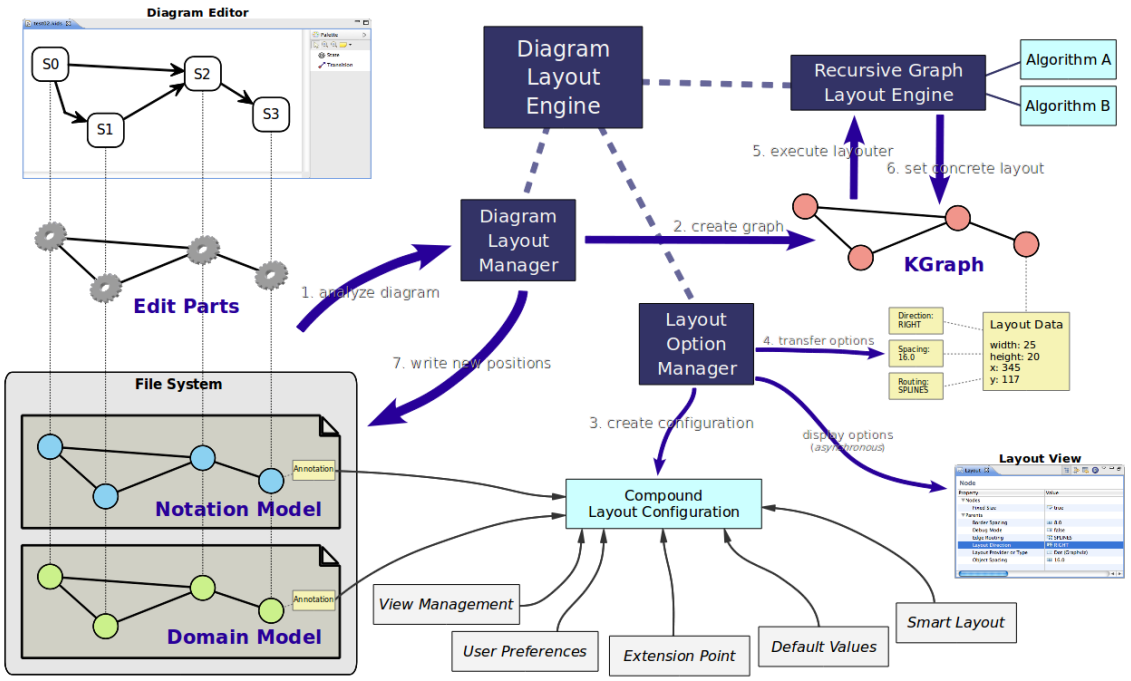

Abbildung 14: Ablauf des KIELER-Layoutalgorithmus

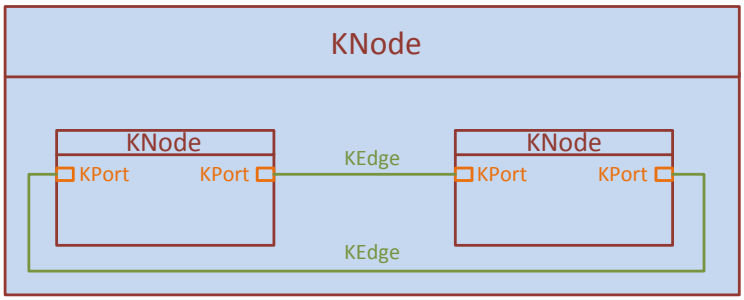

Abbildung 15: Veranschaulichung der Elemente im KGraph

ren. Vom Diagram Layout Manager wird die Layout Engine aufgerufen, in welcher die neuen Positionen und die neue Größe der Elemente durch den gesetzten Algorithmus berechnet werden. Die Rückführung zum Diagramm erfolgt wieder über den Diagram Layout Manager. Dieser überschreibt die alten Positionen und Größen mit den neu berechneten.

Bei der Verwendung des KIELER-Layoutframeworks in Verbindung mit Graphiti ist wichtig zu beachten, dass die einzelnen Elemente in Graphiti einen bestimmten Aufbau besitzen. Dieser kommt der Abbildung 16 gleich. Andernfalls wird das Element für das Layouting nicht beachtet. [Chr13] Die Graphiti-Elemente müssen hierbei aus einem Hauptpiktogramm bestehen, welches sichtbar ist. An dieses werden einzel-

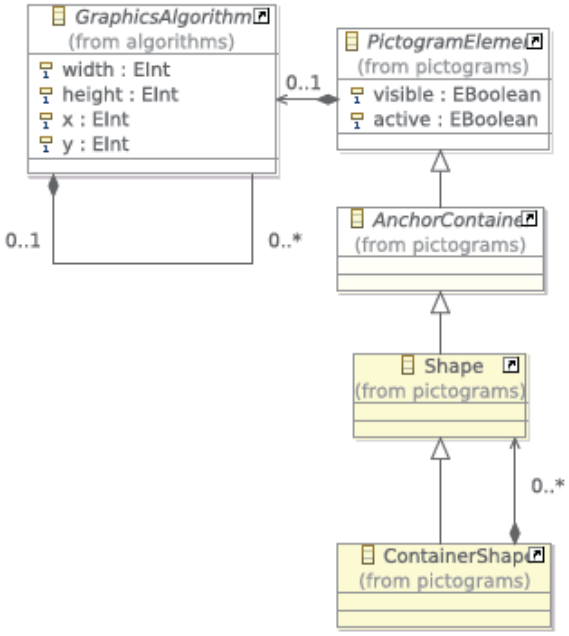

Abbildung 16: Redundanz Graphiti [Rie12]

ne Shapes (z.B.: Text und geometrische Formen) angefügt. Besitzt das Hauptpiktogramm keine Shapes, so ist es für das KIELER-Framework kein korrekter Aufbau und es wird für das Layouting nicht beachtet.

3.5 Visualisierung des Systemdesigns in Anlehnung an SysML

Zur Beschreibung von Komponenten im Satellitendesign kommen häufig **I**nterface **C**ontrol **D**ocuments (ICD) zum Einsatz. Diese Dokumente beschreiben, wie eine Schnittstelle eines Systems aufgebaut ist und wie sie angesprochen werden kann. Der interne Aufbau ist für diese Dokumente nicht von Bedeutung. Die Designvorgaben liegen in textueller und grafischer Form vor. Die Erstellung der grafischen Form der Schnittstellendefinitionen ist meist mit viel Zeitaufwand verbunden. Hinzu kommt, dass die Diagramme bei jeder Änderung per Hand aktuallisiert werden müssen. Dies sind stupide Arbeiten, welche gemacht werden müssen, aber trotzdem kostbare Arbeitszeit der Ingenieure kosten. Zusätzlich sind sie sehr fehleranfällig, da sie ständig aktualisiert werden müssen. Ein Algorithmus zur automatischen Generierung eines Diagramms anhand der zuvor definierten Schnittstellenspezifikationen würde den Ingenieuren eine erhebliche Zeitersparnis bringen.

Das Ziel bei der Schnittstellenbeschreibung ist die Unterteilung der Hauptkomponente in mehrere Unterkomponenten und die korrekte Interaktion zwischen mehreren Elementen. Die Entwickler/Ingenieure können die Hauptkomponente genauer spezifizieren, ohne die genaue Funktionsweise der einzelnen Unterkomponenten zu kennen, da diese durch Schnittstellen entsprechende Dienste bereitstellen. Nicht allein die physikalischen Eigenschaften einer Komponente werden hierbei beschrieben, sondern auch die logischen Kopplungen zum Modell werden in ihm festgelegt. [PBBSA94, Par07]

3.5.1 Blockdefintionsdiagramme

Zur besseren Verständlichkeit der Spezifikationen werden **B**lock**d**efinitions**d**iagramme eingesetzt. BDD kennzeichnen in grafischer Form die Ein- und Ausgänge einer Komponente. Insbesondere die Funktionsabläufe, funktionalen Beziehungen und Verflechtungen von Funktionseinheiten der Schnittstellen, sowie deren hierarchische Beziehungen werden hierbei beschrieben. [Dun12, FMS11]
Ein Beispiel für ein solches BDD ist in Abbildung 17 zu sehen. Dieses BDD zeigt

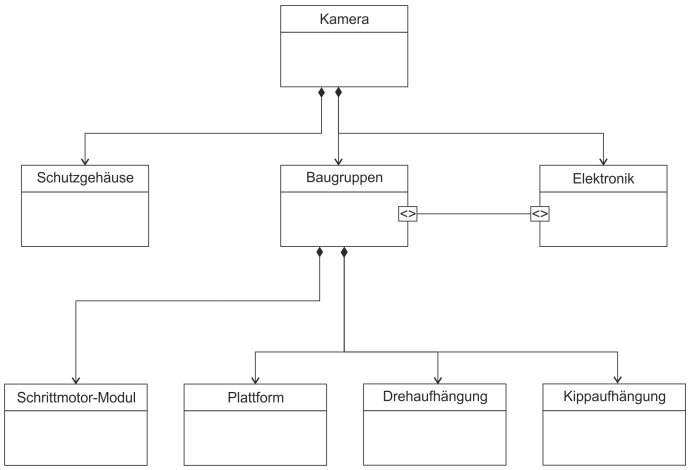

Abbildung 17: Blockdefintionsdiagramm (in Anlehnung an [FMS11])

die hierarchische Zusammensetzung einer Kamera. Die Notationen beschreiben die Zusammenhänge zwischen den einzelnen Komponenten und der Kamera. [FMS11] Es beschreibt, dass eine Kamera aus den Elementen Schutzgehäuse, Baugruppen und Elektronik aufgebaut ist. Dabei ist definiert, dass es ohne die Kamera insgesamt auch kein Bauteil gibt (Kompositionsdefinition). Weiterhin wird in dieser Abbildung beschrieben, dass es einen Zusammenhang zwischen den Baugruppen und der Elek-

tronik gibt. Die Schnittstellen werden hierbei als Ports gekennzeichnet. Beide können jedoch unabhängig voneinander bestehen.

