17 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Modellgetriebene Validierung von System-Architekturen gegen architekturrelevante Anforderungen

Ein Ansatz zur Validierung mit Hilfe von Simulationen

André Pflüger

University of Bamberg Press

17 Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Schriften aus der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität Bamberg

Band 17

University of Bamberg Press
2014

Modellgetriebene Validierung von System-Architekturen gegen architekturrelevante Anforderungen

Ein Ansatz zur Validierung mit Hilfe von Simulationen

von André Pflüger

University
of Bamberg
Press
2014

Bibliographische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Informationen sind im Internet über http://dnb.ddb.de/ abrufbar

Diese Arbeit hat der Fakultät Wirtschaftsinformatik und Angewandte Informatik der Otto-Friedrich-Universität als Dissertation mit dem Titel „Modellgetriebene Validierung von System-Architekturen gegen architekturrelevante Anforderungen mit Hilfe von Simulationen“ vorgelegen.
1. Gutachter: Prof. Dr. Guido Wirtz, Universität Bamberg
2. Gutachter: Prof. Dr. Wolfgang Golubski, Westsächsische Hochschule Zwickau
Tag der mündlichen Prüfung: 17. Juli 2014

Herstellung und Druck: docupoint, Magdeburg
Umschlaggestaltung: University of Bamberg Press

ISSN: 1867-7401
ISBN: 978-3-86309-252-8 (Druckausgabe)
eISBN: 978-3-86309-253-5 (Online-Ausgabe)
URN: urn:nbn:de:bvb:473-opus4-105559

Danksagung

Ich möchte mich bei allen Personen bedanken, die mich während der langen und nicht immer einfachen Zeit meiner Forschungsarbeit direkt und indirekt unterstützt und an schlechteren Tagen aufgemuntert haben. Mein besonderer Dank geht an:

- Meine Eltern für die Unterstützung, das entgegengebrachte Vertrauen und für das Zuhören in Situationen, die für sie vielleicht nicht immer leicht nachzuvollziehen waren.
- Prof. Dr. Guido Wirtz für die unkomplizierte Möglichkeit der kooperativen Promotion, für die fachlichen Diskussionen, das entgegengebrachte Vertrauen bei der Ausgestaltung des Forschungsthemas und die Geduld bis zur Fertigstellung der Arbeit.
- Prof. Dr. Wolfgang Golubski für den initialen Anstoß zum Promotionsvorhaben, seinen Einsatz für das Zustandekommen des Drittmittelprojekts mit SOPHIST, die gewährten Freiheiten bei der Ausgestaltung des Forschungsthemas, kritisch konstruktive Gespräche, die richtigen Worte zur rechten Zeit sowie viel Geduld.
- Dr. Stefan Queins für das entgegengebrachte Vertrauen, die hilfreichen Impulse während der Themenfindung, die vielen fachlichen und nicht immer einfachen Diskussionen, die ehrlichen Einschätzungen, die meist recht kurzfristigen Reviews, die Geduld mit mir und für die aufgewendete Zeit (geplant und nicht geplant, während der Arbeit und in der Freizeit).
- Christine Rupp und Roland Ehrlinger für die Durchführung des Drittmittelprojekts mit der WHZ, das entgegengebrachte Vertrauen und die schöne Zeit während und nach dem Projekt.

- Meine Freundin Gwendolin Lauterbach für so einfache Sachen wie Korrekturlesen und für so schwierige Dinge wie Wortfindungen in einem fremden Fachgebiet, aufbauende Gespräche und viel Verständnis jeglicher Art.
- Meine Arbeitskollegen an der WHZ und bei SOPHIST, insbesondere meine Büronachbarn Oliver Arnold, Tobias Haubold und Nico Herbig für offene Ohren, gute Antworten und für die nicht immer nur fachlich geprägten Gespräche.
- Meine Tochter Laura für den letzten großen Motivationsschub, die Arbeit bis zur ihrer Geburt abzuschließen.

Kurzreferat

Die Entwicklung von Systemen bestehend aus Hardware und Software ist eine herausfordernde Aufgabe für den System-Architekten. Zum einen muss er die stetig steigende Anzahl an System-Anforderungen, inklusive ihrer Beziehungen untereinander, bei der Erstellung der System-Architektur berücksichtigen, zum anderen muss er mit kürzer werdenden Time-to-Market-Zeiten sowie Anforderungsänderungen durch den Kunden bis in die Implementieungsphase hinein umgehen. Die vorliegende Arbeit stellt einen Prozess vor, der dem Architekten die Validierung der System-Architektur gegenüber den architekturrelevanten Anforderungen ermöglicht. Dieser Prozess ist Bestandteil der System-Design-Phase und lässt sich in die iterative Entwicklung der System-Architektur integrieren. Damit der Architekt nicht den Überblick über alle Anforderungen und deren Beziehungen untereinander verliert, fasst er die Anforderungen anhand von architekturspezifischen Aspekten, die sogenannten Validierungsziele, zusammen. Für jedes Validierungsziel werden Validierungszielverfahren und Prüfkriterien zur Bestimmung des Validierungsstatus festgelegt. Falls alle betrachteten Validierungsziele erfüllt sind, d. h. die Ergebnisse der Validierungszielverfahren erfüllen die dazugehörigen Prüfkriterien, erfüllt auch die System-Architektur die dazugehörigen architekturrelevanten Anforderungen. Anstelle von formalen Prüftechniken wie beispielsweise dem Model-Checking bevorzugt der in der Arbeit vorgestelle Ansatz Simulationen als Prüftechnik für die Validierungszielverfahren. Bei der Dokumentation setzt der Ansatz auf die Modellierungssprache Unified Modeling Language (UML). Alle für die Simulationen erforderlichen Daten sind Bestandteil eines UML-Modells. Für die Konfiguration und Durchführung der Simulationen werden diese Daten aus dem Modell ausgelesen. Auf diese Weise wirken sich Modelländerungen direkt auf das Validierungsergebnis der System-Architektur aus. Der in der Arbeit

vorgestellte Prozess unterstützt den Architekten bei der Erstellung einer den architekturrelevanten Anforderungen entsprechenden System-Architektur sowie bei der Auswirkungsanalyse von Anforderungs- oder Architekturänderungen. Die wesentlichen Prozessschritte werden mit Hilfe eines Werkzeugs partiell automatisiert, wodurch die Arbeit des Architekten erleichtert und die Effizienz des Systementwicklungsprozesses verbessert wird.

Abstract

One of the great challenges for a system architect is the design of an architecture containing of software and hardware which fulfills the system requirements. He has to ensure this despite of an increasing number of requirements which are also interconnected and despite of a shortened time-to-market period coming along with requirements changes by the customer up to the implementation phase. This thesis introduces an approach enabling the system architect to validate the system architecture against the architecture-relevant requirements. It is applied in the system design phase and can be integrated into an iterative architecture design. With the help of so-called validation targets, requirements with the same architecture-specific aspects, the architect keeps track of all architecture-relevant requirements. The validation status of these targets are determined by examination processes and check criteria. The system architecture is valid if all check criteria are met by the results of the examination processes, i. e. if all validation targets are valid. The approach introduced prefers simulations for the examination process instead of formal validation techniques like model checking and uses model-based documentation based on the Unified Modeling Language (UML). All data required for the simulations is part of an UML model and extracted to configure and run the simulations. Therefore, changes in the model are affecting the validation result directly. The approach supports the design of an requirement compliant architecture and enables the architect to analyse the impact of architecture or requirements changes. Major steps of the validation process can be partly automated to facilitate the work of the architect and to increase the efficiency of the system development process.

Inhalt

1 Einleitung

> "Nichts ist so beständig wie der Wandel." (frei nach Heraklit von Ephesus)

Diese Aussage Heraklits ist eine passende Beschreibung für eine der großen Herausforderungen in der Systementwicklung: Den Umgang mit anhaltend stattfindenden Veränderungen.

Entwicklungsprojekte sind mit sich häufig ändernden Bedingungen konfrontiert, die eine kontinuierliche Planung und Entwicklung auf Basis vereinbarter Anforderungen fast unmöglich machen. Der Kunde verlangt vom Hersteller größtmögliche Flexibilität bei der Systementwicklung. Änderungen der Anforderungen bis in die Realisierungsphase hinein sind keine Seltenheit. Gleichzeitig steigt die Komplexität der zu entwickelnden Systeme kontinuierlich, was auch mit einer steigenden Anzahl[1] der Anforderungen und deren Beziehungen untereinander einhergeht. Diese Herausforderung wird begleitet durch eine sinkende Zeitspanne vom ersten Prototyp bis zur Etablierung des Produktes am Markt[2]. Um der steigenden Komplexität, den kürzeren Entwicklungszeiten sowie der größtmöglichen Flexibilität gerecht zu werden, werden effiziente Systementwicklungsprozesse benötigt [INC11, Seite 15f.].

Insbesondere bei der Entwicklung von Systemen bestehend aus Software- und Hardware-Komponenten stellen die kontinuierlichen Veränderungen

[1] Hier sind mittlere bis hohe vierstellige oder auch fünfstellige Werte nicht ungewöhnlich.

[2] Die Zeit vom ersten Prototyp eines neuen Produkts bis zu einer signifikanten Marktdurchdringung von 25% hat sich in den letzten 50 Jahren etwa um den Faktor vier reduziert [INC11, Seite 15f.].

eine Herausforderung dar. Hier sind neben den Änderungen an den funktionalen vor allem die Änderungen an den nicht-funktionalen Anforderungen von Bedeutung [CNY95]. Sie beeinflussen die Realisierungsmöglichkeiten für die System-Architektur [Rom85], dem Fundament des zu entwickelnden Systems. Wie beim Hausbau gilt auch hier: Ist das Fundament einmal gegossen, sind grundlegende Änderungen nur noch mit erhöhtem Ressourcenaufwand möglich [CdPL01].

Eine wichtige Aufgabe des Architekten[3] ist die Erstellung einer System-Architektur, die konform gegenüber den System-Anforderungen ist [INC11, Seite 97f.] [RH09, Kapitel 2.3]. Falls Anforderungsänderungen auftreten, muss er deren Auswirkungen auf die Architektur ermitteln (engl. *impact analysis*) und ggf. die Architektur anpassen. Dies ist eine zeit- und damit auch kostenintensive Aufgabe und wiederholt sich bei jeder architekturrelevanten Änderung. Der Aufwand für die Impact-Analyse ist nicht nur von der Anzahl der System-Komponenten abhängig, sondern vor allem vom Komplexitätsgrad des Systems, d. h. von den Abhängigkeiten der System-Komponenten untereinander.

Leider ist in der Praxis häufig zu beobachten, dass die Konformitätsprüfung der System-Architektur gegenüber den Anforderungen nur auf dem Papier stattfindet [WLBF09]. Falls sie doch durchgeführt wird, handelt es sich häufiger um ein einmaliges Ereignis am Ende der Design-Phase. Die Notwendigkeit ist den Beteiligten durchaus bewusst, das Problem liegt vielmehr im benötigten Ressourcenaufwand. Dieser setzt sich zum einen aus der Menge der Anforderungsänderungen und zum anderen aus der häufig verwendeten Prüftechnik *Review* zusammen. Reviews gehören zu den informellen

[3] Hiermit ist die Rolle *Architekt* gemeint, die auch durch mehrere reale Personen besetzt sein kann.

Validierungs- und Verifizierungstechniken [DHJ+10]. Sie sind mühsam, beruhen auf subjektive Einschätzungen der Beteiligten und erzeugen dadurch weder repdroduzierbare Ergebnisse noch bringen sie Potential zur Automatisierung mit sich. Eine Alternative zu den Reviews sind formale Techniken wie beispielsweise das Model-Checking oder Theorem-Proofing. Sie geben Antwort auf konkrete logische Fragestellungen. Für die Ermittlung konkreter Werte, insbesondere zur Analyse des Systemverhaltens, sind sie weniger geeignet. Problematisch ist auch die Prüfung datenintensiver Systeme, bei denen der Definitionsbereich der Daten gegen unendlich geht (State-Explosion-Problem) [BK08]. Formale Techniken werden hauptsächlich für (sicherheits-)kritische Systeme eingesetzt, erfahren aber ansonsten trotz ihres Potentials wenig Akzeptanz in der Industrie und werden eher selten verwendet [WLBF09] [DBH13].

Die Zielsetzung der vorliegenden Arbeit war deshalb die Erarbeitung einer systematischen Vorgehensweise für die Überprüfung der System-Architektur gegenüber den architekturrelevanten System-Anforderungen (siehe auch [PGQ13b]). Dieser Prozess soll sich in das iterative Vorgehen zur Erstellung eines Architekturentwurfs [RH09, Kapitel 4] [Sta11] integrieren und sowohl in frühen als auch in fortgeschrittenen Projektphasen einsetzbar sein. Zur Unterstützung des stetigen Wandels soll der Ansatz einen geringen Aufwand nach Änderungen an Anforderungen oder an der Architektur erzeugen. Als Dokumentationssprache wird der Quasi-Standard in der Industrie für modellbasierte Dokumentation, die Unified Modelling Language (UML), verwendet. Dabei soll sich der Ansatz an die unterschiedlichen projektspezifischen Notationen anpassen lassen und selbst keine Vorgaben an die Notation des Entwicklungsprojekts machen. Bei den Prüfverfahren setzt der Ansatz auf Simulationen, einem dynamischen Validierungs- und Verifi-

zierungsverfahren (V&V-Verfahren)[4]. Sie stellen dem Architekten zusätzliche Informationen zur Verfügung, die ihn bei der Anpassung der Architektur unterstützen. Darüber hinaus sind sie auch für datenintensive Systeme geeignet. Die Wahl eines anderen V&V-Verfahrens soll grundsätzlich möglich sein, wird in der vorliegenden Arbeit aber nicht weiter betrachtet. Der Validierungsprozess soll den Architekten zum einen beim Finden einer zu den Anforderungen konformen System-Architektur, zum anderen bei der Analyse der Auswirkungen von Anforderungs- oder Architekturänderungen unterstützen. Der benötigte Zeit- und Arbeitsaufwand für die Prüfungen während der Systementwicklung soll mit Hilfe von Automatisierungen reduziert werden. Damit trägt der Ansatz dazu bei, die Effizienz des Systementwicklungsprozesses zu verbessern.

Der in der Arbeit vorgestellte Ansatz lässt sich während und am Ende der Design-Phase einsetzen. Die Erhebung der Anforderungen und der Entwurf einer System-Architektur gehören nicht zu seinem Aufgabenbereich. Allerdings müssen weder die Anforderungen noch der Architekturentwurf vollständig sein. Somit unterstützt der Ansatz die nebenläufige Entwicklung von Anforderungen und Architektur, auf deren Bedeutung in [PS07a] hingewiesen wird und die für eine Verwendung in agilen Vorgehensweisen[5] erforderlich ist. In der aktuellen Forschung existieren verschiedene Ansätze, die auf formale Verfahren zur Validierung der Architektur setzen. Dabei liegt der Fokus von Schamai et al. [SHFP10] und dem Werkzeug Simulink [Mat12] eher auf einer physikalischen Architektur und bei Adler et al. [AGMG11] und dem Werkzeug PREEvision [BH12] auf einer elektrisch/elektronischen

[4] Debabbi et al. kategorisieren in [DHJ+10] verschiedene Validierungs- und Verifizierungstechniken. Simulationen zählen dabei zu den dynamischen Verfahren.

[5] Beim agilen Vorgehen werden Anforderungen und Architektur zum benötigten Zeitpunkt erarbeitet. Es existiert keine zeitlich vorgelagerte Phase für die Erhebung der Anforderungen wie in phasenbasierten Vorgehensmodelle (z. B. dem V-Modell).

Architektur. Bei diesen Ansätzen bleiben allerdings einige Ergebnisse der System-Analyse, die Software-Komponenten der System-Architektur und deren Zuordnung zur Hardware unberücksichtigt. Für die Durchführung werden meist detaillierte Informationen über die verwendeten Algorithmen und die System-Architektur benötigt. Gerade dieses Detailwissen steht aber in den frühen Phasen der Entwicklung, bei der Erstellung der ersten Architekturentwürfe, noch nicht zur Verfügung. Durch die Spezialisierung auf eine bestimmte Domäne lässt sich dieses Informationsproblem lösen. Beispielsweise konzentrieren sich Holtmann et al. [HMM11] auf den Automotive-Bereich und Hall [Hal08] und Iqbal et al. [IAB10] auf Realzeit-Systeme. Eine Übertragung auf andere Domänen ist allerdings nicht gegeben. Eine Alternative zu den formalen Verfahren ist die Generierung von Testfällen aus einem autarken Test-Modell zur Prüfung des Designs. Leider werden beim Ansatz von Aydal et al. [APUW09] die für das Design wichtigen nichtfunktionalen Anforderungen nicht weiter berücksichtigt. Verfahren wie die Software Architecture Analysis Method (SAAM) und Architecture Tradeoff Analysis Method (ATAM) [CKK02] dienen der Bewertung verschiedener Architekturentwürfe, der in der Arbeit vorgestellte Ansatz hat aber das Ziel, die Konformität eines Architekturentwurfs gegenüber den Anforderungen zu prüfen.

Der Kern der erarbeiteten Resultate ist ein partiell automatisierbarer Validierungsprozess, der sich in das iterative Vorgehen zum Entwurf bzw. zur Anpassung der System-Architektur integrieren lässt (siehe auch [PGQ11a]). Sogenannte Validierungsziele fassen alle Anforderungen eines architekturspezifischen Aspekts zusammen. Sie geben dem Architekten einerseits einen besseren Überblick über die Einflussfaktoren der Architektur und verdeutlichen andererseits die zahlreichen Anforderungsbeziehungen auf einem für die Architektur passenden Abstraktionsniveau. Ein Validierungsziel enthält

per Definition kein Prüfkriterium. Dies ergibt sich durch die zugeordneten Anforderungen. Zur Feststellung, ob ein Prüfkriterium erfüllt ist, legt der Architekt für jedes Validierungsziel mindestens ein sogenanntes Validierungsverfahren fest. Die Validierung der System-Architektur ist erfolgreich, falls alle Validierungsziele erfolgreich validiert werden.

Zur Entkopplung von Systementwicklung und Architekturvalidierung wird ein von der Dokumentation des zu entwickelnden Systems separates UML-Modell, das Validierungsmodell, erstellt. Es enthält alle validierungsrelevanten Informationen und dient damit als Datenbasis für die Validierungsverfahren. Die validierungsrelevanten Informationen stammen entweder aus dem Entwicklungsmodell (Modell-zu-Modell-Transformation) oder werden durch eine validierungsspezifische Notation (VSN) hinzugefügt. Die Modellierung der VSNs erfolgt mit Hilfe des UML-Profilmechanismus' [Obj11a]. Auf diese Weise lässt sich eine projektspezifische Notation um beliebige validierungsspezifische Inhalte erweitern. Der Ansatz gibt aber keinerlei Vorgaben für die Notation des Entwicklungsmodells.

Für jedes Validierungsziel wird eine VSN erstellt, mit der die für das zugehörige Validierungsverfahren benötigten Informationen im sogenannten Validierungsmodell modelliert werden. Die Modelldaten werden zum einen für die Validierung und zum anderen für eine lückenlose Dokumentation des Architektur-Entwurfs, dessen Entstehung und Anpassung genutzt. Änderungen zum Finden einer passenden System-Architektur werden direkt am Validierungsmodell vorgenommen. Der Ansatz reduziert den Arbeits- und Zeitaufwand für die Validierung, indem möglichst viele Validierungsverfahren automatisiert durchgeführt und die dazugehörigen Werte direkt aus dem Validierungsmodell ausgelesen werden. Der Architekt wird beim Entwurf und der Impact-Analyse nach Anforderungsänderungen unterstützt und Än-

derungen an der Architektur können aufgrund der lückenlosen modellbasierten Dokumentation nachvollzogen werden.

Im Folgenden wird ein Überblick über die vorliegende Arbeit gegeben:

In *Kapitel 2* werden die Definitionen verschiedener in der Arbeit verwendeter Begriffe beschrieben. Neben der Festlegung der Semantik für die Begriffe System, System-Architektur, Anforderung und Simulation werden auch die Validierung und Verifizierung, Validierungsprozess sowie modellgetriebene Validierung festgelegt. Die Bedeutung der Begriffe Validierung und Verifizierung sind in der Literatur nicht eindeutig festgelegt. Die für die vorliegende Arbeit geltende Definition ist in Kapitel 2.5.2 beschrieben.

Kapitel 3 gibt eine Einleitung in die Domäne Radar und die Radarsignalverarbeitung. Es werden die grundsätzliche Funktionsweise eines Radars sowie die Signalverarbeitung eines Radarprozessors vorgestellt. Diese Domäne bildet die Grundlage für die in Kapitel 4 und 6 verwendeten Beispiele zur Erläuterung des Validierungsprozesses.

Das *Kapitel 4* stellt zunächst eine vereinfachte Version des Validierungsprozesses vor. Es nähert sich dem Prozess aus der Vogelperspektive und stellt die Verbindungen zu den Systementwicklungsphasen her. Es nennt die Voraussetzungen zur Anwendung des Validierungsprozesses und definiert die vom Prozess verwendeten Begriffe. Die Prozessschritte werden einzeln beschrieben und durch ein Beispiel aus der Domäne Radar verdeutlicht.

Kapitel 5 nimmt die Einordnung des Validierungsprozesses in die verschiedenen Prozessmodelle der Systementwicklung und die Abgrenzung zu verschiedenen bereits existierenden Ansätzen, Techniken und Methoden vor. Durch die vorhergehende Vorstellung der vereinfachten Version des Vali-

dierungsprozesses wird die Abgrenzung zu den bestehenden Arbeiten erleichtert.

Kapitel 6 beschreibt die Anwendung des Prozesses auf ein Radar-System. Dabei werden auf Basis von System-Anforderungen und einem dazu erstellten ersten Architekturentwurf die verschiedenen architekturrelevanten Aspekte inklusive ihrer Abhängigkeiten herausgearbeitet. Anschließend werden für dieses Beispiel die verschiedenen Prozessschritte und eine Validierung des ersten Architekturentwurfs durchgeführt. Anhand von Szenarien wird gezeigt, wie der Validierungsprozess auf die verschiedenen Änderungen in der Design-Phase reagiert.

Kapitel 7 stellt den Werkzeugprototypen MEASURED[6] vor, der den Architekten bei verschiedenen Schritten des Validierungsprozesses unterstützt. Die Vorteile der modellbasierten Dokumentation für den Validierungsprozess werden durch die Beschreibung des Konzepts für einen generischen Simulator verdeutlicht. Sowohl der Werkzeugprototyp als auch der generische Simulator ermöglichen dem Architekten, den Validierungsprozess effizienter durchzuführen und damit die Effizienz des Systementwicklungsprozesses Design zu verbessern. Das Thema Werkzeugunterstützung wird in diesem Kapitel mit einem Ausblick auf weitere Unterstützungsmöglichkeiten abgeschlossen.

Kapitel 8 geht auf die Anpassung des Validierungsprozesses an spezifische Projektrandbedingungen ein. Dazu wird eine erweiterter Ablauf des Validierungsprozesses vorgestellt, mit dessen Hilfe projektspezifisches Wissen für die Wiederverwendung in einem weiteren Projekt vorbereitet wird. Es werden Modellierungsalternativen für die VSN vorgestellt, ihre Vor- und Nachteile diskutiert und als Hilfe für den Architekten bei der Wahl der Modellierungsart tabellarisch zusammengefasst. Auch auf die Frage, wo die va-

[6] Modellgetriebene Validierung von System-Architekturen mit der UML

lidierungsspezifischen Daten dokumentiert werden, gibt es mehr als eine Antwort. Das Kapitel stellt beide Lösungen vor und diskutiert ihre Vor- und Nachteile für verschiedene Projektrandbedingungen.

Kapitel 9 beschreibt die Anwendung des Validierungsprozesses auf eine nicht-technische Domäne. Als Beispiel dient ein Workflow-System. Workflow-Systeme entstammen einer eher organisatorisch geprägten Domäne, die sich mit Prozessen innerhalb eines Unternehmens und den dafür erforderlichen Ressourcen beschäftigt. Der Architekt eines Workfow-Systems steht hier nicht nur der Herausforderung gegenüber, die abstrakten Prozesse zu modellieren, sondern eine Architektur zu erarbeiten bzw. anzupassen, die eine Bearbeitung aller konkreten Abläufe in einem Unternehmen gemäß den Unternehmensvorgaben ermöglicht.

Kapitel 10 stellt die Resultate der praktischen Verprobungen des Ansatzes vor und nimmt eine kritische Betrachtung des Validierungsprozesses in Bezug auf seine Einsatzmöglichkeiten und die Aussagekraft der Resultate vor.

Kapitel 11 fasst die wesentlichen Ergebnisse der Arbeit noch einmal zusammen und gibt einen Ausblick auf noch nicht oder nicht weiter betrachtete Aspekte der Arbeit.

Im Rahmen des Promotionsprozesses wurden einige Inhalte dieser Arbeit mit dem jeweils aktuellen Forschungsstand bei verschiedenen Konferenzen veröffentlicht. Die ersten Forschungsergebnisse, insbesondere die Beschreibung des Validierungsprozesses (siehe Kapitel 4.3), wurden in [PGQ13b] veröffentlicht. In [PGQ11b] liegt der Fokus auf eine weiterentwickelte Fassung des Prozesses inklusive Anwendungsbeispiel sowie auf den Vorteilen des Ansatzes insgesamt. [PGQ11a] konzentriert sich auf die für den Ansatz verwendete Notationsweise der validierungsspezifischen Daten im Validierungsmodell (siehe Kapitel 8.2). Einen Überblick zu den in Kapitel 5.3 be-

schriebenen vergleichbaren Ansätzen gibt [PGQ11c]. Das in Kapitel 7.1 beschriebene Werkzeug MEASURED wurde in [PGQ12] vorgestellt. Das Konzept des in Kapitel 7.2 beschriebenen generischen Simulators sowie eine Beschreibung der detaillierten Fassung des Validierungsprozesses (siehe Kapitel 8.1) sind als Beitrag eines Sammelbands [PGQ13a] veröffentlicht worden.

2 Begriffe

Die Festlegung der Semantik von Worten bildet das Fundament für ein gemeinsames Verständnis von Sprache. Aus diesem Grund enthält dieses Kapitel die Definitionen der wichtigsten Begrifflichkeiten in der vorliegenden Arbeit. Für einige Begriffe gibt es mehrere Definitionen, die sich zum Teil nur wenig, zum Teil aber auch deutlich voneinander unterscheiden. Hier reicht die reine Nennung der in der Arbeit verwendeten Definition nicht aus, um alle Facetten zu beleuchten. Für die Begriffe Anforderung sowie Validierung und Verifizierung werden deshalb verschiedene Definitionen aufgeführt und miteinander verglichen, bevor die Festlegung auf eine Definition erfolgt bzw. eine für die vorliegende Arbeit gültige Definition vorgestellt wird. Der in der Arbeit häufig verwendete Begriff des Validierungsprozesses wird in Kapitel 2.6 definiert.