3.5.2 Interne Blockdiagramme

Teilweise benötigt ein Ingenieur nicht nur das Wissen über die Schnittstellendefinitionen einer Komponente, sondern er muss auch über den inneren Aufbau inklusive Funktionsweise informiert sein. Zur grafischen Veranschaulichung wurde zu diesem Komplex eine Diagrammart von SysML entwickelt: das **I**nternes **B**lock**d**iagramm (IBD). Dieses setzt auf die Spezifikationen des BDD auf, indem es zusätzlich zu den Schnittstellendefinitionen auch die interne Struktur und die Verbindungen zwischen einzelnen Teilen eines Blocks bestimmt. Zusätzlich finden sich in diesem Diagramm Elemente aus den Datenflussdiagrammen, die die Darstellung von Prozessen und Abläufen ermöglicht (siehe Kapitel 3.2).
Abbildung 18 beschreibt ein IBD der bereits oben genannten Kamera. Es besagt, dass sich das Schutzgehäuse auf die aktuelle Lichtintensität einstellt und es somit mehr oder weniger Licht auf das Objektiv lässt. Mit Hilfe dieses Mechanismus sind auch Bilder bei starker Dunkelheit möglich. [FMS11]

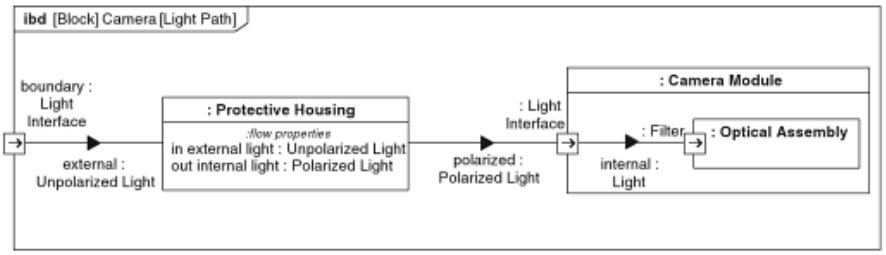

Abbildung 18: Internes Blockdiagramm [FMS11]

3.5.3 SysML und UML2.0

Das **B**lock**d**efinitions**d**iagramm und das **I**nterne **B**lock**d**iagramm stammen aus der **Sys**tems **M**odeling **L**anguage (SysML)-Spezifikation. SysML ist eine Erweiterung der **U**nified **M**odeling **L**anguage **2.0** (UML2.0) ist. Dies ist eine grafische Modellierungssprache im Bereich der Softwareentwicklung und eine Weiterentwicklung der UML 1.0. UML 1.0 wurde von der Object Management Group (**OMG**) 1997 als Standard

akzeptiert und weiter gepflegt. Im Jahr 2005 wurde der ISO-Standard UML2.0 herausgegeben, da die verschiedenen Formen von UML zu unterschiedlich und zu schwer zu spezifizieren waren. Die Ziele von UML2.0 bestehen darin, ein breites Anwendungsgebiet zu schaffen und die Intuitivität zu steigern. Hierbei wurde insbesondere auf die bessere Verständlichkeit der Notationsmittel eingegangen.
UML2.0 dient dazu, Modelle für Softwaresysteme zu erstellen. Dank ihrer Modularität kann sie für unterschiedliche Teilbereiche eingesetzt werden. Unterschieden wird der Standard in zwei große Teilbereiche: die Struktur- und die Verhaltensdiagramme. Die Strukturdiagramme beschreiben den Aufbau eines Systems. Zu ihnen gehören beispielsweise die Klassendiagramme, Kompositionsstrukturdiagramme und Paketdiagramme. Möchte man den Aufbau des Datenmodells im VirSat grafisch in Anlehnung an den UML2.0-Standard darstellen, so geschieht das nach dem Leitfaden der Klassendiagramme (siehe Abbildung 8). Die Verhaltensdiagramme beschreiben hingegen die Abläufe im System. Die bekanntesten Beispiele dafür sind die Aktivitätsdiagramme, Use Case Diagramme und Interaktionsdiagramme. Diese sollen Geschäftsprozesse, komplexe Algorithmen, usw. darstellen.
Im Systems Engineering[8] kann der UML2.0-Standard ohne Erweiterung nicht angewendet werden, da nicht genügend Notationen und Elemente spezifiziert sind. Hierzu wurde im Jahr 2006 die Erweiterung SysML von der OMG offiziell angenommen. SysML ist dem UML2.0-Standard sehr ähnlich, besitzt aber notwendige Erweiterungen und schließt für den Verwendungszweck des Systems Engineering nicht notwendige Methoden aus. Den Vergleich zwischen den Diagrammarten der UML2.0 und SysML ist in Abbildung 19 ersichtlich.
Die in den Kapiteln 3.5.1 und 3.5.2 definierten Diagrammarten **B**lock**d**efinitions**d**iagramm und **I**nternes **B**lock**d**iagramm gehören zu den Strukturdiagrammen (siehe Abbildung 19). Sie wurden bereits in UML2.0 definiert, aber in SysML abgeändert. In UML2.0 trugen sie noch die Namen: Klassendiagramm und Kompositionsstrukturdiagramme. [Cor12, Wei06, JRZ+04, Mül13]
Diagramme in Anlehnung an SysML werden vielseitig genutzt und können auf unterschiedliche Weise erstellt werden. Ein Beispiel ist die Erstellung solcher Diagramme mit Hilfe der Applikation *MagicDraw* in Verbindung mit dem SysML-Plugin. Als Beispiel ist dafür [K. 08] zu nennen, wo es zur Beschreibung eines invertierten Pendels genutzt wird.

[8] "*Systems Engineering* ist ein interdisziplinärer Ansatz mit dem Ziel, erfolgreiche Systeme zu realisieren." [Inf13]

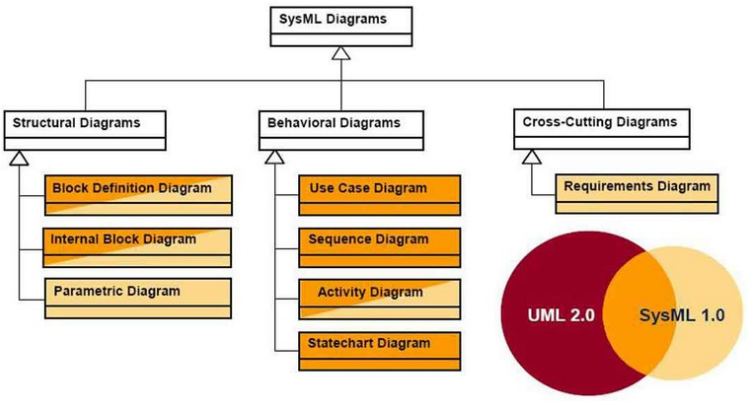

Abbildung 19: Vergleich der Diagrammarten zwischen SysML und UML2.0 [Inf13]

3.6 Bisherige Arbeit

Die wichtigste Vorabeit zu dieser Bachelorarbeit war meine vorangehenden Praxisphase mit dem Thema der Erstellung von Diagrammen im VirSat. [Mül13] Dabei wurde bereits eine Darstellung des Datenmodells in Anlehnung an SysML mit Hilfe des grafischen Modellierungsframeworks *Graphiti* in den Virtuellen Satelliten integriert. Ziel dieser Arbeit war es die Raumfahrtingenieure bei der Erstellung der **I**nterface **C**ontrol **D**ocument (ICD), wie in Kapitel 3.5 beschrieben, zu unterstützen. Das ICD beinhaltet Diagramme nach den Standards BDD und IBD. Der Unterschied beider Diagramme liegt darin, dass BDD nur die Schnittstellen und IBD die Schnittstellen inklusive innerem Aufbau beschreiben, wie in den Kapiteln 3.5.1 und 3.5.2 ausführlich beschrieben.

Mit Hilfe des Kontextmenüs und dem Command-Framework[9] werden solche Diagramme erstellt und geöffnet. Dabei wird das zugrunde liegende Datenmodell des VirSats mit Hilfe eines selbst erstellten Rechenalgorithmus' ausgelesen und nach bestimmten Elementarwerten gefiltert. Diese sind die in einer Komponente vorhandene Parameter und Berechnungen (siehe Abbildung 8). Die Parameter werden nach ihrer Funktionalität unterschieden. Sie können aus einer anderen Komponente oder aus der eigenen stammen und für interne Berechnungen genutzt werden, aus der eigenen stammen und ungenutzt bleiben und/oder für andere Komponenten nutzbar gemacht werden. Diese Möglichkeiten werden im Diagramm abgebildet. Damit der Nutzen der Parameter für den Anwender besser ersichtlich ist, wurden Verbindungslinien zwischen

[9] Das Command-Framework ist ein Eclipse-spezifisches Framework um auf Interaktionen des Benutzers zu agieren. Insbesondere Applikationen basierend auf RCP (Vergleich Kapitel 2.2) wird es angewendet. [C+11]

diesen und ihrer Verwendung integriert. Das Routing zur korrekten Anordnung der Verbindungslinien wurde in dieser Arbeit noch nicht umgesetzt. Die in der Vorarbeit

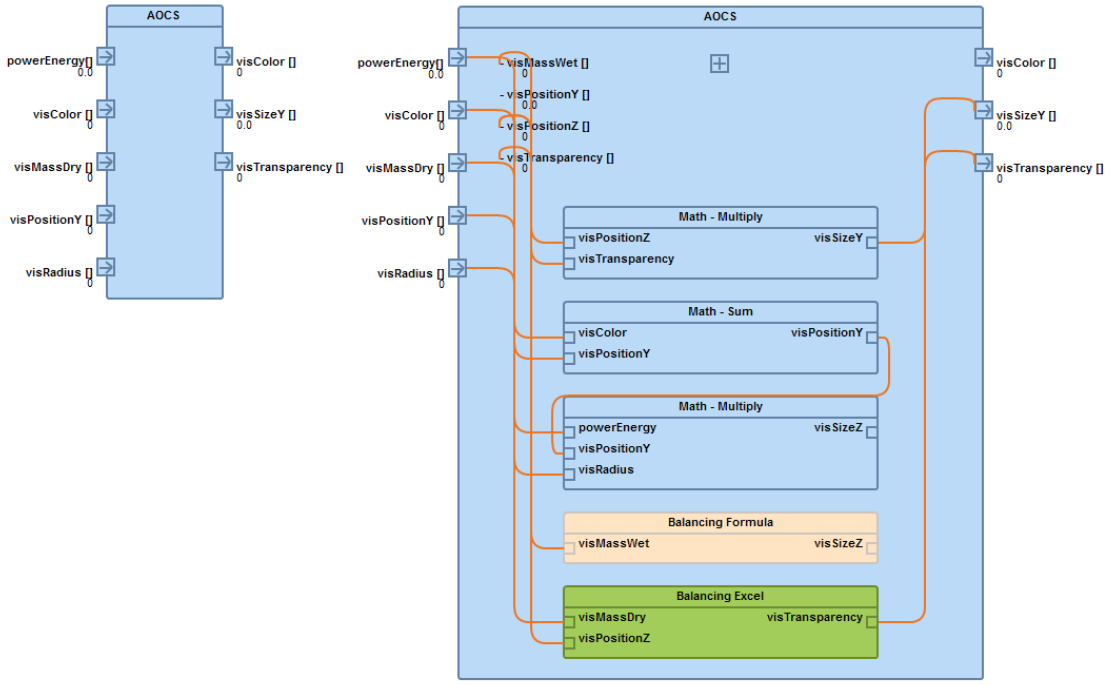

Abbildung 20: Blockdefinitionsdiagramm und Internes Blockdiagramm im Virtuellen Satelliten

erstellten Diagramme sind in Abbildung 20 ersichtlich. Ihre Grundlage ist das Datenmodell, welches aus den zusammengetragenen im VirSat generiert wird.
Beide Diagramme befinden sich in dieser Arbeit in einer Datei. Der Übergang zwischen ihnen erfolgt über einen Doppelklick auf die Hauptkomponente (in diesem Fall die Komponente AOCS). Dabei wird das komplette Diagramm gelöscht und neu gezeichnet. Bei Änderungen im Datenmodell müssen die Diagramme per Hand neu erstellt werden.