2.1 System und System-Architektur

In der vorliegenden Arbeit werden die Begriffe System und System-Architektur nach der Definition des International Council on Software Engineering (INCOSE)-Handbuchs (siehe [INC11]), das konsistent mit der Norm International Organization for Standardization (ISO) 15288 [ISO08] ist, verwendet. Demnach ist ein System

> "a combination of interacting elements organized to achieve one or more stated purposes". [INC11, S. 5]

Etwas anders ausgedrückt:

> "[A system is] an integrated set of elements, subsystems, or assemblies that accomplish a defined objective. These elements include products (hardware, software, firmware), processes, people, information, techniques, facilities, services, and other support elements." [INC11, S. 5]

Die erste Definition ist sehr allgemein gehalten, wobei die zweite mehr auf die interagierenden Elemente eingeht. Ein System besteht demnach nicht nur aus Software, auch wenn im Bereich der Softwareentwicklung der Begriff System häufiger als Synonym für Software verwendet wird.
Für die Architektur sind die Beziehungen zwischen Hardware und Software von besonderer Bedeutung, was sich auch in der kurz gehaltenen Definition des Begriffs System-Architektur wiederfindet. System-Architektur ist

> "the selection of the types of system elements, their characteristics, and their arrangement". [INC11, S. 98]

Maier et. al. schließen in ihrer Definition explizit die Bedingungen mit ein: "The structure - in terms of components, connections, and constraints - of a product, process, or element." [MR09, S. 423]. Diese Bedingungen sind in den nicht-funktionalen Anforderungen[7] beschrieben, die eine bedeutende Rolle im Validierungsprozess spielen.

Die Verwendung der Begriffe Entwicklung, Architekt, Architektur und Anforderungen erfolgt in der vorliegenden Arbeit immer im Kontext des Systems; mit Architektur ist also die System-Architektur gemeint. Sollten die

[7] Der Begriff der nicht-funktionalen Anforderung ist in der Literatur nicht eindeutig definiert. Kapitel 2.4 stellt hierzu einige Definitionen vor. Die vorliegende Arbeit verwendet den Begriff nach der Definition von [Rup09b] (siehe Seite 32).

Begriffe in einem anderen Kontext stehen, so wird dieser explizit angegeben, z. B. Software-Architektur.

2.2 Systems Engineering

Eine gute Beschreibung für den Begriff Systems Engineering (SE) liefert das INCOSE-Handbuch:

> "Systems Engineering is a interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufactoring, and disposal. SE considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the user needs." [INC11, S. 7]

Zusammenfassend ausgedrückt findet Systems Engineering in einer frühen Phase der Entwicklung statt. Es werden die Kundenwünsche in Form von Anforderungen dokumentiert, das Design wird erarbeitet und eine Überprüfung der Design-Ergebnisse gegenüber den Anforderungen durchgeführt. Das Ziel ist ein Produkt, das den Wünschen des Kunden entspricht.
Systems Engineering ist ein iterativer Prozess, der Lernen und kontinuierliche Weiterentwicklung unterstützt. Dies ist notwendig, um das mit fortlaufender Entwicklung steigende Verständnis zu nutzen. Es kommen neue, (bisher) vergessene Anforderungen hinzu und vorhandene werden entspre-

chend des besseren Verständnisses anders interpretiert [INC11, Kapitel 2.2]. Die Basis für die System-Architektur verändert sich demnach kontinuierlich durch die fortschreitende Entwicklung. Zu diesen durch die Entwicklung entstehenden Änderungen kommen noch die Änderungswünsche des Kunden hinzu, die zeitlich nicht auf die frühen Entwicklungsphasen beschränkt sind.

2.3 Design

In der vorliegenden Arbeit wird der Begriff *Design* als Synonym für den "Architectural Design Process" in der ISO/International Electrotechnical Commission (IEC) 25288:2008 verwendet. Das Ziel dieses Prozesses ist die Entwicklung einer Architektur, welche den System-Anforderungen genügt. Während dieses Vorgehens werden verschiedene Lösungsmöglichkeiten erarbeitet, dokumentiert und auf ihre Eignung hin untersucht. Die Ergebnisse bilden die Basis für die Erstellung und Überprüfung des zu entwickelnden Systems. Zu den Ergebnissen zählen neben der Architekturbeschreibung beispielsweise auch Schnittstellen-Anforderungen, Anforderungen an die System-Komponenten, deren Verbindung zu den System-Anforderungen sowie eine Menge der verwendeten Auswahlkriterien für die Wahl der optimalen Architektur. Für diese Ergebnisse des "Architectural Design Process" wird ebenfalls der Begriff Design verwendet. Um eine Verwechslung zu vermeiden, wird in der vorliegenden Arbeit der Begriff Design nur als Synonym für den "Architectural Design Process" bzw. für die "Design-Phase" verwendet. Die Architektur ist damit ein Teilergebnis des Designs und der in der Arbeit vorgestellte Ansatz unterstützt den Architekten bei der Erarbeitung dieses Teilergebnisses (siehe auch Kapitel 5.2.1).

2.4 Anforderung

Wird drei Requirements-Engineers die Frage gestellt, was sie unter einer Anforderung verstehen, sollte es einen nicht verwundern, wenn in den Antworten mehr als drei verschiedene Definitionen enthalten sind. Dies trifft besonders auf die nicht-funktionale Anforderung (NFA) zu. Aufgrund ihrer besonderen Bedeutung für den Erfolg eines Projektes und der System-Architektur [FD96] [CNYM99] [CdPL01] werden in den folgenden Unterkapiteln verschiedene Definitionen vorgestellt. Bei der Semantik hält sich die vorliegende Arbeit an die Definitionen von [Rup09b] (siehe Kapitel 2.4.2).

2.4.1 Pohl

Eine häufig verwendete Definition für den Begriff Anforderung liefert das Institute of Electric and Electronic Engineers (IEEE) Standard Glossary of Software Engineering Terminology [IEE90]. Pohl übersetzt diese Definition in [Poh08] in die deutsche Sprache:

> "Eine Anforderung ist:
>
> 1. Eine Bedingung oder Eigenschaft, die ein System oder eine Person benötigt, um ein Problem zu lösen oder ein Ziel zu erreichen
>
> 2. Eine Bedingung oder Eigenschaft, die ein System oder eine Systemkomponente aufweisen muss, um einen Vertrag zu erfüllen oder einem Standard, einer Spezifikation oder einem anderen formell auferlegten Dokument zu genügen

3. Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft wie in 1 oder 2 definiert" [Poh08, S. 13]

Demnach definieren Anforderungen "Wünsche und Ziele von Benutzern" [Poh08, ebd.] sowie "Bedingungen und Eigenschaften des zu entwickelnden Systems" [Poh08, ebd.]. Dabei können Anforderungen dokumentiert oder undokumentiert sein. Zur besseren Unterscheidung definiert Pohl den Begriff des Anforderungsartefakts:

> "Ein Anforderungsartefakt ist eine dokumentierte Anforderung." [Poh08, S. 14]

Pohl unterteilt Anforderungen in drei Arten:

- Funktionale Anforderungen,
- Qualitätsanforderungen und
- Rahmenbedingungen.

Dabei definiert er diese Arten folgendermaßen:

> "Eine funktionale Anforderung definiert eine vom System bzw. von einer Systemkomponente bereitzustellende Funktion oder einen bereitzustellenden Service. Als Benutzeranforderung kann eine funktionale Anforderung sehr allgemein beschrieben sein. Als Bestandteil einer Spezifikation beschreibt eine funktionale Anforderung detailliert die Eingaben und Ausgaben sowie bekannte Ausnahmen." [Poh08, S. 15] (angelehnt an [Som04])

> "Eine Qualitätsanforderung definiert eine qualitative Eigenschaft des gesamten Systems, einer Systemkomponente oder einer Funktion." [Poh08, S. 16]

> "Eine Rahmenbedingung ist eine organisatorische oder technologische Anforderung, die die Art und Weise einschränkt, wie ein Produkt entwickelt wird." [Poh08, S. 18] (deutsche Übersetzung aus [RR06])

Für funktionale Anforderungen gibt es drei Perspektiven, aus denen diese dokumentiert werden: die Funktions-, Verhaltens- und Datenperspektive. Eine detaillierte Beschreibung ist in [Poh08, Kapitel 13] enthalten.

Für die Architektur sind Qualitätsanforderungen von besonderer Bedeutung. Pohl nennt sie auch die primären Architekturtreiber [Poh08, S. 15], da die Qualitätsmerkmale meist das ganze System betreffen und dadurch die Struktur der Architektur beeinflussen. Eine andere Form der Beeinflussung liefern die Rahmenbedingungen. Sie richten sich an das System oder an den Entwicklungsprozess und sind für die Projektbeteiligten meist unveränderlich. Rahmenbedingungen können sowohl funktionale Anforderungen als auch Qualitätsanforderungen einschränken und führen zur Reduzierung der Menge möglicher Architekturen oder Umsetzungsmöglichkeiten von Funktionen.

Über den Begriff der nicht-funktionalen Anforderung sagt Pohl, dass es sich bei diesen um einen Sammelbegriff für unterspezifizierte funktionale Anforderungen und Qualitätsanforderungen handelt. Durch die Einordnung der unterspezifizierten funktionalen Anforderungen als nicht-funktionale

Anforderungen bleibe ein Hinterfragen der genauen Anforderungsinhalte und damit eine Konkretisierung häufig aus, was zu ungenauen Anforderungen, d. h. einer schlechten Anforderungsqualität, führe. Er rät deshalb von der Verwendung dieses Begriffs in Anforderungsspezifikationen ab und weist darauf hin, dass die Konkretisierung einer unterspezifizierten funktionalen Anforderung wiederum funktionale Anforderungen und Qualitätsanforderungen hervorbringt.

2.4.2 Rupp

Christine Rupp und die SOPHISTen verweisen in [Rup09b] auch auf die IEEE-Definition für den Begriff der Anforderung. Allerdings benennen sie auch eine eigene, eher praxisorientierte Definition:

> "Eine Anforderung ist eine Aussage über eine Eigenschaft oder Leistung eines Produktes, eines Prozesses oder der am Prozess beteiligten Personen." [Rup09b, S. 14]

Im Vergleich mit der IEEE-Definition, die in [Poh08] verwendet wird, ist die Definition deutlich kompakter und unterscheidet nicht zwischen dokumentierten und nicht-dokumentierten Anforderungen. Außerdem wird, angelehnt an das V-Modell, der Begriff Produkt anstelle von System verwendet, unter dem neben dem System auch beispielsweise Testfälle, Handbücher, Protokolle und Planungsdokumente verstanden werden.

Auch bei der Unterteilung der Anforderungen gehen die SOPHISTen einen anderen Weg als beispielsweise [Poh08]. Es werden die folgenden Anforderungsarten unterschieden:

- Funktionale Anforderungen
- Qualitätsanforderungen
- Technologische Anforderungen
- Anforderungen an die Benutzungsoberfläche
- Anforderungen an sonstige Lieferbestandteile
- Anforderungen an durchzuführende Tätigkeiten
- Rechtlich-vertragliche Anforderungen

Für den Begriff der funktionalen Anforderung wird folgende Definition genannt:

> "Eine funktionale Anforderung definiert eine vom System oder von einer Systemkomponente bereitzustellende Funktion des betrachteten Systems." [Rup09b, S. 18]

Dies ist auch der erste Satz der Definition einer funktionalen Anforderung von Pohl, was nicht weiter verwundert, da beide Autoren Mitglied im International Requirements Engineering Board (IREB) e.V.[8] sind. Das Verständnis für den Begriff wird durch die folgende Beschreibung noch etwas deutlicher:

> "Funktionale Anforderungen beschreiben Aktionen, die von einem System selbstständig ausgeführt werden sollen, Interaktionen des Systems mit menschlichen Nutzern oder Systemen (Eingaben, Ausgaben) und Anforderungen zu allgemeinen,

[8] Der IREB e.V. [IRE13] setzt sich unter anderem für eine "Standardisierung der Aus- und Weiterbildung im Requirements Engineering" [IRE13, Mission] ein und baut dafür ein (unter den Beteiligten konsolidiertes) Glossar [IG11] auf. In diesem Glossar können allerdings nicht alle Meinungen enthalten sein, so dass sich einige Begriffsdefinitionen bei den Mitglieder weiterhin unterscheiden. Ein gutes Beispiel ist der Begriff der nicht-funktionalen Anforderung.

> funktionalen Vereinbarungen und Einschränkungen." [Rup09b, S. 18]

Die Definition der nicht-funktionalen Anforderungen fällt bei den SOPHISTen sehr pragmatisch und zirkulär aus: "Alle Anforderungen, die nicht funktional sind, sind nicht-funktionale Anforderungen." [Rup09b, S. 249]. Anders ausgedrückt: Qualitätsanforderungen[9], technologische Anforderungen, Anforderungen an die Benutzungsoberfläche, Anforderungen an sonstige Lieferbestandteile, Anforderungen an durchzuführende Tätigkeiten und rechtlich-vertragliche Anforderungen sind im Sprachgebrauch der SOPHISTen nicht-funktionale Anforderungen. Für die System-Architektur sind vor allem die technologischen Anforderungen und die Qualitätsanforderungen von besonderer Bedeutung. Einen indirekten Einfluss können auch rechtlich-vertragliche Anforderungen haben.

Die Definition des Begriffs Qualitätsanforderung unterscheidet sich kaum von der in [Poh08] genannten, falls die Verwendung des Begriffs *Produkt* anstelle des Systems (siehe oben) berücksichtigt wird. Damit gilt die in 2.4.1 aufgeführte Begründung zum Einfluss der Qualitätsanforderungen auf die Architektur auch an dieser Stelle.

"Technologische Anforderungen beschreiben die Anforderungen an das System, die einen direkten Einfluss auf die in der Entwicklung zu verwendenden Technologien haben." [Rup09b, S. 255] Sie beeinflussen also ein Stück weit die Realisierung des Systems oder geben Lösungen vor. Somit gibt ein Teil der technologischen Anforderungen Vorgaben an die Architektur.