4 Entwurf und Implementierung

Die in den **I**nterface **C**ontrol **D**ocuments verwendeten **B**lock**D**efinitions**d**iagramme und die damit verbundenen **I**nternen **B**lock**d**iagramme (siehe Kapitel 3.5) wurden in der Vorarbeit zu dieser in den **Vir**tuellen **Sat**elliten integriert. Diese Diagramme sollen alle Schnittstelleninformationen, inklusive dem inneren Aufbau beim IBD, in grafischer Form darstellen. Im VirSat besteht eine Komponente, hier als **S**ystem**C**omponent (SC) bezeichnet, hauptsächlich aus Unterkomponenten, Parametern und drei unterschiedlichen Berechnungsarten für diese (siehe Kapitel 2.2 und Abbildung 8). Die Parameter können für andere Komponenten nutzbar gemacht werden. Sie beschreiben somit die Ein- und Ausgänge der Komponente. Zur Zeichnung dieser notwendigen Elemente wurde, wie in Kapitel 3.6 beschrieben, das Datenmodell des VirSats nach ihnen gefiltert. Zur Erinnerung aus Kapitel 3.6 können Parameter wie folgt unterschieden werden: extern oder intern genutzte, ungenutzte oder für andere Komponenten zugängliche Parameter. Die intern genutzten und ungenutzten Parameter sind nur für das IBD erforderlich.
Um die relevanten Elemente auszufiltern wurde der in der Vorarbeit bestehende Suchalgorithmus abgeändert und verbessert. Zusätzlich wurde eine Hilfsklasse in die Software integriert, welche das Duplizieren von Quellcode verhindert und mehrfach benötigte Methoden beinhaltet. Dieser Algorithmus läuft anhand dieses Schemas ab:

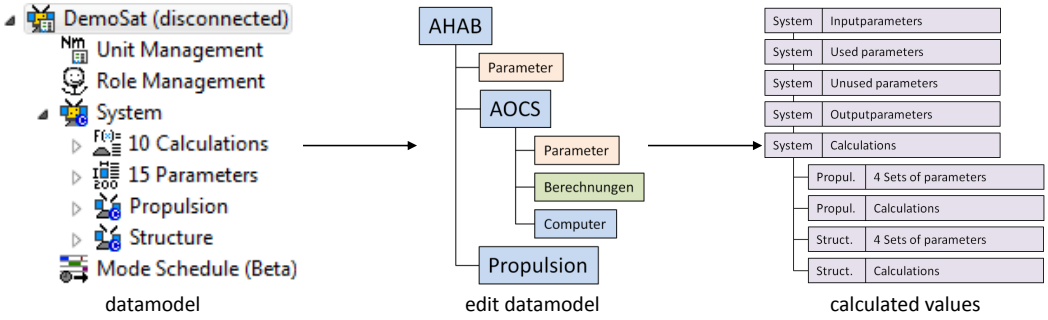

Abbildung 21: Auslesen der benötigten Elemente

Zunächst werden zu den gesuchten Parametergruppen Maps initialisiert, welche als Schlüssel die Komponente besitzen, in der sie sich befinden und als Wert ein Set von Parametern. Hierbei wurde ein Set und keine Liste gewählt, da da eine Set-Datenstruktur keine Duplikate enthalten kann. Dies ist wichtig, damit später kein Parameter doppelt gespeichert und somit auch doppelt gezeichnet wird. Danach werden

die Berechnungen der zu betrachtenden Komponente ausgelesen (siehe Abbildung 21, *datamodel* und *edit datamodel*). Alle Eingangsparameter der jeweiligen Berechnung werden dann in die Map-Datenstruktur der *Input Parameter* mit der dazugehörigen Berechnung abgespeichert (siehe Abbildung 21 *Sets*). Zu den Input Parametern wird zusätzlich das Herkunftsobjekt ermittelt. Dieser kann eine externe Komponente, eine andere Berechnung, eine interne oder die eigene Komponente sein. Auch dieses Wertepaar wird in der jeweiligen Map abgelegt. Zusätzlich werden auch die Output Parameter der Berechnung in die Map der *Output Parameter* abgelegt. Alle internen Parameter, welche für andere Komponenten zugänglich gemacht wurden, befinden sich in dem Set der *Shared Parameters*. In diesem Fall ist das Speicherobjekt ein Set und keine Map, da alle Parameter aus einer Komponente stammen. Anhand dieses Algorithmus werden alle Berechnungen nach den gewünschten Werten durchsucht. Nach der Durchsuchung der Berechnungen werden all jene Parameter der Komponente, welche noch in keinem Objekt abgespeichert wurden, automatisch in das Set der ungenutzten Parameter abgelegt. Nach diesem Durchlauf sind alle relevanten Elemente ermittelt und das Diagramm wird erstellt. Überprüft wird die Benutzbarkeit der Implementierung mit Hilfe eines JUnit-Tests.
Im Kapitel 4.1 wird die Anordnung der Elemente aus den oben genannten Maps im Diagramm definiert. Dabei wird in den Abschnitten 4.1.1 und 4.1.2 näher auf den Entwurf der Diagramme in Anlehnung an BDD und IBD eingegangen. Der Abschnitt 4.1.3 beschreibt die neuen Anforderungen an die Erstellung von Diagrammen im VirSat nach einer Präsentation in der CEF. Die Kapitel 4.2 und 4.3 charakterisieren die Modifikation des Erstellens der Diagramme durch die Änderungen in dieser Arbeit und die bidirektionale Kopplung des grafischen Editors mit dem Datenmodell des VirSat. Im Kapitel 4.3.1 wird genauer auf die stetige Aktualisierung der Diagramme an die Änderungen im Datenmodell eingegangen. Das Kapitel 4.3.2 beschreibt die Rückführung der Änderungen im grafischen Editor in das Datenmodell. Das Kapitel 4.4 beschäftigt sich mit der Integration des Layoutframeworks *KIELER* in den VirSat und den verwendeten Layoutalgorithmen.

4.1 Entwurf des Layouts von Diagrammen

Die einzelnen Diagrammarten müssen auf die im Datenmodell des VirSats befindlichen Informationen angepasst werden. Hierbei musste beachtet werden, dass die Diagramme nicht zu komplex werden, aber dennoch alle notwendigen Informationen

beinhaltet. Weiterhin musste die Anordnung der Elemente im Diagramm festgelegt werden. Hierzu wurde ein Layout in Abstimmung mit den Raumfahrtingenieuren der CEF entwickelt, welches in den folgenden Kapiteln erklärt wird.

4.1.1 Entwurf des Blockdefinitionsdiagrams

Wie in Kapitel 3.5.1 beschrieben ist die Hauptaufgabe der BDD die Darstellung der Schnittstelleninformationen einer Komponente, ohne den inneren Aufbau zu betrachten. Im VirSat bilden die Ein- und Ausgangsparameter einer Komponente die Schnittstellen.
Im SysML-Standard werden Schnittstellen als Ports bezeichnet. Ihre Darstellung ist in Abbildung 22 abgebildet. In Bezug mit dem Datenmodell des VirSats handelt es sich bei den Ein- und Ausgängen um Objektflussports (Flow Port), da diese von einer anderen Komponente stammen und in dieser Komponente verwendet werden. Als Beispiel kann man sich die Arbeit des Konfigurationsingenieurs vorstellen, bei der ein Ingenieur beispielsweise die die Definitionen der Größe anderer Komponenten benötigt um die Positionen der einzelnen Elemente ideal zu berechnen. Im Design grafischer Elemente im europäischen Raum ist es auf Grund der rechtsläufigen Schrift gängig, dass die Eingänge auf der linken und die Ausgänge auf der rechten Seite eines Elements im Diagramm dargestellt werden. Pfeile, die zu einer Komponente hin oder von einer Komponente weg dargestellt sind, kennzeichnen weiterhin die Richtung des Parameterflusses.

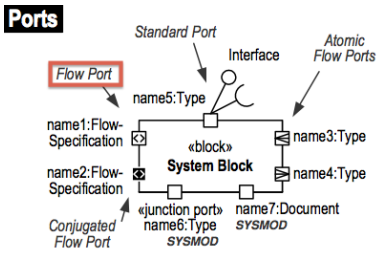

Abbildung 22: Schnittstellendarstellung nach dem SysML-Standard [Wei06]

Anhand dieser Spezifikationen wurde ein eigenes Layout für die Diagramme im VirSat entwickelt. Es soll an die Abbildung 23 angepasst sein. In dieser Grafik sieht man die Eingangs- und den Ausgangsparameter der Komponente Struktur. Diese sind auf der linken bzw. auf der rechten Seite der Komponente angeordnet. Zusätzlich zu den Parameternamen sollen im fertigen Diagramm der Wert und die Einheit des Parameters

und bei den Eingängen die Ursprungskomponente dargestellt werden, in diesem Fall das System.

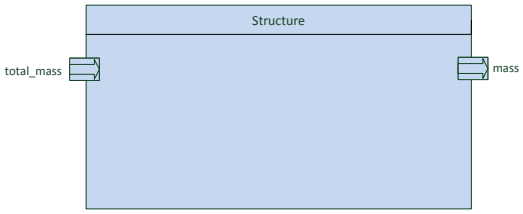

Abbildung 23: Entwurf des Blockdefinitionsdiagramms

Das gewünschte Diagramm für die Schnittstellendarstellung im VirSat beinhaltet Elemente aus dem Standard BDD. Es orientiert sich demnach stark an SysML.

4.1.2 Entwurf des Internen Blockdiagramms

Der innere Aufbau von Komponenten wird im **I**nternen **B**lock**d**iagramm (siehe Kapitel 3.5.2) grafisch dargestellt. Hierbei wird besonders auf die interne Struktur und Funktionsweise Wert gelegt. Eine Komponente besitzt im Datenmodell des VirSats die Berechnungen und interne Parameter als innere Struktur (Erinnerung an Abbildung 8). Dabei muss darauf geachtet werden, dass einzelne Berechnungen nach einer bestimmten Reihenfolge abgearbeitet werden müssen. Nimmt man als Beispiel wieder die Aufgaben des Konfigurationsingenieurs, welcher die Platzierung der Komponenten erst dann festlegen kann, wenn alle anderen Ingenieure die Informationen zu ihrer Komponente festgelegt haben. Andernfalls kommt es zu Fehlern in den Berechnungen.
Zur besseren Verständlichkeit des Aufbaus einer Komponente wurde ein Diagramm in Anlehnung an das IBD entworfen. Dieses spiegelt das BDD inklusive Spezifikationen des inneren Aufbaus wieder. Als Grundgerüst für den Entwurf wurde hierbei das BDD aus Abbildung 23 verwendet. Hierbei musste eine möglichst intuitive Art entwickelt werden die Berechnungen und intern genutzten Parameter effizient anzuordnen. Dieses IBD beschreibt die Berechnung der Massen in Struktur. Die Eigenmasse berechnet sich hierbei aus der Gesamtmasse multipliziert mit *percentageOfTotalMass* (0,15) und die Eigenmasse inklusive dem Unsicherheitsfaktor (massWithMargin) ist die Addition von sich selbst mit der Eigenmasse inklusive des Unsicherheitsfaktors des darunterliegenden Equipments. Der Entwurfs besitzt die Struktur aus Abbildung

24.

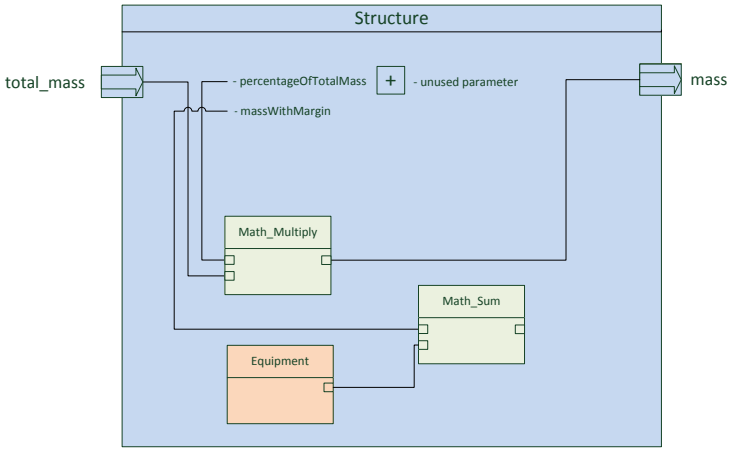

Abbildung 24: Darstellung der internen Struktur nach dem SysML-Standard

Die intern genutzten Parameter werden dabei in einer Auflistung in die linke obere Hälfte der Komponente platziert (siehe *percentageOfTotalMass* und *massWithMargin*). Von ihnen gehen Verbindungslinien zu den Berechnungen ab, in denen sie als Eingangsparameter verwendet werden. Die Verbindungslinien dienen zur besseren Verdeutlichung der Zusammenhänger der Elemente. Die Eingangsparameter von Berechnungen können nicht nur als der Komponente selbst, sondern auch aus weiteren Berechnungen oder Komponenten stammen. Auch diese Zusammenhänge werden mittels Verbindungslinien dargestellt (Vergleich Abbildung 24 *total_mass* und *Math_-Division*). In der rechten oberen Hälfte der Komponente werden die ungenutzten Parameter der Komponente grafisch dargestellt. Diese sollen mittels Klick auf das *Plus* angezeigt werden. Der Anwender behält hierbei die Entscheidung, ob er die ungenutzten Parameter anzeigen lassen möchte, was eventuell zu Unübersichtlichkeit führen kann, oder nicht. Die Farbgebung der einzelnen Elemente war teils willkürlich, aber nach harmonischen Farben gewählt. Die Farbe grün der Berechnungen bezieht sich auf die Standardfarbe grün bei der Applikation Microsoft Excel.
Dieser Entwurf ist stark an den Entwurf der Vorarbeit angepasst (Vergleich mit Kapitel 3.6). Jedoch beinhaltet dieser die Subkomponenten als weitere grafische Darstellung. Im direkten Vergleich mit dem IBD aus Abbildung 18 wird deutlich, dass es sich im Entwurf tatsächlich nur um eine Annäherung an den Standard der IBD handelt. Im Gegensatz zum SysML-Standard werden in diesem Diagramm keine Notationen im Diagramm verwendet. Die Wiederverwendung von Ports und das Kreieren von Verbindungslinien im Diagramm spiegelt jedoch die SysML-Anlehnung wider.

4.1.3 Anpassung der Diagramme an die Wünsche der CEF-Mitglieder

Damit die erstellten grafischen Editoren auch in der CEF genutzt werden, wurden sie am 30.07.2013 vor dem Grundteam der CEF, inklusive des Projektleiters, vorgestellt. In dieser Live-Demonstration wurde nicht nur der Aufbau und die Struktur präsentiert, sondern auch alle bisher eingebauten und noch zu erstellenden Funktionalitäten vorgestellt. Dazu gehören der Tooltip, die Festlegung der Einfärbung, das Erstellen, Editieren oder Löschen von Elementen im Datenmodell anhand des grafischen Editors und die Anwendung von Layoutalgorithmen im Diagramm. Während und nach der Präsentation konnten die Raumfahrtingenieure ihre Gedanken zu den grafischen Editoren erläutern. Insgesamt waren sie sehr begeistert von der Struktur und Benutzerfreundlichkeit der Editoren. Insbesondere die bidirektionale Kopplung des Editors mit dem Datenmodell des VirSats empfanden sie als großen Vorteil, da sie, wie in Kapitel 2.1 und Abbildung 5 beschrieben, den Parameters-Tabulator des VirSats aufgrund der vielen Scrollrichtungen ungern nutzen. Das geplante Feature der Darstellung von ingenieurstechnischen Besonderheiten im Diagramm empfanden die Raumfahrtingenieure als überflüssig und nicht notwendig.
Jedoch wünschten sie sich zusätzlich zu den bisherigen Diagrammtypen, dem BDD und IBD, noch eine weitere Diagrammart im Editor - das **F**unktions**b**lock-**D**iagramm (FBD). Der Grund dafür war, dass der Systemingenieur/Projektleiter anhand eines Diagramms die Beziehungen zwischen den einzelnen Komponenten erkennen kann. Das FBD charakterisiert sich dadurch, dass es die Funktionsweise und Abhängigkeiten in einer Komponente darstellt. Dieser Diagrammtyp gehört nicht zu dem SysML-Standard und wird meist in der Anlagentechnik oder in der Softwareentwicklung verwendet. [Per94]
Ein Beispiel eines solchen FBD ist in Abbildung 25 zu sehen. Diese Grafik beschreibt den funktionalen Aufbau in einer Kamera. In ihr ist definiert, welche Einstellungen bei bestimmten äußeren Umständen, beispielsweise der Änderung der Lichtintensität, gesetzt werden. Intern werden daraufhin zum Beispiel der Fokus der Kamera an die Umgebung angepasst und weitere Änderungen in den technischen Einstellungen vollzogen.
Ein solcher Diagrammtyp soll anstatt der Darstellung von ingenieurstechnischen Besonderheiten in den VirSat integriert werden. Dafür muss zunächst ein Entwurf erstellt werden. In Absprache mit den Raumfahrtingenieuren der CEF wurden folgende Anforderungen an das Diagramm definiert:

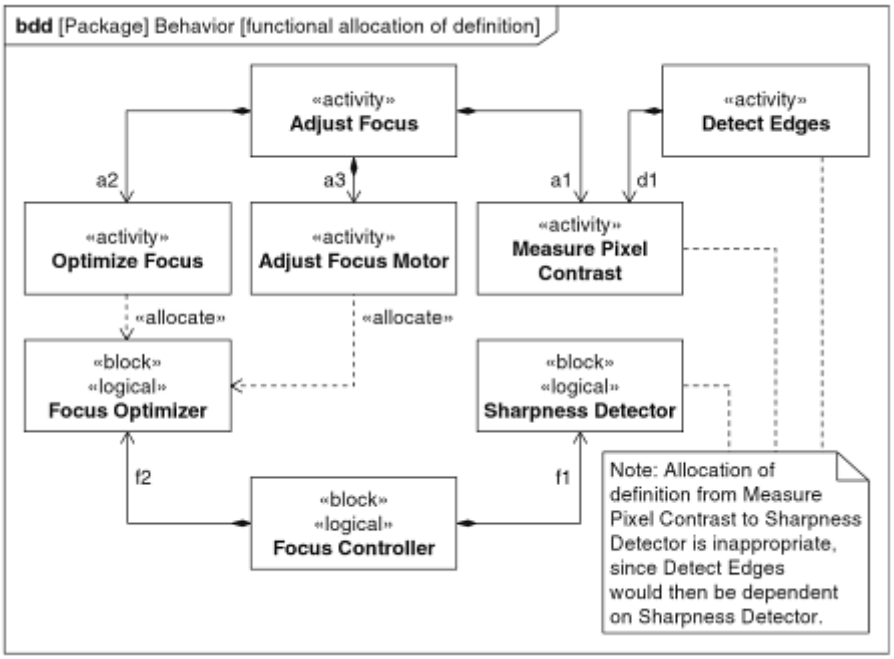

Abbildung 25: Functional Block Diagram [FMS11]

- Erstellung des FBD auf Ebene der StudyRepositories (siehe Abbildung 8)
- Verdeutlichung der Parameterbeziehungen der Komponenten inklusive Subkomponenten
- Benennung der Parameternamen
- gute Anordnung
- Bidirektionalität muss nicht vorhanden sein

Der Entwurf eines solchen Diagramms ist in Abbildung 26 zu sehen. Die einzelnen Subkomponenten (*Orbit*, *Structure* und *Propulsion* der Beispielstudie aus Abbildung 26) können weitere Subkomponenten in sich beinhalten, welche jedoch in diesem Diagramm nicht angezeigt werden. Dieses Beispiel der FBD verdeutlicht die Zusammenhänge der Hauptkomponente (*DemoSat*) in Bezug auf die Raketengleichung von Ziolkowski (siehe Kapitel 3.1).
Im direkten Vergleich der Diagramme 25 und 26 wird deutlich, dass es sich hierbei nur um eine Anlehnung an die Spezifikationen des FBD handelt. Die Notationen und Beschreibung der Blöcke sind in diesem Entwurf weggelassen bzw. für die Ingenieure auch ohne Vorkenntnisse von UML2.0 lesbar gestaltet.

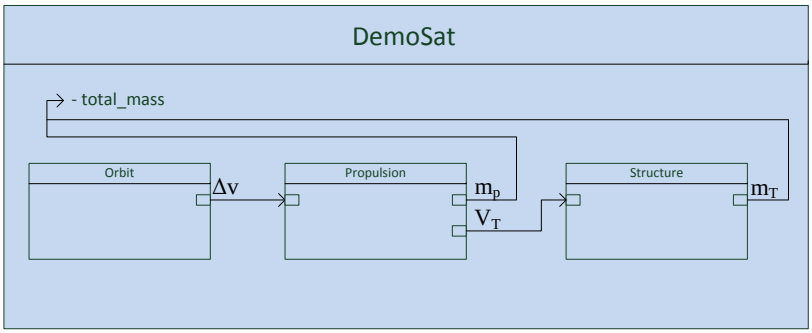

Abbildung 26: Darstellung der funktionalen Beziehungen nach dem Funktionsblock-Diagramm

Der Rechenalgorithmus zur Erkennung aller relevanten Elemente für diesen Diagrammtyp ist wie folgt aufgebaut: Zunächst wird die Hauptkomponente der Studie ermittelt. Anhand dieser werden die Subkomponente aus dem direkten Level unter der Hauptkomponente ausgelesen. Vergleicht man den Aufbau beispielsweise mit Abbildung 27, so spiegelt die Hauptkomponente das System wieder. Die Subkomponenten der darunterlegenden Ebene sind Propulsion und Structure. Bei der Andendung der

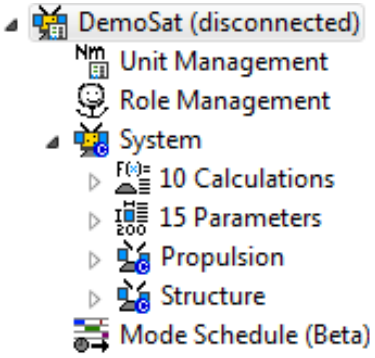

Abbildung 27: Studienansicht im Virtuellen Satelliten

Suchalgorithmen auf die Hauptkomponente werden, wie in Kapitel 4, zunächst alle Berechnungen nach ihren Inputwerten gefiltert. Stammt einer der Eingangswerte aus einer anderen, nicht darunterliegenden Subkomponente, so wird er in der dafür vorgesehenen Map der InputParameter mit dem Schlüssel der gerade auszulesenden Komponente abgespeichert. Vergleichen kann man diesen Sachverhalt damit, dass beispielsweise in der Komponente Propulsion ein Parameter aus Structure für eine Berechnung genutzt wird. Nach dem vollständigen Auslesen der Berechnungen werden alle Ursprungskomponenten der Inputwerte geortet und wieder in einer Map abgelegt. Dieser Vorgang wiederholt sich auch für alle darunterliegenden Subkompo-

nenten, wobei als Schlüssel in der Map der Inputwerte immer die oberste Subkomponente genutzt wird. Die darunterliegenden Subkomponenten sind alle Komponenten unter Propulsion oder Structure.