"Unter rechtlich-vertraglichen Anforderungen verstehen wir alle Spezifikationen zu vereinbarten Rechten und Pflichten in Bezug auf Entwicklung und

[9] Die Norm ISO/IEC 25000 [ISO05] unterteilt Qualitätsanforderungen in die Merkmale: Änderbarkeit, Benutzbarkeit, Effizienz, Funktionalität, Übertragbarkeit und Zuverlässigkeit.

Verwendung des zu erstellenden Produkts." [Rup09b, S. 280] Darunter fallen beispielsweise Vorgaben zu den maximalen Beriebskosten des Systems, wodurch preisintensive Hardware mit einer entsprechend kurzen Lebenszeit oder ein hoher Ressourcenverbrauch (Wasser, Strom, usw.) unter Umständen[10] nicht möglich ist.

2.4.3 Dörr

Jörg Dörr beschreibt in seiner Dissertation [Dör10] einen Prozess zur Erfassung einer vollständigen Menge von NFAs (NFA-Prozess). Der Fokus der Arbeit liegt auf den NFAs, weshalb er keine explizite Definition für die Begriffe Anforderung und funktionale Anforderung liefert. Auf den Anforderungsbegriff geht er nicht ein, über die funktionalen Anforderung schreibt er: "To summarize, almost all definitions agree that the functional requirements describe what the system is supposed to do, i.e. they describe the behavior or services of the system." [Dör10, S. 16]. Semantisch stimmt dies mit Rupp und Pohl überein.

Bevor er auf die Begriffsdefinition für die NFA eingeht, nimmt Dörr eine Anforderungsklassifizierung vor. Sie ist in Abbildung 2.1 [Dör10, S. 21] zusammen mit einigen Beispielen für die verschiedenen Klassifizierungen dargestellt. Auf der obersten Ebene gibt es eine Aufteilung in *Organizational Requirements* und *System-/Product-Requirements*. Organizational Requirements werden weiter in *Process Requirements* und *Project Requirements* unterteilt.

[10] An dieser Stelle kommt es natürlich auf alle beeinflussenden Faktoren an. Ein hoher Ressourcenverbrauch kann akzeptabel sein, falls dadurch die Lebenszeit benötigter Hardware verlängert wird.

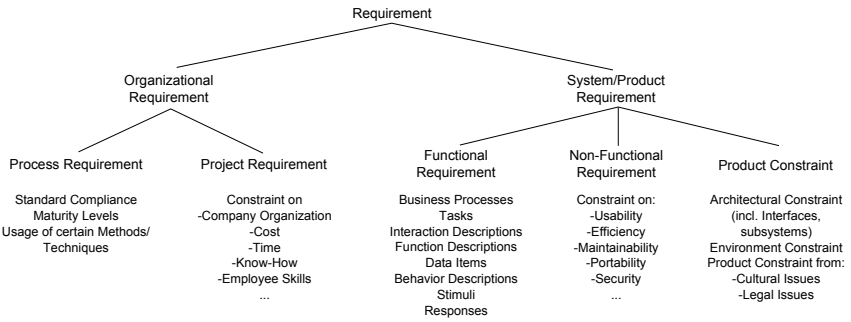


Abbildung 2.1: UnterteilungAnforderungsklassifizierung nach Dörr

Bei den System-/Product-Requirements[11] wird zwischen *Functional Requirements*, *Non-Functional Requirements* und *Product Constraints* unterschieden. Die Wortwahl in der Klassifizierung zeigt, dass die Ermittlung einer vollständigen Menge von NFAs sich auf die System-/Product-Requirements beschränkt. Einschränkungen (engl. *constraints*) an den Entwicklungsprozess (Process-Requirements) oder an das Projekt (Project-Requirements) gehören für Dörr begrifflich nicht zu den NFAs.

Den Begriff nicht-funktionale Anforderung (engl. *non-functional requirement*, kurz NFR) definiert Dörr wie folgt:

> "A non-functional requirement (NFR) constraints one or more functional conceptual elements or subsystems by determining values (or value domains) for one or more metrics of a specified elementary quality attribute that should be achieved by the functional conceptual element or subsystem. The non-functional requirement is of the type of the elementary quality attribute that

[11] Produkt-Requirements und System-Requirements werden synonym verwendet.

characterizes the functional conceptual element or subsystem." [Dör10, S. 26]

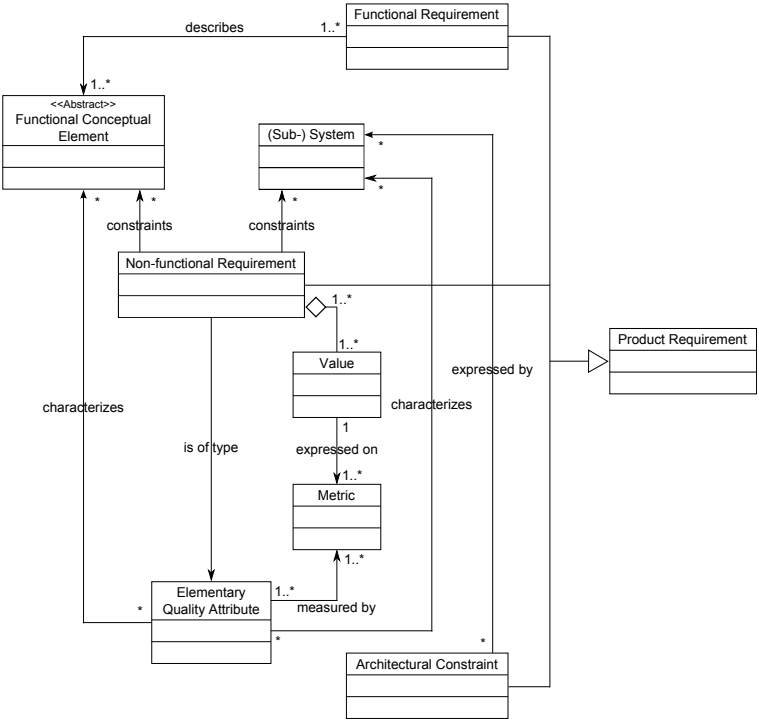


Abbildung 2.2: Vereinfachtes Anforderungsmetamodell

Diese Definition ist eine verbalisierte Form der Beziehungen einer NFA zu den Elementen Functional Conceptual Element (FCE), Value, Subsystem, Metric und Elementary Quality Attribute (EQA). Diese (und weitere) Beziehungen hat Dörr in einem Metamodell festgehalten. Ein Ausschnitt daraus ist in Abbildung 5.2 [Dör10, S. 24] dargestellt. In ihm sind die bereits angesprochenen Product Requirements, Functional Requirements, Non-Functional Requirements und die Architectural Constraints zu sehen.

Die *Architectural Constraints* sind eine für den vorgestellten Ansatz zur Ermittlung der NFAs relevante Untermenge der Product Constraints. Die Unterteilung der Product Requirements ist hier durch eine Vererbungsbeziehung modelliert.

Die abstrakten FCEs dienen als Platzhalter für Benutzer- und Systemaufgaben sowie Datenlementen, die durch funktionale Anforderungen beschrieben werden. NFAs haben eine einschränkende Wirkung auf Subsysteme und FCEs. Sie besitzen einen Typ, das EQA, und mindestens ein zugewiesenes Value-Element, wodurch sie mit Hilfe der dazugehörigen Metrik messbar und damit analytisch auswertbar sind. EQAs sind die "quality characteristics" [Dör10, S. 24] der FCEs und der Subsysteme (siehe *characterizes*-Verbindungen in Abbildung 2.2), weshalb vier Ausprägungen existieren: User Task Quality Attribute (QA), System Task QA, Data Item QA und Subsystem QA. Eine ausführliche Beschreibung des Anforderungsmetamodells und wie die Ergebnisse des NFA-Prozesses für den Validierungsprozess genutzt werden können, sind in Kapitel 5.1.1 und 5.2.5 zu finden. Zusätzlich stellt Anhang A.1 eine Zusammenfassung des NFA-Prozesses bereit.

Im Vergleich mit den Definitionen von Pohl (Kapitel 2.4.1) und Rupp (Kapitel 2.4.2) lässt sich folgendes zusammenfassen:
Dörr gibt in seiner Arbeit keine explizite Definition der Begriffe Anforderung und funktionale Anforderung an. Er unterteilt Anforderungen in Organizational Requirements und Product Requirements. Sowohl bei Pohl als auch bei Rupp richten sich die Anforderungen hauptsächlich an das Produkt bzw. das System und weniger an den Prozess und das Projekt. Werden der Prozess und das Projekt jedoch in den Kontext des Produktes verschoben - beide dienen ja nur dem Zweck das Produkt zu erstellen - lassen sich diese Anforderungen auch bei Pohl und Rupp einordnen. Bei Pohl wären sie in

den Rahmenbedingungen enthalten, bei Rupp gehörten sie zu der Menge der NFAs, z. B. der Qualitätsanforderungen, der rechtlich-vertraglichen Anforderungen, der Anforderungen an durchzuführende Tätigkeiten und der Anforderungen an sonstige Lieferbestandteile. Die Unterteilung der Product Requirements ähnelt den drei Anforderungsarten von Pohl. Bei den funktionalen Anforderungen herrscht Namensgleicheit, die NFAs entsprechen den Qualitätsanforderungen und die Product Constraints lassen sich in die Rahmenbedingungen[12] einordnen. Nach dem Verständnis von Rupp gehören die Product Constraints zur Menge der NFAs. Das semantische Verständnis aller drei Autoren für den Begriff funktionale Anforderung ist gleich.

Für den NFA-Prozess ist nicht nur eine eindeutige, analytisch auswertbare Definition der NFAs notwendig, sondern auch eine Beschreibung aller relevanten Elemente und deren Beziehungen unter- und miteinander. Dörr beschreibt dies in seinem Anforderungsmetamodell und bietet damit die umfangreichste Definition für den NFA-Begriff an. Es bedarf einiger Einarbeitungszeit, um diese Definition mit allen Verbindungen zu den restlichen Elementen des (vollständigen) Metamodells nachzuvollziehen. Die Zusammenhänge und Abhängigkeiten sind dadurch aber deutlich besser nachzuvollziehen als durch die textuellen Beschreibungen bei Pohl, Rupp und vielen anderen Autoren. Ein grundlegend unterschiedliches Verständnis der NFAs gegenüber Rupp und Pohl ist jedoch nicht zu erkennen. Sowohl Dörr als auch Rupp unterteilen die NFAs in Kategorien, wobei Rupp sechs und Dörr vier[13] Hauptkategorien verwendet.

[12] Rahmenbedingungen enthalten das Produkt einschränkende technologische Anforderungen.

[13] Hierbei handelt es sich um die vier "quality characteristics": Data QA, User Task QA, System Task QA und Subsystem QA.

2.5 Validierung und Verifizierung

In der Literatur sind verschiedene Beschreibungen und Definitionen zu den Begriffen Validierung und Verifizierung zu finden. Eine eindeutige Definition für die beiden Begriffe lässt sich nicht aus der Literatur entnehmen. Die Unterschiede reichen von kleinen semantischen Feinheiten bis zu einer kontextspezifischen Deutung der Begriffe. Die untersuchten Quellen sind in Kapitel 2.5.1 nach Autoren bzw. Quelle getrennt beschrieben. Um den Interpretationsspielraum für die vorliegende Arbeit zu reduzieren, werden in Kapitel 2.5.2 die zwei in der vorliegenden Arbeit geltenden Begriffsdefinitionen festgelegt und erläutert. Darauf basierend wird im nachfolgenden Kapitel 2.6 der Begriff Validierungsprozess definiert.

2.5.1 Definitionen in der Literatur

Boehm Eine bekannte und vor allem in der Praxis häufig verwendete Erklärung für die beiden Begriffe Validierung und Verifizierung stammt von Barry W. Boehm. Er sieht die Validierung und Verifizierung als integralen Bestandteil der einzelnen Software-Lebenszyklus-Phasen, wobei er sich an dieser Stelle auf die Phasen des Wasserfall-Modells bezieht. Das Ziel ist der Nachweis, dass die Ergebnisse einer Phase auch den dazugehörigen Zielen entsprechen. Dabei versteht er unter Validierung: "To establish the fitness or worth of a software product for its operational mission (from the Latin *valere*, 'to be worth')" ([Boe81, S. 37])). Für die Validierung ist der Nachweis zu erbringen, dass ein Software-Produkt passend oder von Wert für seinen praktischen Einsatz ist. Unter Verifizierung versteht er: "To establish the truth of correspondence between a software product and its specification (from the Latin *veritas*, 'truth')" (ebd.). Bei der Verifizierung ist der Nach-

weis zu erbringen, dass eine wahrheitsgemäße Übereinstimmung zwischen Software-Produkt und dessen Spezifikation erreicht wurde.

Interessant ist hier das Wort *wahrheitsgemäß*, weil es abhängig vom Standpunkt unterschiedliche Wahrheiten gibt. Eine etwas freiere Übersetzung wäre: Das Software-Produkt muss den Wünschen des Kunden, die er in Form der Spezifikation dokumentiert hat, entsprechen. Bei der Festlegung der Sprachsemantik ist also die Interpretation des Kunden, d. h. dessen Wahrheit, ausschlaggebend, nicht die des Entwicklers.

Zur Vereinfachung wandelt Boehm diese Definitionen in zwei kurze informelle Fragen um: "Are we building the right product?" (ebd.) für die Validierung und "Are we building the product right?" (ebd.) für die Verifizierung. Die relativ kurzen Fragelängen und die annähernd gleiche Wortwahl erleichtern das Verinnerlichen dieser informellen Definitionen, wobei sich der semantische Unterschied anhand der Position des Wortes *right* herleiten lässt.