4.2 Erstellung von Diagrammen

Anhand der Suchalgorithmen aus Kapitel 4.1 und Unterkapitel 4.1.3 ist die Erstellung der Diagrammtypen im VirSat möglich. Der abstrakte Ablauf der Entwicklung der Diagramme aus dem Unterkapitel 4.1 im VirSat ist in Abbildung 28 zu sehen. Auf eine

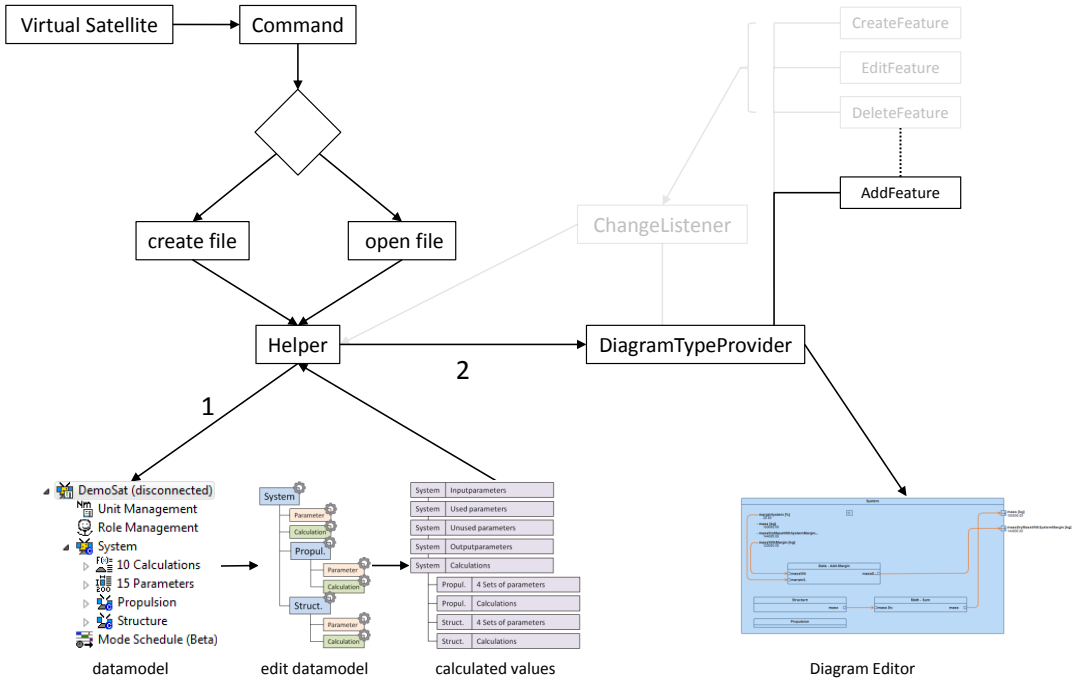

Abbildung 28: Struktur der Anbindung des Graphiti-Frameworks (Diagramm erstellen)

Benutzerinteraktion im Kontextmenü hin wird ein Command ausgeführt, welcher zunächst die Datei für das Diagramm erstellt. Hierzu wurde allein für die Diagramme ein Projekt im Hintergrund erstellt, an welches die Diagramme angeheftet werden. Ist das gewünschte Diagramm bereits vorhanden, so wird es nur geöffnet. In der Hilfsklasse findet das Auslesen des Datenmodells nach den gesuchten Elementen und die

darauffolgende Zeichnung in das Diagramm statt. Das Auslesen erfolgt Diagrammtyp-spezifisch nach den bereits beschriebenen Rechenalgorithmen. Nach dessen Durchlauf wird ein für den Diagrammtypen entworfener Diagram Type Agent (siehe Kapitel 3.3.1) an das Diagramm angeheftet. In ihm sind die Features, wie beispielsweise das Hinzufügen der Elemente zum Diagramm, definiert. Nach Beendigung der Zeichnung erscheint das Diagramm automatisiert im Editor.
Die Diagrammtypen **B**lock**d**efinitions**d**iagramm und **I**nternes **B**lock**d**iagramm sind im VirSat nicht allein durch ihr Aussehen und ihre innere Struktur aneinander gekoppelt, sondern auch in ihrer Entstehung. Das IBD wird, im Gegensatz zum BDD und FBD, nicht mit Hilfe des Kontextmenüs erstellt, sondern aufgrund einer Benutzerinteraktion im BDD - dem Doppelklick auf die angezeigte Komponente. Dies ist die gleiche Vorgehensweise wie in der Vorarbeit, jedoch erscheinen die Diagramme in unterschiedlichen Dateien. Der Grund dafür war das Erscheinen von Fehlermeldungen bei Modifikationen im grafischen Editor, wie im Unterkapitel 3.6 beschreiben. Um dem zu entgehen wurden die Diagramme voneinander getrennt.
Zusätzlich wurde in dieser Arbeit das Zeichnen von Elementen im Diagramm abgeändert. Vor dieser Arbeit wurde das Diagramm an sich als Hauptbezugspunkt zu jedem Element gewählt. Dies hatte zur Folge, dass beispielsweise beim Verschieben einer Berechnung die Parameter in ihm an gleicher Position wie davor blieben. Nun standen die Parameter nun außerhalb der dazugehörigen Berechnung und konnten vom Benutzer nicht zugewiesen werden. Um dem zu entgehen wird nun immer das direkt darüber liegende Element als Bezugspunkt genutzt. Vergleicht man diese Vorgehensweise mit Abbildung 24, so werden bei der Verschiebung der Berechnung 1 automatisch auch alle in ihm befindlichen Parameter mit verschoben.
Die Parameterdefinitionen der Masse und der Leistung sind für die Ingenieure besonders interessant, da sie in den meisten Komponenten benötigt/zusammengesetzt werden. Zum besseren Verständnis der IBD, insbesondere bei einer Vielzahl an Berechnungen und Parametern, wurde ein Filter in den grafischen Editor integriert. Mit ihm ist die Darstellung von allen Berechnungen, in denen beispielsweise die Masse berechnet wird oder für eine Berechnung genutzt wird, dargestellt. Alle anderen Elemente werden hierbei aus dem Diagramm ausgesondert.
Das im VirSat erstellte BDD ist in Abbildung 29 zu sehen. Dies beschreibt die allgemeine Massengleichung der Struktur bei keinen weiteren gegebenen Spezifikationen außer der Gesamtmasse. Dies ähnelt stark dem Entwurf der BDD aus Abbildung 23.
Das IBD, erstellt im VirSat besitzt als Beispiel dieses Aussehen:

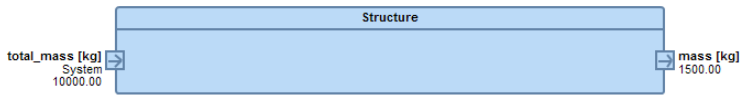

Abbildung 29: Schnittstellendarstellung der allgemeinen Massengleichung im Virtuellen Satelliten

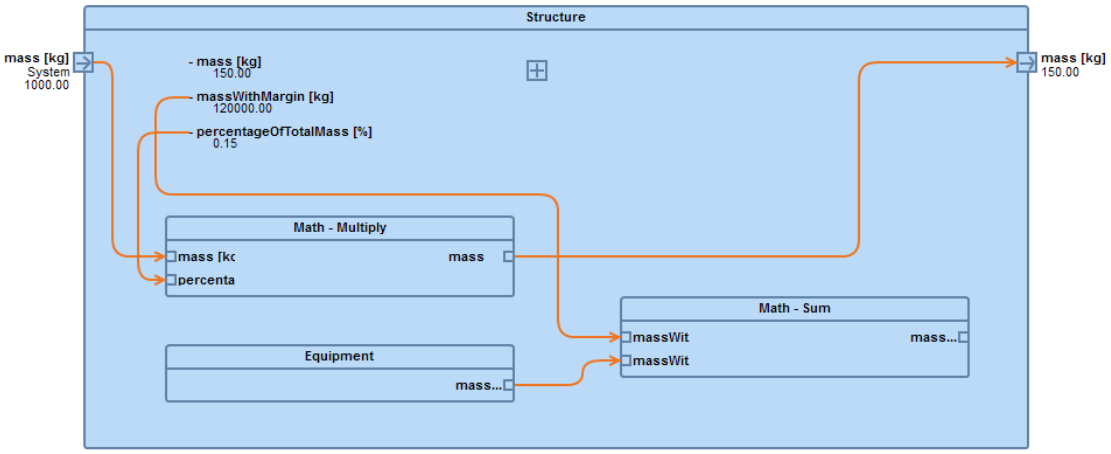

Abbildung 30: Innere Zusammensetzung der Komponente Struktur im Virtuellen Satelliten

Abbildung 30 zeigt das IBD der Komponente *Struktur* mit der internen Berechnung der Masse und der Masse inklusive Hülle. Hieraus wird deutlich, dass die interne Masse etwa 15 Prozent der Gesamtmasse besitzt. Die Zusammenhänge der Parameter mit den Berechnungen sind mit Hilfe der Verbindungslinie dargestellt. Die Pfeilspitze zeigt die Richtung an. Zusätzlich wurde ein dritter Diagrammtyp, das FBD in die Software integriert und ist in Abbildung 31 zu sehen. In dieser Abbildung ist die Gleichung von

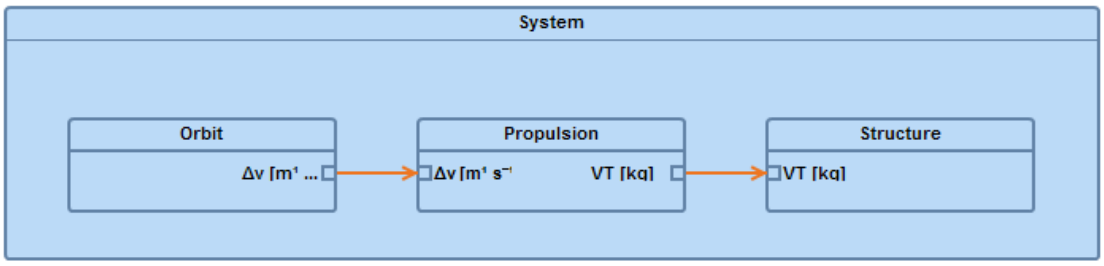

Abbildung 31: Funktionale Beziehungen der Gleichung von Ziolkowski im Virtuellen Satelliten

Ziolkowski abgebildet. Diese besagt, dass sich die Totale Masse der Hauptkomponente aus den Massen des Antriebs und der Struktur zusammensetzt (siehe Kapitel 3.1). In diesem Diagramm werden die Zusammenhänge aller Subkomponenten veranschaulicht. Der Datenfluss wird hierbei mit Hilfe der Verbindungslinie dargestellt.