Hopkins In einem Sammelband über Validierung und Verifizierung komplexer Systeme beschreibt Hopkins Validierung als datensammelnden Prozess ("collecting process") und die Verifizierung als einen bewertenden Prozess ("assessment process" [Hop93, S. 11]) und zieht daraus folgende Schlussfolgerungen:

- Validierung und Verifizierung sind eher serielle als parallele Prozesse.
- Verifizierung findet normalerweise vor der Validierung statt, falls beide Prozesse durchgeführt werden.
- Es sollten beide Prozesse verwendet werden, außer einer ist hinreichend.

- Validierung und Verifizierung sollten in Beziehung zueinander geplant werden.
- Sie sollten als komplementär und gegenseitig unterstützend anstatt als Alternative zueinander verstanden werden.

Diese strikte Trennung der Datensammlung und ihre Bewertung lässt sich in keiner anderen Quelle wiederfinden. Sie zeigt aber, dass ein unterschiedliches Verständnis der Begriffe existiert. Die Interpretation der Validierung und Verifizierung als Prozesse kann als Gemeinsamkeit mit anderen Quellen herausgenommen werden. Auch die zweite und fünfte Folgerung lassen sich in Definitionen anderer Quellen wiederfinden. So findet beispielsweise im V-Modell die Verifizierung auf der linken Seite des Vs zwischen den Entwicklungsphasen statt, während die Validierung nach Durchlaufen dieser Phasen auf der rechten Seite des Vs zu finden ist. Die häufig verwendete Abkürzung V&V für die Validierung und Verifizierung zeigt auch, dass diese Prozesse zusammengehören und nicht als Alternativen gesehen werden.

Endres/Rombach, IEEE-Glossar Im Buch *A Handbook for System and Software Engineering* gehen die beiden Autoren Endres und Rombach auf die Validierung und Verifizierung ein (siehe [ER03, S. 98]:

Sie verwenden zunächst eine etwas abgewandelte Form der Fragen von Boehm (siehe Abschnitt 2.5.1), die dann durch Definitionen aus dem IEEE-Glossar ergänzt und als "little more restricted" (ebd.) beschrieben werden. Die hinter der Validierung stehende Aktivität wird zunächst folgendermaßen beschrieben: "determining whether the design fulfills all [functional and non-functional] requirements, assuming the requirements correctly describe what is intended or needed" (ebd.). Bei der Verifizierung wird die Frage "Are we developing the system correctly?" (ebd.) etwas ausführlicher for-

muliert: "is the design an optimal solution for the requirements and is the implementation consistent with the design?" (ebd.). Während Boehm die Beschreibung auf ein System oder Produkt im Allgemeinen bezieht, beschreiben Endres und Rombach die Begriffe im Kontext des System- und Software-Engineerings mit Fokus auf das Ergebnis der Designaktivität. Eine semantisch ähnliche Verwendung der Begriffe ist aber zu erkennen.

Die Definitionen aus dem IEEE-Glossar werden folgendermaßen beschrieben: "validation evaluates a system or component to determine whether it satisfies specified requirements"[14] (ebd.) und "Verfification [...] evaluates a system or a component to determine whether the products of a given development phase satisfy conditions imposed at the start of that phase". Anhand der Beschreibungen lassen sich zwei Unterschiede zu den zuvor von den Autoren verwendeten Beschreibungen herausarbeiten: Zum einen wird bei der Validierung gegen Anforderungen geprüft, bei der Verifizierung allerdings gegen Bedingungen. Zum anderen wird bei der Verifizierung implizit von einem phasenorientierten Entwicklungsmodell ausgegangen, bei dem zu Beginn jeder Phase klar definierte Bedingungen festzulegen sind, die das in der Phase entstehende Produkt erfüllen muss.

INCOSE-Handbuch, ISO 9000 In der Norm Deutsche Institut für Normung (DIN) Europäische Norm (EN) ISO 9000 sind die Grundlagen und Begriffe für Qualitätsmanagement-Systeme festgelegt, die in der ISO 9000-Familie verwendet werden. Die aktuelle Fassung stammt aus dem Jahr 2005 (siehe [ISO05]). Bei der ISO 9000 handelt es sich um eine überarbeitete Fassung der DIN EN ISO 8402. Die Begriffsdefinitionen in der ISO 9000 werden beispielsweise im INCOSE-Handuch verwendet, das unter der Validierung die

[14] Obwohl an dieser Stelle das Wort *requirements* nicht näher beschrieben wird, lässt sich aus dem Kontext ableiten, dass die Verfeinerung "[which] describe what is intended or needed" auch an dieser Stelle gültig ist.

"confirmation, through the provision of objective evidence, that the requirements for a specific intended use or application have been fulfilled" versteht und unter Verifizierung die "confirmation, through the provision of objective evidence, that specified requirements have been fulfilled".

Beim Vergleich mit den Begriffsbeschreibungen von Boehm im Abschnitt 2.5.1 fällt auf, dass die Beschreibung im INCOSE-Handbuch etwas detaillierter ausfällt, beide Quellen aber grundsätzlich semantisch das Gleiche unter den Begriffen verstehen. Bei der Validierung soll bestätigt werden, dass Anforderungen an einen spezifischen Verwendungszweck oder an eine Anwendung (d. h. an die erwünschte Funktionalität des Systems oder an das Produkt (Anmerkung des Autors)) erfüllt werden. Bei der Verifizierung wird die Bestätigung über die Erfüllung spezifizierter Anforderungen gefordert, was als eine abstrakte Beschreibung zu dem im Abschnitt 2.5.1 beschriebenen Implementierungsbeispiel angesehen werden kann. Dabei entspricht die mathematische Berechnungsformel den spezifizierten Anforderungen, die von der Implementierung richtig umgesetzt werden muss.

Grady Grady beschreibt in seinem Buch *System Validation and Verification* die Begriffe Validierung und Verifizierung in zwei verschiedenen Kontexten: System Engineering und Software Development. Im Kontext des System Engineering ist Verifzierung ein "proof process unequivocally showing that a particular design will or does satisfy the corresponding requirements upon which it is based" ([Gra98, S. 3]). Unter Validierung versteht er einen "process to demonstrate that one or more requirements are clearly understood and that it is possible to satisfy them through design work within the current technological state of the art." (ebd.). Wie schon bei [ER03] werden die Begriffe hier mit Fokus auf das Design beschrieben. Allerdings unterscheidet Grady die Begriffe anhand einer zeitlichen Trennung im Entwicklungsablauf.

Den Nachweis, dass das richtige System gebaut wird, nennt er Validierung. Verifizierung ist für ihn "all postdesign V&V work" (ebd.). Damit sieht er die Durchführung der Validierung vor der Verifizierung, was im Gegensatz zu [Hop93] steht.

Im Kontext Software Development beschreibt Grady Verifzierung als ein "process of ensuring that the information developed in phase n of a development process is traceable to information in phase n-1" ([Gra98, S. 4]). Wie schon bei [IEE90] wird hier ein phasenorientiertes Entwicklungsmodell vorausgesetzt. Die Ergebnisse einer Phase n werden mit den Ergebnissen der vorhergehenden Phase n-1, die wiederum als Eingabe für Phase n zu sehen sind, verglichen. Unter der Validierung im Kontext des Software Development versteht Grady den "process of proving compliance with customer requirements". Wird von Details abgesehen, sind diese beiden Begriffsdefinitionen semantisch mit allen anderen aufgeführten Literaturquellen vergleichbar. Grady merkt an, dass die Validierung im Kontext des Software Development essenziell mit der Verifizierung aus dem Bereich des System Engineerings übereinstimmt und verwendet diese Begriffsdefinition in seinem Buch. Diese inverse Deutung des Begriffs wird in [DHJ+10] u. a. als Grund für das uneindeutige Verständnis der Begriffe Validierung und Verifizierung genannt. Das unterschiedliche Verständnis wird aber bereits von Grady erwähnt. Er weist unter anderem darauf hin, dass die Begriffe in anderen Kreisen invers zu seiner Festlegung verwendet werden und empfiehlt eine frühzeitige Klärung der Begriffe bei allen Projektbeteiligten, um Missverständnisse zu vermeiden.

Defence Modeling and Simulation Organization In dem Buch *Verification and Validation in Systems Engineering*[DHJ+10] wird auf die Definitionen der Defence Modeling and Simulation Organization (DMSO) verwiesen, da sie

die "most widely used definitions" ([DHJ+10, S. 75]) sein sollen. Verifizierung ist demnach "the process of determining that a model implementation and its associated data accurately represent the developer's conceptual description and specifications". Hinter Validierung versteckt sich "the process of determining the degree to which a model and its associated data provide an accurate representation of the real world from the perspective of the model's intended use". Es lässt sich erkennen, dass in diesen Definitionen der Fokus auf Modelle gelegt wurde. Semantisch stimmen sie mit Boehm überein und damit auch mit den Quellen, die eine detaillierte Begriffsbeschreibung auf Basis dieser Fragen erstellt haben. Debabbi et al. führen ein Beispiel an, was die Bedeutung der Validierung und Verifizierung über die reine Begriffsdefinition hinaus verdeutlicht:

Falls ein System den Anforderungen der Spezifikation, d. h. den Kundenwünschen, entspricht, aber logische Fehler enthält, dann kann es erfolgreich validiert werden, aber die Verifizierung schlägt fehl. Falls das System keine logischen Fehler enthält, aber sich nicht wie vom Kunden erwartet verhält, dann besteht es die Verifizierung, aber die Validierung schlägt fehl.

Pohl Pohl bezieht seine Definitionen Validierung und Verifizierung auf Anforderungen und schreibt zur Verifizierung: "Die Aufgabe der Verifikation von Anforderungen besteht darin, Aussagen über ein Anforderungsmodell des Systems auf ihren Wahrheitsgehalt hin zu überprüfen." ([Poh08, S. 423]). Für die Validierung führt er eine Definition an, für die an dieser Stelle der erste Teil ausreichend ist: "Validierung bezeichnet die Überprüfung der Eingaben, der Aktivitätsdurchführung und der Ausgaben (Anforderungsartefakte) der Requirements-Engineering-Kernaktivitäten auf die Erfüllung der definierten Qualitätskriterien." (ebd.). An einer anderen Stelle formuliert er dies weniger formell in einem Satz: "Die Validierung von Anforderungen ist

eine Form der analytischen Qualitätssicherung." ([Poh08, S. 422]), wobei er unter der analytischen Qualitätssicherung folgendes versteht: "Die analytische Qualitätssicherung bezeichnet Maßnahmen zur Überprüfung der Qualität von Softwareartefakten im Anschluss an die Fertigstellung eines finalen Entwurfs eines Artefakts. Die Überprüfung eines Datenmodells auf Vollständigkeit und Korrektheit durch ein Validierungsteam ist ein Beispiel für die analytische Qualitätssicherung." (ebd.). Pohl sieht die Validierung nach dem Erstellen eines finalen Entwurfs eines zu prüfenden Artefakts. Die Verifizierung liegt für ihn zeitlich nach der Validierung, da zunächst überprüft werden muss, ob überhaupt die richtigen Anforderungen erfasst wurden. Diese Reihenfolge stimmt mit der von Grady (siehe Abschnitt 2.5.1) überein, wobei die Begriffsdefinitionen nicht vergleichbar sind. Pohl verlangt bei der Validierung von Anforderungen nicht nur die Prüfung der Anforderungsartefakte gegenüber Qualitätskriterien, sondern auch die Kontextbetrachtung und die Aktivitätsdurchführung, da diese Einfluss auf die Vollständigkeit und Fehlerfreiheit der Anforderungen haben.

Rupp Chris Rupp und die SOPHISTen sprechen in [Rup09a] bewusst nicht von Validierung und Verifizierung, sondern von Prüfungen, die allgemein dem Bereich der Qualitätssicherung zuzuordnen sind. Der Begriff wird wie bei Pohl nur in Verbindung mit Anforderungen verwendet. Die Prüfung besteht aus mehreren Teilen, zu denen auch das Ergebnis gehört. Die Prüfung wird vom Prüfer, dem Ziel, dem Prüfgegenstand, der Vorgabe an den Prüfgegenstand, die Prüfmethoden und die Vorgaben zur Prüfung beeinflusst (siehe Abbildung 2.3). Im Bezug auf die Anforderungen wird in [Rup09b] zwischen der konstruktiven und der analytischen Qualitätssicherung zur Prüfung[15] von Anforderungen unterschieden.

[15] Es wird an dieser Stelle von Prüfung gesprochen, obwohl in [Rup09b] in einer Überschrift noch von "Anforderungen validieren" die Rede ist. Das Wort *validieren* wird ansonsten aber nicht weiter verwendet.

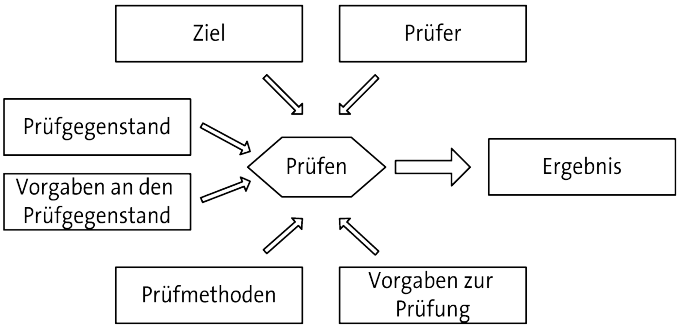


Abbildung 2.3: Einflussfaktoren auf die Aktivität "Prüfen", nach [Rup09a, Seite 97]

Dabei ist unter der konstruktiven Qualitätssicherung zu verstehen, dass Fehler oder Mängel in den Anforderungen bereits während der Erstellung durch die Wahl geeigneter Hilfsmittel vermieden werden. Unter der analytischen Qualitätssicherung wird das Erkennen und Beheben von Fehlern und Mängeln in den Anforderungen nach der Dokumentation und vor der Anforderungsnutzung verstanden. Hier ist ein ähnliches Verständnis wie bei [Poh08] (siehe Abschnitt 2.5.1) zu erkennen, der die Anforderungsvalidierung als eine Form der analytischen Qualitätssicherung sieht. Die konstruktive Qualitätssicherung ist bei Pohl in der Überprüfung der Anforderungserstellung wiederzufinden.

2.5.2 Verwendung in dieser Arbeit

Wie gezeigt wurde, gibt es in der Literatur keine eindeutige Definition für die Begriffe Validierung und Verifizierung. Zwar lässt sich eine grundsätzlich ähnliche Interpretation der Begriffe in den untersuchten Quellen wiederfinden, dennoch gibt es immer wieder kleine und größere Interpretati-

onsspielräume. Um diesen Spielraum für die vorliegende Arbeit und den darin beschriebenen Ansatz zu begrenzen, wird jeweils eine Definition für die beiden Begriffe festgelegt. Darüber hinaus erfolgt im nachfolgenden Kapitel eine Definition des Begriffs *Validierungsprozess*, um den anvisierten Fokus des in der Arbeit vorgestellten Ansatzes zu schärfen und Abgrenzungen und Vergleiche mit anderen Arbeiten (siehe Kapitel 5) zu erleichtern.

In der vorliegenden Arbeit werden sowohl die Validierung als auch die Verifizierung als Prozesse verstanden, die eine Konformitätsprüfung eines Betrachtungsgegenstandes[16] gegen spezifizierte Vorgaben durchführen. Als Hauptunterscheidungspunkt wird die Intention der Prüfung gesehen.

> Validierung ist die Feststellung der Konformität eines Betrachtungsgegenstandes gegenüber den vom Kunden spezifizierten Anforderungen. Es wird geprüft, ob der Betrachtungsgegenstand den Vorstellungen des Kunden entspricht.

> Verifizierung vergleicht das Ergebnis einer Entwicklungsphase mit den für die Phase spezifizierten Anforderungen. Es wird geprüft, ob bei der Entwicklung des Betrachtungsgegenstandes Fehler gemacht wurden.

Als Beispiel für die Validierung kann die Abnahme eines entwickelten Betrachtungsgegenstandes durch den Kunden genannt werden. Dies findet häufig am Ende einer Entwicklung statt und ist auch unter dem Begriff *Abnahmetest* bekannt. Ein gutes Beispiel für die Verifizierung ist in der Softwareentwicklung zu finden. Während der Implementierungsphase wird

[16] Bei dem Betrachtungsgegenstand kann es sich um das ganze System oder nur um einzelne System-Komponenten handeln.

die zuvor erarbeitete Software-Architektur-Spezifikation in Quellcode umgesetzt. Diese Umsetzung kann auch als Transformation bezeichnet werden. Die Überprüfung, ob der Quellcode auch wirklich der spezifizierten Software-Architektur entspricht, also die Transformation von der Software-Architektur-Spezifikation zum Quellcode fehlerfrei verlaufen ist, entspricht einer Verifizierung.

Bei der Konformitätsprüfung einer System-Architektur kann es vorkommen, dass die beiden Prüfungsarten miteinander vermischt werden. Das hat zwei Ursachen: Zum einen ist der Kunde häufig an der Beschreibung der Vorgaben für die Design-Phase beteiligt. Neben einem indirekten Einfluss auf die Architektur durch die gewünschten Funktionalitäten, kann er mit Hilfe von NFAs auch konkrete Vorgaben formulieren. Dies reicht von beispielsweise zeitlichen Vorgaben über die Wartbarkeit bis hin zu konkreten Vorgaben für Hardware- oder Software-Komponenten. Zum anderen ist die Entwicklung einer Architektur eine kreative Tätigkeit [MR09], die von Menschen durchgeführt wird. Da Menschen Fehler machen, ist vor der weiteren Verwendung eine Überprüfung des Resultates sinnvoll. In beiden Fällen wird das Resultat der Designphase, die Architektur, gegenüber den Anforderungen geprüft. Es findet also sowohl eine Validierung als auch eine Verifizierung statt. Nur mit Hilfe der Prüfungsintention lässt sich eine genauere Unterscheidung treffen.

Zur Festlegung eines Begriffs für die Prüfung der System-Architektur gegenüber den Anforderungen hält sich die vorliegende Arbeit an die Interpretation von Grady [Gra98]. Im Kontext der Systementwicklung findet die Validierung zeitlich vor der Verifizierung statt. Dabei wird geprüft, ob das richtige System gebaut wird. Verifizierung findet erst nach der Erstellung

der Architektur statt ("The author [...] refers to all postdesign V&V work as verification" [Gra98, Seite 3]).

Trotz der unterschiedlichen Intentionen sind die beiden Prozesse doch sehr eng miteinander verbunden und werden häufig zusammen genannt. In der theoretischen Annahme, dass

- eine vollständige Menge an Anforderungen ermittelt werden könnte,
- durch eine Validierung nachgewiesen werden könnte, dass diese genau den Wünschen des Kunden entsprächen und
- die Korrektheit der Transformationen in der Entwicklung durch Verifizierungen sichergestellt werden könnte,

wären am Ende der Entwicklung keine Abnahmetests erforderlich. Das Produkt würde genau den Wünschen des Kunden entsprechen. Da allerdings die Ermittlung einer vollständigen Anforderungsmenge bereits illusorisch ist und auch eine vollständige Validierung der Anforderungen gerade bei einer natürlichsprachlichen Dokumentation nicht möglich ist, bleibt die Notwendigkeit bestehen, Zwischenresultate der Systementwicklung, insbesondere die System-Architektur als Fundament des Systems, zu validieren.

2.6 Validierungsprozess

Nach der Definition der Begriffe Validierung und Verifizierung sowie den dazugehörigen Erläuterungen im vorherigen Kapitel wird im Folgenden der Begriff für den in der Arbeit vorgestellte Ansatz definiert. Diese Definition erleichtert die in Kapitel 5 vorgenommenen Abgrenzungen zu anderen Ar-

beiten, indem sie zum einen die Intention des Ansatzes, die Validierung, und zum anderen die für den Ansatz bevorzugte Prüftechnik festlegt.

> Der Validierungsprozess für System-Architekturen besteht aus mehreren Prüfverfahren, die während und zum Abschluss der Design-Phase die Übereinstimmung der System-Architektur mit den architekturrelevanten Anforderungen des Kunden ermitteln. Für die Prüfverfahren wird die Verwendung von Simulationen als Vertreter von dynamischen Prüftechniken bevorzugt.

Details zu den verschiedenen Arten von Prüftechniken sind im nachfolgenden Kapitel beschrieben.

2.7 Validierungs- und Verifizierungsmethoden

Für die Durchführung von Validierungen oder Verifizierungen können unterschiedliche Prüftechniken eingesetzt werden. Nach Debabbi et al. [DHJ+10, Kapitel 5.1] lassen sich diese grundsätzlich in vier Kategorien einordnen:

Informell Diese Techniken basieren auf menschlicher, subjektiver Interpretation. Sie werden zwar strukturiert angewandt, beruhen aber nicht auf mathematischen Formalismen. Im Vergleich mit den anderen Kategorien sind sie arbeitsintensiv und ineffizient. Beispiele sind Reviews und Inspektionen.

Statisch Diese Techniken werden für Auswertungen des Modells und der Implementierung eingesetzt. Sie überprüfen die Einhaltung von Modellie-

rungsrichtlinien oder Traceability-Verbindungen und können syntaktische, eingeschränkt auch semantische Modellierungsfehler erkennen. Eine erfolgreiche Überprüfung hat ein konsistentes Modell zur Folge, was eine Voraussetzung für den Einsatz dynamischer Prüftechniken ist. Beispiele sind Datenfluss-, Kontrollfluss oder Schnittstellenanalysen sowie die Auswertung von Traceability-Verbindungen.

Dynamisch Diese Techniken werten das Verhalten des modellierten Systems aus. Wird zusätzlicher Code in das Modell eingefügt, lässt sich die Durchführung im Modell nachverfolgen. Beispiele sind das Debuggen, Functional und Executional Testing sowie Simulationen.

Formal Diese Techniken basieren auf einem mathematischen Modell, wodurch sie mit mathematischen Schlussfolgerungen und Beweisen arbeiten können. Sie werden vor allem für sicherheitskritische Systeme eingesetzt, bei denen das Versagen schwerwiegende Konsequenzen nach sich zieht. Beispiele für formale Prüftechniken sind das Model-Checking und Theorem-Proving.

2.8 Modell und Simulation

In der vorliegenden Arbeit wird der Begriff Simulation wie folgt definiert:

> "Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Insbesondere werden die Prozesse über die Zeit entwickelt."[Ver10, Blatt 1, Seite 3]

In der Verein Deutscher Ingenieure (VDI)-Norm werden weiterhin die Begriffe Modell, Simulationsexperiment und Simulationslauf definiert:

> "Ein Modell ist eine vereinfachte Nachbildung eines geplanten oder existierenden Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmen vom Vorbild."[Ver10, Blatt 1, Seite 3]

> "Ein Simulationsexperiment ist die gezielte empirische Untersuchung des Verhalten eines Modells durch wiederholte Simulationsläufe mit systematischer Parameter- oder Strukturvariation."[Ver10, Blatt 1, Seite 3f]

> "Ein Simulationslauf ist die Nachbildung des Verhaltens eines Systems mit einem spezifizierten ablauffähigem Modell über einen bestimmten (Modell-) Zeitraum, auch Simulationszeit genannt, wobei gleichzeitig die Werte untersuchungsrelevanter Zustandsgrößen erfasst und gegebenenfalls statistisch ausgewertet werden."[Ver10, Blatt 1, Seite 4]

Durch die letzten beiden Begriffe sollen einige Eigenschaften der Simulation hervorgehoben werden, die für das Verständnis des Simulationsbegriffs hilfreich sind. Gerade der letzte Satzteil der Simulationslauf-Definition, die Erfassung und Auswertung von Werten während des Simulationslaufs, ist für die Abgrenzung zu den formalen Verifikationsmethoden von Bedeutung. Formale Verifikationsmethoden können keine Simulationszwischen-

werte zur Verfügung stellen. Für die Problemanalyse des Architekten nach einer fehlgeschlagenen Validierung sind diese aber hilfreich.

2.9 Modellgetriebene Validierung

Dieses Kapitel erläutert die Verwendung des Begriffs *modellgetrieben* im Zusammenhang mit den zur Validierung eingesetzten Simulationen[17] und damit indirekt auch für die Validierung der System-Architektur. Es geht auf die Probleme der modellgetriebenen Entwicklung in der praktischen Anwendung ein, wie diese zu vermeiden sind und welche Vorteile eine ordentliche Anwendung der modellgetriebenen Validierung für den Architekten mit sich bringt.

Das Modell dient in dem in der Arbeit vorgestellten Validierungsprozess nicht nur Dokumentationszwecken, sondern auch als Datenquelle für die zur Validierung eingesetzten Simulationen[18]. Die Daten können über die grafische Schnittstelle eines UML-Modellierungswerkzeugs benutzerfreundlich und kontextbezogen[19] eingegeben werden. Änderungen am Modell wirken sich auf die Ergebnisse der Simulationen aus oder anders ausgedrückt: Die Simmulationsergebnisse sind getrieben durch die modellierten Daten.

Die Erfahrungen in der praktischen Anwendung zeigen allerdings, dass oft nur der erste Schritt in Richtung modellgetriebene Entwicklung gegangen

[17] An dieser Stelle wird der Begriff Simulation verwendet, obwohl auch andere Prüftechniken zur Validierung eingesetzt werden könnten. Mit der Vorstellung des Validierungsprozesses in Kapitel 4 wird der etwas allgemeingültigere Begriff *Validierungszielverfahren* eingeführt.

[18] Es müssen keinesfalls alle für die Simulationen erforderlichen Daten im Modell enthalten sein.

[19] Mit *kontextbezogen* ist hier die Eingabe der Daten über eine geeigneten Architektursicht in Form eines Diagramms gemeint.

wird[20]. Es wird eine modellbasierte Dokumentation eingesetzt, die darin enthaltenen Daten werden aber entweder überhaupt nicht genutzt[21] oder sie müssen manuell auf die Zielplattform, hier die Simulationen, transformiert werden. Modelländerungen beeinflussen damit nicht zwingend die Simulationsergebnisse. Zudem lassen sich die Eingabedaten der Simulationen auch ohne Änderungen am Modell beeinflussen, wodurch eine Inkonsistenz zwischen den modellierten Daten und den Simulationsergebnisen entstehen kann. Damit die Validierung wirklich durch die im Modell enthaltenen Daten *getrieben* wird, muss das UML-Modell die einzig mögliche Datenquelle für diese Daten sein.

Die Inkonsistenz wird im Projektalltag akzeptiert, weil Änderungen einen doppelten Arbeitsaufwand (Änderung des Modells und der Eingabedaten der Simulationen) zur Folge haben. Da es sich bei der Entwicklung einer Architektur um einen inkrementellen Prozess [RH09, Kapitel 4] [Sta11] handelt, ist von einer hohen Änderungsrate für die Architektur und damit für das UML-Modell auszugehen. Bereits für die Erarbeitung eines ersten, gegenüber den Anforderungen validen Architekturentwurfs sind häufig mehrere Iterationen erforderlich. Auch nach der Erstellung eines validen Architekturentwurfs finden Änderungen an den Anforderungen oder der Architektur statt. Dies ist insbesondere bei den Entwicklungsmodellen der Fall, bei denen das System-Design und die Anforderungserfassung zeitlich parallel stattfinden (z. B. agile Vorgehensweisen, Rational Unified Process (RUP) oder COSMOD-RE[22]). Mit einer Verdopplung des Arbeitsaufwandes bei Än-

[20] Diese Aussage bezieht sich sowohl auf die Entwicklung von Systemen als auch auf die Entwicklung von Software.