Zusätzlich wurden einige benutzerfreundliche Features in die Diagramme integriert. Als Beispiele sind das Vergrößern-/Verkleinern-Feature und der ToolTip zu nennen. Dieser wurde im grafischen Editor so abgeändert, dass er bei Parametern immer den Namen, die Ursprungskomponente, den aktuellen Wert und die Einheit anzeigt und bei Komponenten und Berechnungen immer deren Namen darstellt.

4.3 Bidirektionale Kopplung

In der Software VirSat wird nach dem Prinzip verfahren, dass sich alle abhängigen Editoren bei Änderungen im Datenmodell automatisch aktualisieren. Stellt man sich beispielsweise die Zuweisung der Masse der Struktur im Satelliten vor. Würden sich nicht automatisch alle abhängigen Editoren und Berechnungen mit dem neuen Wert aktualisieren, so würde es zu Inkonsistenz und damit verbunden zu einer falschen Berechnung der Gesamtmasse und zur Gefährdung der Mission kommen.
Genau dies ist bei den bisherigen Diagrammen der Fall. Sie spiegeln das Datenmodell zum Zeitpunkt der Erstellung in korrekter Form wieder, aktualisieren sich jedoch nicht bei Änderungen im Datenmodell. Dafür muss das Diagramm bisher stets von den Raumfahrtingenieuren mittels einer Aktion neu gezeichnet werden, was schnell zu Unlust und Irritationen führt.
Andersherum werden bisher auch alle Änderungen im grafischen Editor vom Datenmodell des VirSats ignoriert. Erstellt oder modifiziert man beispielsweise die Masse der Struktur im grafischen Editor, so werden die hier definierten Informationen nicht in die anderen Editoren übertragen und stets bei der Aktualisierung anhand des Datenmodells verworfen.
Um eine benutzerfreundliche Anwendung zu schaffen müssen beide Probleme gelöst und in den VirSat integriert werden.

4.3.1 Automatisierte Aktualisierung

Damit der grafische Editor genutzt werden kann, muss er immer die aktuellen Informationen des Datenmodells anzeigen. Das Graphiti-Framework bietet hierzu das Update-Feature, welches auf Änderungen im Datenmodell automatisch reagieren soll. Es vergleicht ständig das zugrunde liegende Datenmodell mit dem Modell des Diagramms. Bei Veränderungen wird ein Zeichen im Diagramm gesetzt, sodass dieses

aktualisiert werden muss. Diese Funktionalität gelingt jedoch nicht korrekt bei eigenen Datentypen, wie denen im VirSat. Somit konnte das Feature alleinstehend nicht zur Aktualisierung der Diagramme genutzt werden.
Im **Vir**tuellen **Sat**ellit wird die Aktualität aller Editoren mittels des Beobachter-Patterns sicher gestellt. Bei diesem Entwurfsmuster in der Softwareentwicklung werden alle registrierten Objekte eines beobachteten Objektes informiert, sobald eine Änderung erfolgt. Im Falle des VirSats bilden alle geöffneten Editoren inklusive der StudyNavigator-Ansicht(siehe Abbildung 3 Part 1) die beobachteten bzw. beobachtenden Objekte. Wird, wie in Abbildung 32 dargestellt, ein Editor modifiziert, so werden automatisch

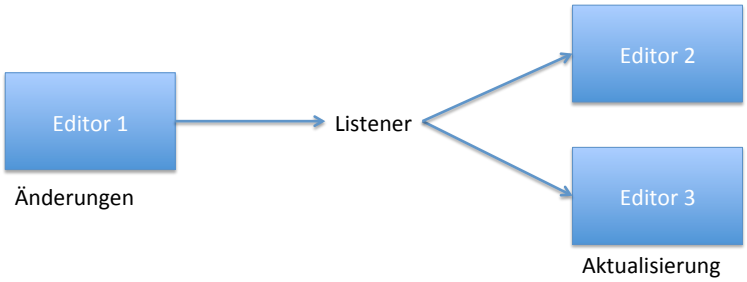

Abbildung 32: Observer-Pattern im Virtuellen Satelliten

alle weiteren registrierten Editoren aktualisiert. Die Registrierung des Listeners erfolgt bereits bei der Initialisierung der Editoren.
Um Sicherzustellen, dass der grafische Editor auch auf alle Änderungen im Datenmodell reagiert, wurde auch für ihn ein Listener erzeugt und den Beobachtern hinzugefügt. Die Struktur der Aktualisierung bei Aufruf der Beobachter ist in Abbildung 33 zu sehen. Dabei werden zunächst mittels des RemoveFeatures alle Elemente aus dem Diagramm, aber nicht aus dem Datenmodell, gelöscht. Danach wird die dazugehörige Hilfsklasse erneut aufgerufen, das Datenmodell des VirSats nach den notwendigen Elementen gefiltert und neu gezeichnet. Die komplette Neuzeichnung ist aufgrund der Anordnung der Elemente notwendig, da es andernfalls zu beispielsweise Überlagerungen im Diagramm käme.
Zusätzlich zu dem Zeichnen- und UpdateFeature wurden noch weitere Features in die Diagramme integriert. Als Beispiele sind das Vergrößern-/Verkleinern-Feature und der ToolTip zu nennen. Dieser wurde im grafischen Editor so abgeändert, dass er bei Parametern immer den Namen, die Ursprungskomponente, den aktuellen Wert und die Einheit anzeigt und bei Komponenten und Berechnungen immer deren Namen darstellt.
Weiterhin muss bei der Neuzeichnung darauf geachtet werden, ob sich der grafische

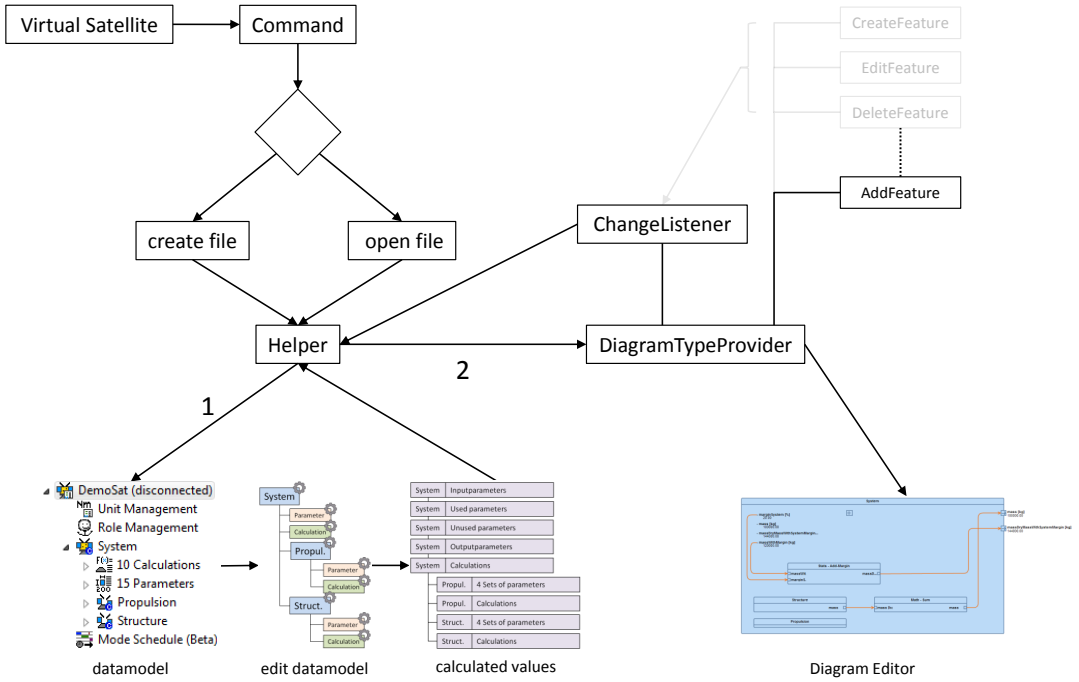

Abbildung 33: Struktur der Anbindung des Graphiti-Frameworks (automatische Aktualisierung)

Editor bereits vorher in der aktuellen Ansicht befand oder nicht. Nach der Aktualisierung soll wieder der gleiche Editor in der Ansicht sein wie vorher.

4.3.2 Rückführung der Modifikationen im Editor in das Datenmodell

Die Raumfahrtingenieure sollen den grafischen Editor nicht allein als Zeichenhilfsmittel nutzen, sondern er soll ihnen auch die Arbeit während einer Raumfahrtstudie vereinfachen. Hierfür soll er auf Benutzerinteraktionen reagieren. Die tabellarischen Editoren des VirSats bieten die Möglichkeiten des Erstellens, Editierens und Löschens von Parametern und Berechnungen. Für die Erstellung und Modifikation wurden Dialoge entwickelt, in welchen der Benutzer die erarbeiteten Information eintragen kann. Den Dialog zum Erstellen/Editieren eines Parameters ist in Abbildung 34 zu sehen. In ihm kann der Nutzer den Namen, den Wert, die Einheit usw. eintragen. Nach Beendigung der Dialogeingabe wird der Parameter mit den definierten Einstellungen in

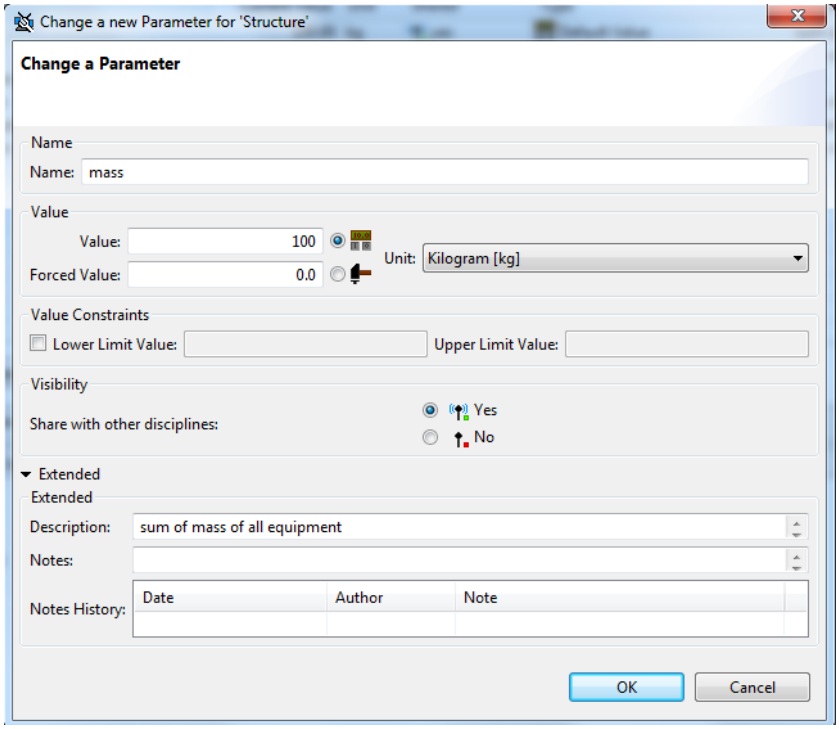

Abbildung 34: Dialog zur Erstellung/Modifikation eines Parameters

den tabellarischen Editoren erstellt oder abgeändert. Die Interaktion des Erstellens und Editierens von Parametern und Berechnungen sollen im grafischen Editor auch anwendbar sein. Dafür sollen die bereits bestehenden Dialoge wiederverwendet werden. Dabei soll das im **Vir**tuellen **Sat**elliten befindliche und Rechte vergebende Rollenmanagement mit beachtet werden.
Der VirSat arbeitet nach dem Verfahren, dass jeder Ingenieur alles sehen, aber nur die für ihn definierte Komponente editieren kann. Somit wird gewährleistet, dass jeder Ingenieur nur Änderungen an den ihm zugewiesenen Komponenten ausführen kann (siehe Kapitel 2.1 und 2.2).
Die Nutzeränderungen sollen jedoch allein im **I**nternen **B**lock**d**iagramm anwendbar sein, da nur in diesem Diagrammtyp die interne Struktur einer Komponente angezeigt wird. Um neue Elemente in das hinter dem Diagramm befindliche Datenmodell zu integrieren bietet das Graphti-Framework das *CreateFeature*, welches im FeatureProvider festgelegt wird. Ein FeatureProvider kann mehrere CreateFeatures von unterschiedlichen Datentypen beinhalten. Der Nutzer sieht diese dann in der Palette des grafischen Editors.
Bei der Erstellung eines neuen Elements, beispielsweise Parameters, wird der dafür im VirSat befindliche Dialog aufgerufen (siehe Abbildung 34). Nach dem Erstellen

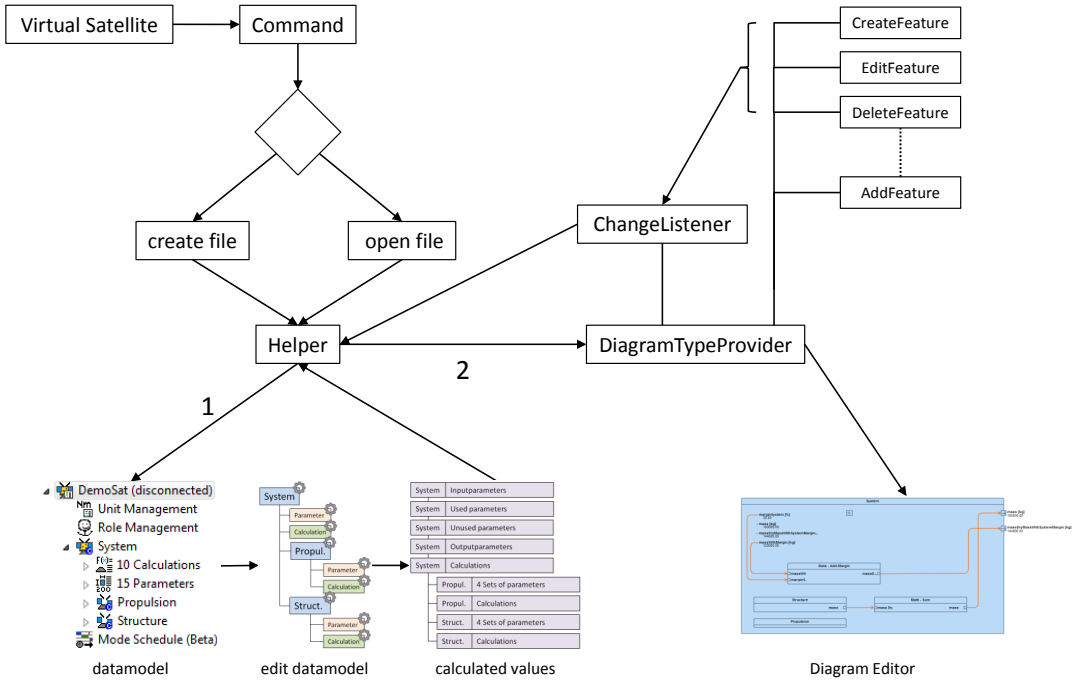

Abbildung 35: Struktur der Anbindung des Graphiti-Frameworks (Änderungen im grafischen Editor)

wird eine Aktualisierung an alle registrierten Listener gesendet und das Observer-Pattern abgearbeitet (siehe Unterkapitel 4.3.1). Das Gleiche geschieht bei der Erstellung einer neuen Berechnung oder Unterkomponente. Ist das neu kreierte Element eine Berechnung, so wird automatisch ihre Korrektheit überprüft. Wird ein Parameter beispielsweise als Ein- und Ausgang einer Berechnung genutzt, so erscheint automatisch eine für den Nutzer sichtbare Warnung. Je nach Definition wird der neu erstellte Parameter in die definierten Maps hinzugefügt und gezeichnet.
Das Graphiti-Framework zeichnet die neu erstellten Elemente automatisch immer an die Stelle, wohin sie vom Nutzer gezogen werden. Die CreateFeatures im VirSat sind allerdings so konzipiert, dass sie immer an der passenden Stelle im Diagramm angeordnet werden. So besitzt das Diagramm stets einen guten und übersichtlichen Aufbau.
Zusätzlich zu dem Erstellen von Elementen im Datenmodell und Diagramm wurde ein EditFeature integriert. Dieses kann mit Hilfe des Doppelklicks auf eine Komponente

oder per Kontextmenü aufgerufen werden. Bei ihrer Auswahl erscheint automatisch der dafür zuständige Dialog mit den bereits vordefinierten Einstellungen (siehe Abbildung 34). Nach Beendigung der Dialogeingabe werden die neu bestimmten Definitionen automatisch in das Datenmodell übertragen. Dies wirft wieder eine Aktualisierung an alle registrierten Listener, womit das Diagramm neu gezeichnet wird. Jedoch können nicht alle Elemente automatisch im Diagramm editiert werden. Grund hierfür ist, dass für die Veränderung einer Systemkomponente beispielsweise im VirSat der dazugehörige Editor geöffnet werden muss. In ihm kann der Nutzer dann seine Änderungen in den tabellarischen Editor eintragen. Das Gleiche trifft bei den Berechnungen in ingenieursüblicher Form und bei den Berechnungen in Anlehnung an Excel zu.
Beim Löschen von Elementen wird der Nutzer auf seine Zuständigkeit überprüft und nocheinmal abgefragt, ob das Element wirklich gelöscht werden soll. Bei Bejahung wird es nicht allein aus dem Diagramm, sondern auch aus dem Datenmodell, gelöscht und das Diagramm aufgrund der stetigen Aktualisierung neu gezeichnet.

4.4 Anordnung von Diagrammelementen durch Layoutalgorithmen

Bei der Neuzeichnung von Diagrammen wird die Anordnung der Elemente nur grob betrachtet. Hierbei wird nur darauf geachtete, dass sich keine Elemente überlappen dürfen. Die Anordnung an sich ist aber dadurch noch nicht ideal. Auf Überschneidungen der Verbindungslinien und ein intelligentes Layout wird noch nicht geachtet. Insbesondere bei komplexeren oder sehr schlecht angeordneten Diagrammen führt dies zu Unverständlichkeit (siehe Abbildung 36). In dieser Abbildung ist das

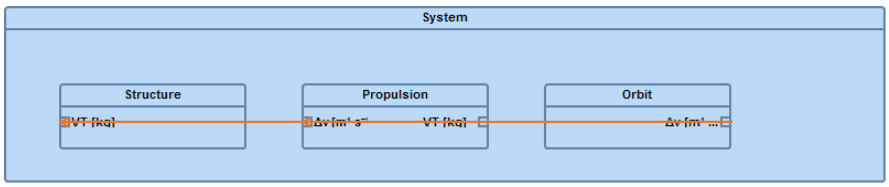

Abbildung 36: Beispiel eines IBD ohne Layouting

Beispiel des FBD aus der Abbildung 31 der Gleichung von Ziolkowski ohne Anwendung von Layoutalgorithmen dargestellt. Das Nutzen eines Diagramms in einer

solchen Form ist kaum möglich. Zur Verbesserung müssen Layoutalgorithmen integriert werden. Dafür gibt es mehrere Möglichkeiten. Die Erste ist das Implementieren eigener Layoutalgorithmen, da das Graphiti-Framework standardmäßig keine Algorithmen mitliefert. Dies ist jedoch sehr umständlich und komplexe Layoutframeworks wurden schon mehrfach von anderen Personen entwickelt. Somit wurde ein geeignetes Framework zur automatisierten Anordnung von Elementen gesucht, welches auf das Framework Graphiti anwendbar, in Eclipse integrierbar und zusätzlich noch leicht zu bedienen ist. Das passende Tool für diese Anforderungen wurde gefunden, das KIELER-Layoutframework. In Erinnerung an Kapitel 3.4 können die Nutzer hierbei mittels einer Layout View die gewünschten Layoutoptionen einstellen. Standardmäßig bringt das KIELER-Framework sechs verschiedene Layoutalgorithmen mit sich. Das populärste hierbei ist das KLay Layered, welches die Elemente nach diesem Aufbau anordnet: Es zeichnet die Verbindungslinien orthogonal zueinander und ordnet die

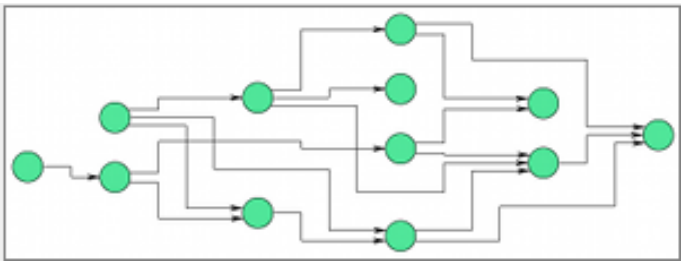

Abbildung 37: KLay Layered Anordnung (Quelle: Anwendung des KIELER-Frameworks in Eclipse)

Elemente in einer übersichtlichen Form an. Genau diese Anordnung ist in den IBD von den Mitgliedern der CEF gewünscht. Hierbei sollen allein die Berechnungen und Subkomponenten neu angeordnet werden. Alle Parameter sollen sich weiterhin an der gleichen Stelle befinden.
Jedoch ist das Framework standardmäßig nicht auf jede Art der Diagramme im Graphiti-Editor anwendbar. Der Grund dafür ist, dass jedes Diagramm einen anderen Aufbau besitzt. Damit das Layoutframework auch auf die eigene Diagrammart anwendbar ist, muss es darauf angepasst werden. Bei der Anpassung gab es langwierige Probleme, da das Framework in Bezug auf die Anpassung an eigene Diagrammmodelle nicht genügend dokumentiert ist. Für die Anpassung muss der in 3.4 beschriebene Diagram Layout Manager abgeändert werden. Die Komponenten, Subkomponenten und Berechnungen repräsentieren hierbei die KNodes und die Parameter die KPorts. Die Hauptkomponente bildet den Rahmen und die restlichen Elemente. Dabei ist darauf zu achten, dass alle Elemente als Containershape (siehe Kaptiel 3.4) im Diagramm

dargestellt sind. Andernfalls werden sie beim Layouting nicht beachtet. Zusätzlich müssen einige Layoutoptionen gesetzt werden, welche beispielsweise eine feste Position der Ports bestimmen. Nach den richtigen Einstellungen kann das Layouting ausgeführt werden und das Diagramm aus Abbildung 36 besitzt nun die Form der Abbildung 31.
Weiterhin soll im VirSat das Layouting direkt nach dem Erstellen der Diagramme ausgeführt werden. Dies geschieht mit Hilfe eines Aufrufs der Diagram Layout Engine. Im derzeitigen Stand ist das KIELER-Framework nicht mit der Eclipse Version 4.3 kompatibel, welches im VirSat genutzt wird. Hierfür musste die Software VirSat manuell