[21] UML ist dann häufig nur ein aufwändiges Malwerkzeug zur Erstellung von Übersichtsbildern.

[22] sCenario and gOal based SysteM delOpment methoD-Requirements Engineering, Details zu COSMOD-RE sind in Kapitel 5.2 beschrieben.

derungen an Architektur oder Anforderungen lässt sich eine modellgetriebene Validierung im normalen Projektalltag nicht etablieren.

Ein neues Verfahren lässt sich hingegen gut etablieren, wenn eine Effizienzsteigerung (benötigter Aufwand gegenüber der erzielten Qualität) für die Beteiligten, hier den Architekt, erreicht wird. Dies ist bei der modellgetriebenen Entwicklung möglich, wenn die modellierten Daten für weitere Aufgaben im Entwicklungsprozess genutzt werden, so dass die Beteiligten mindestens um den erforderlichen Mehraufwand zur Einführung des modellgetriebenen Ansatzes entlastet werden. Dies lässt sich bereits durch die Automatisierung des Auslesens der Modelldaten erreichen. Änderungen müssen dann nur noch im UML-Modell durchgeführt werden, dessen Bearbeitung über eine grafische Oberfläche möglich ist. Dadurch wird auch die Inkonsistenz zwischen modellierten Daten und den Validierungsergebnissen vermieden.

Ein weiterer Vorteil des modellgetriebenen Ansatzes für den Architekten ergibt sich bei genauerer Betrachtung der Datennutzung in den Simulationen. Mit Hilfe der fachlichen und strukturellen Daten aus dem UML-Modell lässt sich ein generischer Simulator in eine domänenspezifische Simulation transformieren. Dadurch ist es dem Architekten möglich, verschiedene System-Architekturen im UML-Modell zu entwerfen, zu dokumentieren und unmittelbar, ohne Mehraufwand für die Datentransformation oder die Anpassung der Simulation, gegenüber den Anforderungen zu validieren. In praktischen Tests (siehe Kapitel 10.1) war zu beobachten, dass dadurch die Hemmschwelle, eine alternative, valide Architektur zu finden, deutlich gesunken ist. Auf diese Weise wird der Architekt nicht nur bei der Validierung einer Architektur, sondern auch bei der Erarbeitung von Architekturalternativen unterstützt. Eine Auswahl an validen Architekturen ist eine Voraus-

setzung, um diese anhand von Kriterien zu bewerten und die am besten geeignete auszuwählen. Die Bewertung von Architekturen gehört allerdings nicht mehr zu den Aufgaben des in der vorliegenden Arbeit vorgestellten Ansatzes (siehe Kapitel 5.3.4).

3 Einführung in die Domäne Radar-Systeme

Vor der Definition des Validierungsprozesses wird in diesem Kapitel eine grundlegende Einführung in den Bereich der Radar-Systeme gegeben. Das Radar-System repräsentiert die Domäne der eingebetteten Systeme und dient bei der Prozessdefinition in Kapitel 4 als praktisches Beispiel zum besseren Verständnis der einzelnen Prozessschritte. Hierzu beschreibt Kapitel 3.1 zunächst die grundlegende Funktionsweise eines Radars, anschließend geht Kapitel 3.2 etwas näher auf die Verarbeitung der empfangenen Signale ein. Die Besonderheiten bei der Verarbeitung von Datenströmen werden in Anhang A.2 beschrieben. Eine detaillierte Beschreibung der Anwendung des Validierungsprozesses auf Radar-Systeme inklusive der Simulationsmöglichkeiten nimmt Kapitel 6 vor.

3.1 Funktionsweise eines Radars

Der Begriff Radar ist eine Abkürzung für RAdio Detection And Ranging[23], also der funkgestützten Ortung von Objekten und der Ermittlung von deren Abstand zum Sendeort. Radar-Systeme lassen sich je nach Einsatzgebiet und Funktionsweise in unterschiedliche Bereiche einteilen. In der vorliegenden Arbeit liegt der Fokus auf einem aktiven bildgebenden Primärradar-System, das mit elektro-magnetischen Signalimpulsen arbeitet. Für eine ausführliche Beschreibung wird auf Ludloff [Lud08] verwiesen.

[23] Die ursprüngliche Abkürzung lautet Radar Aircraft Detecting And Ranging. Der deutsche Fachbegriff hierfür lautet Funkmesstechnik. Dieser wurde allerdings in Westdeutschland nach dem zweiten Weltkrieg und in Ostdeutschland nach der Wiedervereinigung durch Radar ersetzt.

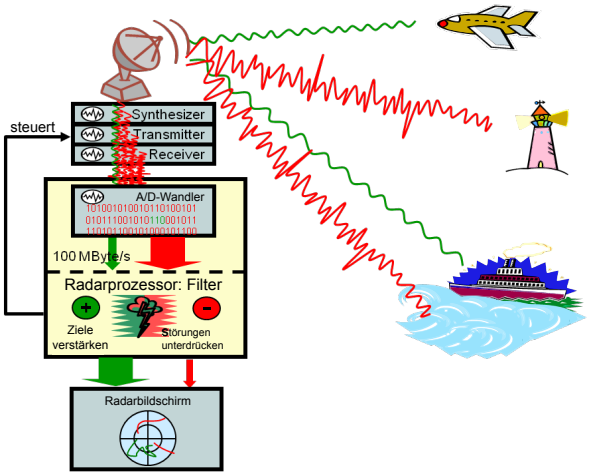


Abbildung 3.1: Schematische Darstellung der Funktionsweise eines aktiven bildgebenden Primärradarsystems

Abbildung 3.1 zeigt schematisch die Funktionsweise eines aktiven, bildgebenden Primärradargeräts. Die von einem *Funksynthesizer* erzeugten elektro-magnetischen Wellen[24] werden vom *Transmitter* als Impuls ausgestrahlt. Nach dem Aussenden *lauscht* der *Receiver* eine bestimmte Zeit auf die Signalreflexionen (Echo) der realen Objekte, bevor es den nächsten Sendeimpuls absendet. Die vom Receiver empfangenen Echoimpulse werden zur digitalen Verarbeitung mit Hilfe eines *Analog-/Digital-Wandlers* in ein digitales Signal konvertiert[25]. Der sogenannte Radarprozessor, das Herzstück des Radar-Systems, filtert die in der Abbildung rot dargestellten Störungssignale heraus und verstärkt die gewünschten Ziele (grün). Die zu verarbeitende Datenmenge kann bei modernen Radar-Systemen bei etwa 100 MiByte pro Se-

[24] Die für die Funkmessung verwendeten Frequenzen befinden sich im Kilo- bis Gigahertz-Bereich. Durch den Einsatz der unterschiedlichen Frequenzen sind Messungen mit unterschiedlicher Genauigkeit möglich.

[25] Ob der Analog-/Digital-Wandler Bestandteil des Radarprozessors oder des Radar-Systems selbst ist, hängt von den physikalischen Komponenten des Radar-Systems ab.

kunde liegen. Die so gefilterten Signale werden auf einem Radarbildschirm als Ziele (engl. *tracks*) dargestellt.

Neben der aktuellen Position der Objekte können auch zusätzliche Informationen wie die Geschwindigkeit, eine Objektcharakterisierung sowie die Darstellung der vergangenen Objektpositionen angezeigt werden. Welches Objekt als Störung oder als Ziel eingestuft wird, hängt vom Einsatzgebiet des Radars ab. Während ein Schwarm Vögel für ein Wetterradar eine geringe Relevanz hat, ist diese Information für ein Rundsichtradar an einem Flughafen für das sichere Starten und Landen von Flugzeugen von großer Bedeutung.

Die Verarbeitung der Radarsignale erfolgt in Echtzeit, was hohe Anforderungen an die Leistungsfähigkeit der eingesetzten Hardware stellt. Hinzu kommen hohe Anforderungen an das verwendete Material und die Beschaffenheit des Systems selbst, so dass der Betrieb beispielsweise in einer materialunfreundlichen Umgebung (Salzwasser, Meeresluft) und in einem möglichst breiten Temperaturbereich (-30°C bis +50°C) störungsfrei und zum Teil über mehrere Monate ohne größere Wartungsmöglichkeiten möglich ist.

3.2 Signalverarbeitung

Abbildung 3.2 zeigt einen vereinfachten Ablauf der Signalverarbeitung eines Radar-Systems mit Hilfe eines UML-Aktivitätsdiagramms. Die Verarbeitung beginnt, sobald Signale von einer Antenne empfangen und an den Radarprozessor weitergegeben werden. Im ersten Schritt, *Beams verarbeiten*, werden

die Echoimpulse bezüglich des Elevationswinkels[26] in sogenannte Keulen (engl. *beams*) zusammengefasst und anschließend den einzelnen Sweeps[27] zugeordnet (siehe Aktion *Sweeps verarbeiten*).

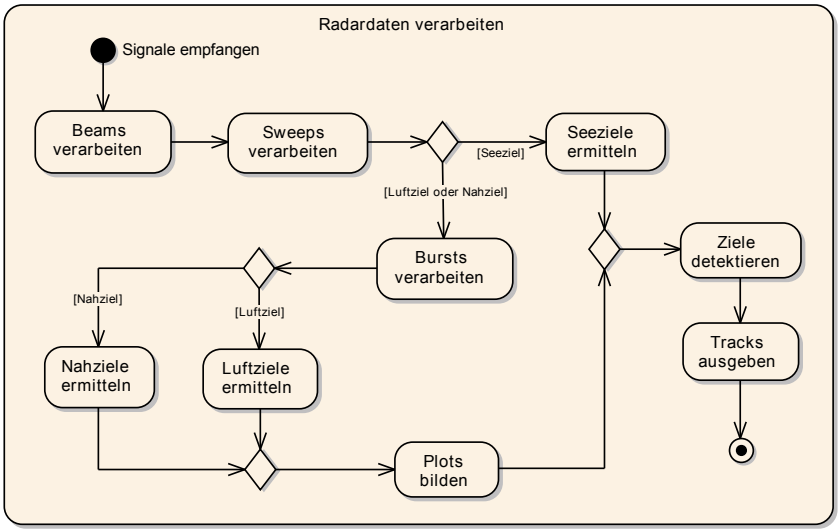


Abbildung 3.2: Vereinfachter Ablauf der Signalverarbeitung eines Radar-Systems

Die Ermittlung der empfangenen Treffer[28] (engl. *hits*) erfolgt in Abhängigkeit von der verwendeten Signalcharakteristik, die für die verschiedenen Zielarten, hier Luft-, Nah- und Seeziele, verwendet werden (siehe Aktionen

[26] Der Elevationswinkel, auch Höhenwinkel genannt, beschreibt die Richtung in der Vertikalen. Bei 0 Grad erfolgt eine waagerechte Abstrahlung und bei 90 Grad eine Abstrahlung senkrecht nach oben.

[27] Bei einem Rundsichtradar entspricht der Sweep einer Umdrehung der Antenne. Ein Sweep ist also der Zeitraum, in dem der abzutastende Zielraum einmal vollständig mit Sendeimpulsen bestrahlt wurde.

[28] Ein Treffer entspricht einem empfangenen Echoimpuls, der eine bestimmte Signalstärke überschritten hat. Durch die Signalgrenze werden Störungen, beispielsweise das Rauschen der physikalischen Bauteile, herausgefiltert. Ein Objekt der realen Welt erzeugt abhängig von der reflektierenden Fläche mehrere Treffer.

Seeziele ermitteln, *Nahziele ermitteln* und *Luftziele ermitteln*). Für Luft- und Nahziele werden die Sweeps vorher zu sogenannten Bursts zusammengefasst (siehe Aktion *Bursts verarbeiten*). Die Aktion *Plots bilden* ermittelt, welche Luft- und Nahzielhits zu einem Objekt der realen Welt gehören und fasst diese Hits entsprechend zu einem Plot zusammen. Bei den Seezielen ist dieser Verarbeitungsschritt in der Aktion *Seeziele verarbeiten* bereits enthalten. Die Aktion *Ziele detektieren* filtert anhand von verschiedenen Eigenschaftswerten aus den Plots die vom Anwender des Radarsystems gewünschten Ziele (engl. tracks). Im letzten Schritt, *Tracks ausgeben*, werden die ermittelten Tracks über eine Schnittstelle an eine externe Komponente, z. B. den Radarbildschirm, weitergeben, der diese mit weiteren Radardaten (z. B. die Umgebung, Koordinaten, usw.) grafisch darstellt.

Bei der beschriebenen Signalverarbeitung handelt es sich um eine vereinfachte Beschreibung. Der Fokus liegt auf dem grundlegenden Ablauf, bei dem aus einer von der Antenne kommenden sehr großen Datenmenge schrittweise unerwünschte Informationen[29] herausgefiltert werden bis am Ende die gewünschten Ziele übrig bleiben. Bezogen auf eine Zielart ergibt sich eine lineare Verarbeitungskette. In einem realen Radarprozessor ist die Verarbeitungsreihenfolge allerdings deutlich komplexer, da häufig Informationen aus späteren Verarbeitungsschritten von früheren benötigt bzw. verwendet[30] werden. In der Signalverarbeitung werden auch Funktionen wie z. B. die Protokollierung von Daten oder die Konfiguration des Systems nicht betrachtet.

[29] Dies können beispielsweise Regentropfen oder Nebelschwaden sein.

[30] Ein Beispiel ist die kontinuierlich stattfindende Kalibrierung bei beweglichen Radarsystemen.