5 Fazit und Ausblick

Zusammenfassend wurde ein grafischer Editor zur Visualisierung von Informationen in die Software **Vir**tueller **Sat**ellit integriert. Im grafischen Editor können drei Diagrammtypen automatisch erstellt werden. Diese entsprechen den Spezifikationen des **B**lock**d**efinitions**d**iagramms, **I**nternen **B**lock**d**iagramms und **F**unktions**b**lock**d**iagramms und wurden an die Anforderungen der Raumfahrtingenieure angepasst.
Der Entwurf zu den Diagrammen BDD und IBD wurde bereits in der Vorarbeit zu dieser erarbeitet. Ihre Struktur nach der Erstellung im VirSat ist stark an die Entwürfe angepasst. Die Darstellung des BDD ist beispielsweise in Abbildung 29 zu sehen. In ihm wurden die Schnittstelleninformationen dargestellt. Im Diagramm in Anlehnung an das BDD im grafischen Editor sind keine Benutzerinteraktionen, bis auf den ToolTip möglich, da keine internen Informationen dargestellt werden.
Zusätzlich zu dem BDD wurde ein Diagramm in Anlehnung an das IBD in den grafischen Editor integriert. Mittels Doppelklick auf die Hauptkomponente im BDD wird dieser Diagrammtyp erstellt. Dieser beschreibt zusätzlich zu den Schnittstellen auch den inneren Aufbau einer Komponente. Als Beispiel ist Abbildung 30 zu sehen. In diesem grafischen Editor ist das Bearbeiten und Hinzufügen von Elementen mit Hilfe einer Palette möglich. Die dort vorhandenen Aktionen wurden selbst im Feature Provider erstellt. Bei ihrer Ausführung wird nicht nur das Diagramm, sondern auch das dahinterliegende Datenmodell modifiziert. Für die Erstellung und Editierung von Elementen werden die bereits bestehenden Dialoge für das jeweilige Element wiederverwendet. Im IBD gibt es weiterhin die Möglichkeit der Filterung nach Elementen. Dabei kann das Diagramm nach den beiden meist verwendesten Parameter *Masse* und *Leistung* dargestellt werden. Alle Parameter und Berechnungen, die nichts mit beiden zu tun haben werden hierbei nicht dargestellt.
Weiterhin wurde eine dritte Diagrammart - das FBD - in die Software VirSat anstatt der Darstellung von Elementen im ingenieurstechnischen Design entworfen und integriert. Grund hierfür war ein Gespräch mit den Mitgliedern der **C**oncurrent **E**ngineering **F**acility. Diese empfanden die ingenieurstechnische Darstellung als entbehrlich und äußerten den Wunsch nach einem FBD. Diese Diagrammart kann im VirSat nur auf der höchsten Ebene der Studie (dem Repository) ausgeführt werden. Es beinhaltet die Darstellung der funktionalen Beziehung aller Subkomponenten inklusive der weiter darunterliegenden Komponenten. Dieser Typ wurde auf die Anforderungen der Raumfahrtingenieure angepasst und in den VirSat eingefügt. Dabei werden jedoch

allein die Beziehungen zwischen den Subkomponenten betrachtet.
Darüber hinaus wurde in dieser Arbeit das KIELER-Layoutframework integriert und auf das Datenmodell des VirSats angepasst. Dieses ist für die verständliche Anordnung der Elemente und das korrekte Routing der Verbindungslinien zuständig. Dabei entstanden aufgrund der komplexen Struktur des Frameworks einige Probleme, welche jedoch mit Hilfe der Entwickler des Frameworks gelöst werden konnten. Zusammenfassend wurden die Aufgaben der Bachelorarbeit umgesetzt.
In einer weiterführenden Arbeit könnte das Highlighting von Elementen umgesetzt werden. Hierbei könnte man den Datenfluss eines Parameters von der Definition bis hin zur Ausgabe nachvollziehen. Zusätzlich könnte das Layouting besser in den grafischen Editor integriert werden, was jedoch in der kurzen Zeit nicht realisierbar war.
Ein weiterer Ausblick ist

Literatur

[AAC+13] Alhoff, M. ; Ayaz, B. ; Cetin, M. ; Kaltchev, L. ; Kopetzki, D. ; Kozianka, M. ; Lerche, L. ; Nagel, B. ; Schröder, C. ; Yaman, M. ; Zaki, Y.: *Eclipse4Bio*. http://hdl.handle.net/2003/29847, 2013. – Webzugriff: 2013-07-02

[BMJ12] Baxter, R. ; Mouat, A. ; Jackson, M.: JISC Final Report. (2012), S. 9–10

[C+11] Corporation, IBM u. a.: *Commands*. http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fextension-points%2Forg_eclipse_ui_commands.html, 2011

[Chr13] Christian-Albrechts-Universität zu Kiel, Institut für Informatik: *KIELER Wiki*. http://rtsys.informatik.uni-kiel.de/confluence/display/KIELER/Overview, 2013

[Com03] Computing, Bolour: *Notes on the Eclipse Plug-in Architecture*. http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html, 2003. – Webzugriff: 2013-08-14

[Cor12] Corp, PivotPoint T.: *SysML Frequently Asked Questions*. http://www.sysmlforum.com/faq/#Why_use_SysML, 2003 - 2012

[DIL+12] Dubois, H. ; Ibanez, V. ; Lopez, C. ; Machrouh, J. ; Meledo, N. ; Silva, A.: The product-line engineering approach in a model-driven process. In: *Proceedings of the 6th European Congress for Embedded Real Time Software and Systems Conference, ERTSC, Toulouse, France*, 2012

[Dun12] Dunn, M. J.: *Global Positioning Systems Directorate Systems Engineering & Integration Interface Specification IS-GPS-800*. http://www.gps.gov/technical/icwg/IS-GPS-200G.pdf, 2012

[Ebe11] Ebert, R.: *Eclipse RCP - Entwicklung von Desktop-Anwendungen mit der Eclipse Rich Client Platform*. 2011

[Ecl13] Eclipse Foundation: *eTrice*. http://www.eclipse.org/etrice/, 2013

[FMS11] *Kapitel* 7.1, A.4. In: Friedenthal, S. ; Moore, A. ; Steiner, R.: *A Practical Guide to SysML: The Systems Modeling Language*. Elsevier Science, 2011 (The MK/OMG Press). – ISBN 9780123852076

[Fou13] Foundation, Eclipse: *About the Eclipse Foundation*. http://www.eclipse.org/org/, 2013. – Webzugriff: 2013-08-14

[FSS11] In: Fortescue, P. ; Swinerd, G. ; Stark, J.: *Spacecraft Systems Engineering*. Wiley, 2011. – ISBN 9781119978367, S. 644 – 648

[Inf13] Informatik, OOSE I.: *Was ist Systems-Engineering?* http://www.oose.de/nuetzliches/fachliches/was-ist-systems-engineering/, 2013

[JRZ+04] Jecklet, M. ; Rupp, C. ; Zengler, B. ; Queins, S. ; Hahn, J.: UML 2.0 - Neue Möglichkeiten und alte Probleme. In: *Informatik-Spektrum* 27 (2004), 323-331. http://dx.doi.org/10.1007/s00287-004-0416-7. – DOI 10.1007/s00287–004–0416–7. – ISSN 0170–6012

[K. 08] K. Uttamang: Design of Inverted Pendulum System Using SysML. In: *No Magic Inc.* (2007 - 2008)

[Kle12] Klevtsova, E.: *Realisierung der konkreten Syntax von "DeepML" mit dem Graphiti-Framework*. https://www2.cs.fau.de/teaching/thesis/download/i2S00447.pdf, 2012

[Mül13] Müller, J.: Entwicklung eines graphischen Editors zur Darstellung der Schnittstellen im Satellitendesign / Praxisbericht, DHBW Mannheim, DLR Braunschweig. 2013. – Forschungsbericht

[Par07] Partain, Philip: Cloudsat MODIS-AUX auxiliary data process description and interface control document. In: *Fort Collins, Colorado, version* 5 (2007), S. 18

[PBBSA94] Pasian, F ; Benacchio, L ; Bonoli, C ; Serego Alighieri, S di: TNG Archives Interface Control Document. In: *Osservatorio Astronomico Padova-Asiago* (1994)

[PDLS13] Prof. Dr. Leymann, F. ; Schumm, D.: *Repository*. http://wirtschaftslexikon.gabler.de/Archiv/569801/repository-v3.html, 2013. – Webzugriff: 2013-08-12

[Per94] Perozzo, James: *The complete guide to electronics troubleshooting:*. Delmar Pub., 1994. – 72 S. http://books.google.de/books?id=rpfgAAAAMAAJ. – ISBN 9780827350458

[Rie12] Riess, M.: *KRendering*. http://rtsys.informatik.uni-kiel.de/teaching/11ws/s-ober/osem11ws-mri-talk.pdf, 2012. – Webzugriff: 2013-07-04

[SFL+10] Schaus, V. ; Fischer, P. M. ; Lüdtke, D. ; Braukhane, A. ; Romberg, O. ; Gerndt, A.: Concurrent engineering software development at german aerospace center-status and outlook. In: *4th International Workshop on System & Concurrent Engineering for Space Applications*, 2010

[SFQG12] Schaus, V. ; Fischer, P. M. ; Quantius, D. ; Gerndt, A.: Automated Sensitivity Analysis in Early Space Mission Design. (2012). http://elib.dlr.de/78109/. – Webzugriff: 2013-09-02

[The13] The Eclipse Foundation: *Graphiti*. http://127.0.0.1:56618/help/index.jsp, 2013

[Tie13] Tiede, M.: *Evaluierung von Suchalgorithmen zur formalen Verifikation von Raumfahrtmissionen, Ostfalia Hochschule für angewandte Wissenschaften, Deutschland*. 2013

[Wei06] Weilkiens, T.: *Systems engineering mit SysML/UML*. dpunkt-Verlag, 2006 http://books.google.de/books?id=py1dywAACAAJ. – ISBN 9783898644099

[Wel08] Welfle, A.: *Erweiterte Identifizierung, automatische Generierung und Analyse von konservierten Sequenzmustern und vergleichende Analyse enzymatischer Reaktionen unter Verwendung von homologen Enzymdomänen*, Universität zu Köln, Diss., 2008