4 Definition des Validierungsprozesses

Dieses Kapitel definiert den Prozess zur Validierung von System-Architekturen gegenüber den architekturrelevanten System-Anforderungen. Als Einführung betrachtet Kapitel 4.1 den Ansatz zunächst aus der Vogelperspektive, ordnet ihn grob in die Umgebung der Systementwicklung ein und stellt die wesentlichen Merkmale heraus. Kapitel 4.2 nennt die notwendigen Voraussetzungen bevor Kapitel 4.3 zunächst den gesamten Prozess vorstellt und die einzelnen Prozessschritte jeweils in einem eigenen Unterkapitel erläutert. Als unterstützendes Beispiel dient dabei das in Kapitel 3 eingeführte Radar-System. Eine detaillierte Anwendung des Prozesses auf die Domäne datenintensiver eingebetteter Systeme ist in Kapitel 6 beschrieben. Mit der Möglichkeit zur Übertragung des Validierungsprozesses auf eine andere Domäne beschäftigt sich Kapitel 10.

4.1 Vogelperspektive

Abbildung 4.1 zeigt eine schematische Sicht auf die Umgebung des Validierungsprozesses. Die *System-Architektur* stellt anhand der Ergebnisse des *System-Designs* und den *Anforderungen*[31] einen *Architektur-Entwurf* bereit. Die Ergebnisse des System-Designs fließen wieder in die System-Analyse mit ein, wodurch eine iterative Entwicklung des Systems ermöglicht wird. Die *System-Architektur-Validierung* unterstützt den System-Architekten bei der Erstellung und Weiterentwicklung des Architektur-Entwurfs. Hierzu werden neben dem Entwurf selbst die architekturspezifischen Anforderun-

[31] Hier sind vor allem die architekturbeeinflussenden nicht-funktionalen Anforderungen gemeint.

gen[32] sowie *validierungsspezifische Informationen* benötigt. Die architekturspezifische Sicht für den Architekten ist eine eigene Sicht auf die Anforderungen, die durch die Zuordnung von Anforderungen und architekturspezifischen Aspekten entsteht. Der Validierungsprozess weist den Architekten auf *verletzte Anforderungen* hin und stellt ihm mit den *Validierungsdetails* Daten zur Verfügung, mit denen er Architekturanpassungen vornehmen kann, um eine valide System-Architektur[33] zu erhalten.

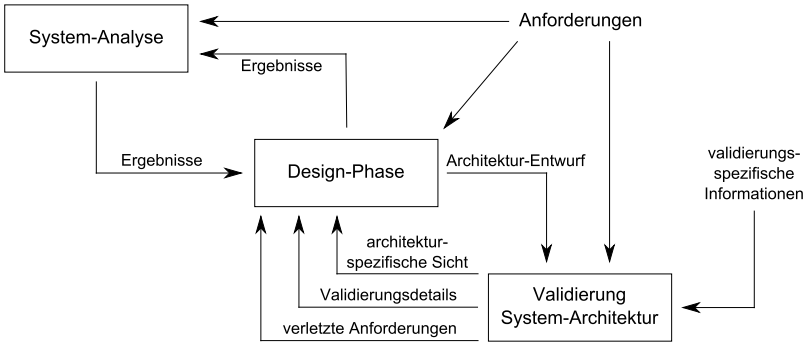


Abbildung 4.1: Schematische Sicht auf den Validierungsprozess

Abbildung 4.1 nimmt nur eine grobe Einordnung des Validierungsprozesses in die Phasen der Systementwicklung vor. Hauptsächlich dient sie als Überblick, welche Daten von wem konsumiert und bereitgestellt werden. Eine detaillierte Einordnung nimmt Kapitel 5.2 vor.

Für den Architekten ist der Prozess ein Hilfsmittel zur Validierung der Architektur gegenüber den Anforderungen. Er lässt sich in den iterativen

[32] Hierbei handelt es sich um nicht-funktionalen Anforderungen; die funktionalen Anforderungen haben nur indirekt Einfluss auf die Architektur.

[33] Eine System-Architektur ist valide, wenn sie alle architekturrelevanten Anforderungen erfüllt.

Prozess der Architekturentwicklung integrieren und ermöglicht sowohl eine punktuelle als auch eine vollständige Validierung. Damit der Architekt bei der stetig steigenden Anzahl an Anforderungen und deren Verbindungen untereinander den Überblick behält, bietet der Prozess dem Architekten eine architekturspezifische Sicht auf die Anforderungen. Sie erlaubt es dem Architekten, Zusammenhänge zwischen den verschiedenen architekturspezifischen Aspekten[34] zu identifizieren und so bei der Entwicklung des System-Designs zu berücksichtigen. Durch die Aspekte sind die Anforderungen indirekt mit der Architektur verbunden, wodurch der Validierungsprozess den Architekten bei der Analyse der Auswirkungen (engl. *impact analysis*) von Veränderungen an der Architektur auf die Anforderungen und umgekehrt unterstützen kann.

Die Kernideen des in der vorliegenden Arbeit beschriebenen Ansatz zur Architekturvalidierung sind zunächst unabhängig von einer bestimmten Modellierungssprache. Für die praktische Anwendung ist allerdings die Festlegung auf eine Sprache notwendig. Die Wahl fällt hierbei auf die UML, da sie in der Industrie häufig für eine modellbasierte Dokumentation eingesetzt wird und deshalb den Status eines Quasi-Standards genießt. An dieser Stelle muss zwischen der Verwendung der UML als Modellierungssprache für den Validierungsprozess und für die Dokumentation des zu entwickelnden Systems unterschieden werden. Die Verwendung derselben Modellierungssprache erleichtert zwar die Anwendung des Ansatzes, die Wahl der Systemdokumentationssprache kann aber unabhängig von der für den Ansatz verwendeten Modellierungssprache erfolgen (siehe Kapitel 8.2). Auf Basis der verwendeten Modellierungssprache lassen sich verschiedene Ansätze für Werkzeuge erarbeiten, die den Architekten bei der Durchführung des Vali-

[34] Die Zusammenhänge zwischen den Aspekten ergeben sich durch die zugeordneten Anforderungen und deren Verbindung untereinander.

dierungsprozesses unterstützen (Unterstützungsmöglichkeiten). Die prototypische Implementierung ausgesuchter Ansätze als Nachweis der Realisierbarkeit bildet die unterste Ebene des Validierungsansatzes (Werkzeugprototypen). Diese vier Ebenen des in der Arbeit beschriebenen Ansatzes sind in Abbildung 4.2 dargestellt. Dabei sinkt der Abstraktionsgrad von oben nach unten.

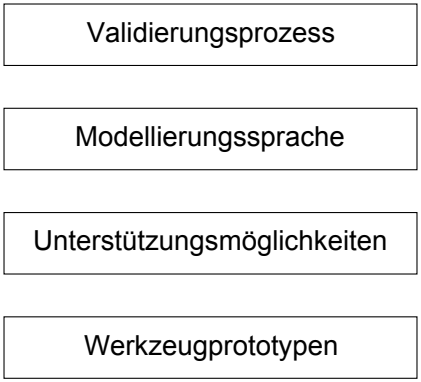


Abbildung 4.2: Abstraktionsgrad des Ansatzes zur Architekturvalidierung

4.2 Voraussetzungen

Es gibt grundsätzlich zwei Voraussetzungen, die für eine sinnvolle Durchführung des Validierungsprozesses vorhanden sein müssen. Dabei handelt es sich zum einen um die Existenz von System-Anforderungen und zum anderen um einen Entwurf der Architektur. Sowohl Anforderungen als auch Architektur müssen für den Einsatz des Prozesses nicht vollständig sein. Eine hohe Anforderungsqualität ist wünschenswert, aber nicht essentiell.

Die Vollständigkeit der Anforderungen wäre eine schwierig zu testende Voraussetzung. Sie muss allerdings einem Grad entsprechen, der einen (partiellen) Architekturentwurf und dessen Validierung ermöglicht. Dieser Grad lässt sich allerdings nicht quantifizieren, so dass die Entscheidung, ob die erfassten Anforderungen für einen Architekturentwurf ausreichend sind, am Ende von einem Menschen getroffen werden muss. In Abhängigkeit von den Projektrandbedingungen können unterschiedliche Methoden eingesetzt werden, um fehlende Anforderungen zu finden. Als Beispiele seien hier der Einsatz verschiedener Ermittlungstechniken[35], die Erhöhung der Anforderungsqualität (beispielsweise die Präzisierung nicht-eindeutigen Inhalts) oder auch systematische Ansätze zum Finden von funktionalen und nicht-funktionalen Anforderungen (beispielsweise die Use Case-Analyse, die strukturierte Analyse oder der NFA-Prozess nach Dörr) genannt. Letztendlich kann das Fehlen von Anforderungen aber nicht ausgeschlossen werden. Es lassen sich z. B. keine Anforderungen von einem Stakeholder erheben, der bei der Ermittlung nicht berücksichtigt wurde.

Für die Entwicklung einer Architektur empfiehlt es sich, Anforderungen mit einer gewissen Grundqualität zu verwenden. In der Praxis hat sich dabei bewährt, die Anforderungen vor der Architekturentwicklung bzgl. einiger ausgesuchter Qualitätskriterien[36] (z. B. eindeutig, korrekt, konsistent, vollständig und testbar) zu überprüfen. Dabei werden grobe Fehler beseitigt, die Iterationen im Architekturentwurf zur Folge hätten. Bei der Erstellung des Entwurfs selbst treten durch den Fokus auf die Architektur weitere Unklarheiten und ggf. fehlende Anforderungen zu Tage. Durch die zeitlich parallele Durchführung der Phasen Anforderungsermittlung (als Teil der System-

[35] Rupp [Rup09b] stellt hierfür einen guten Überblick bereit.

[36] Qualitätskriterien für Anforderungen sind im IEEE-Standard 830 [IEE98], bei den Robertsons [RR06] oder bei Rupp [Rup09b] zu finden.

Analyse) und Erstellung der System-Architektur lassen sich Synergieeffekte nutzen, die bei einer streng sequentiellen Durchführung nicht auftreten würden[37]. Auf diese Weise lässt sich die Architektur auf einem soliden Fundament errichten.

Für den Validierungsprozess ist es erforderlich, dass die architekturrelevanten Anforderungen validiert werden können. Dies ist möglich, falls die Anforderung testbar ist oder mindestens ein Abnahmekriterium besitzt. Eine testbare Anforderung hat die Eigenschaft, dass der von ihr geforderte Inhalt überprüft werden kann. Ein Beispiel hierfür ist Bereitstellung eines Ergebnisses nach maximal 3ms. Es gibt allerdings auch Anforderungen, deren geforderter Inhalt nicht zur Zeit der Systemabnahme bzw. mit einem vertretbarem Aufwand überprüft werden kann. Ein Beispiel hierfür ist die 24x7-Verfügbarkeit eines Systems. Zur Abnahme dieser nicht-testbaren Anforderungen werden Abnahmekriterien definiert. Ein mögliches Abnahmekriterium für die 24x7 Verfügbarkeit eines Systems wäre der durchgängige Betrieb des Systems über einen bestimmten Zeitraum und mit bestimmten Testdaten. Sowohl bei den testbaren als auch bei den nicht-testbaren Anforderungen existieren letztendlich Prüfkriterien bestehend aus einem Vergleichswert und einem Vergleichsoperator, mit deren Hilfe die Anforderung als erfüllt deklariert werden kann. Die Festlegung von Prüfkriterien ist teilweise vor dem Beginn der Design-Phase möglich, für den Validierungsprozess allerdings nicht essentiell. Falls kein Verfahren zur Prüfung der Anforderung vorgegeben ist, muss der Architekt bei der Durchführung des Validierungsprozesses ein geeignetes Verfahren festlegen.

[37] Weitere Informationen zur Einordnung des Validierungsprozesses in die Systementwicklung sind in Kapitel 5.2 beschrieben.

Eine weitere Voraussetzung ist die Verwendung einer modellbasierten Dokumentation des Systems. Mit ihr werden möglichst viele der für die Entwicklung des Architekturentwurfs benötigten Informationen dokumentiert. Hierbei handelt es sich unserer Erfahrung nach um die verschiedenen Sichten der Architektur [Kru95], die System-Komponenten[38], deren Schnittstellen untereinander und eine Zuordnung der Software-Komponenten zu den Hardware-Komponenten. Konkrete Vorgaben zum Inhalt der Dokumentation macht der Validierungsprozess allerdings nicht. Dieser kann wie die Semantik der Modellelemente projektspezifisch festgelegt werden (siehe Kapitel 8.2). Der in der Arbeit vorgestellte Validierungsprozess verwendet die Modellierungssprache UML. Es sind zwar grundsätzlich andere Modellierungssprachen möglich, die Einschränkung auf eine UML-basierte Entwicklungsdokumentation erleichtert allerdings das Verständnis des Prozesses.

Zusammengefasst ergeben sich die folgenden Voraussetzungen für den Validierungsprozess:

- System-Anforderungen und ein Entwurf der System-Architektur sind vorhanden.
- Die System-Anforderungen müssen mindestens ein Prüfkriterium besitzen.
- Zur Dokumentation sollte eine modellbasierte Dokumentationsform, vorzugsweise die UML, eingesetzt werden.
- Die System-Anforderungen können unvollständig sein, sie müssen aber die Erstellung eines Architekturentwurfs ermöglichen.
- Eine hohe Anforderungsqualität ist wünschenswert.

[38] Basierend auf der Definition in Kapitel 2.1 besteht ein System aus Software und Hardware. Eine System-Komponente kann demnach eine Software-Komponente oder eine Hardware-Komponente sein.

4.3 Prozessdefinition

Abbildung 4.3 zeigt den Ablauf des Validierungsprozesses mit Hilfe eines Aktivitätsdiagramms. Zum besseren Verständnis werden zunächst der Prozess als Ganzes sowie die Zusammenhänge zwischen den verschiedenen Begriffen beschrieben. Die Unterkapitel gehen dann detailliert auf die einzelnen Prozessschritte ein. Auf die Wiederverwendungsmöglichkeiten der VSN und der Validierungszielverfahren geht Kapitel 8.1 näher ein.

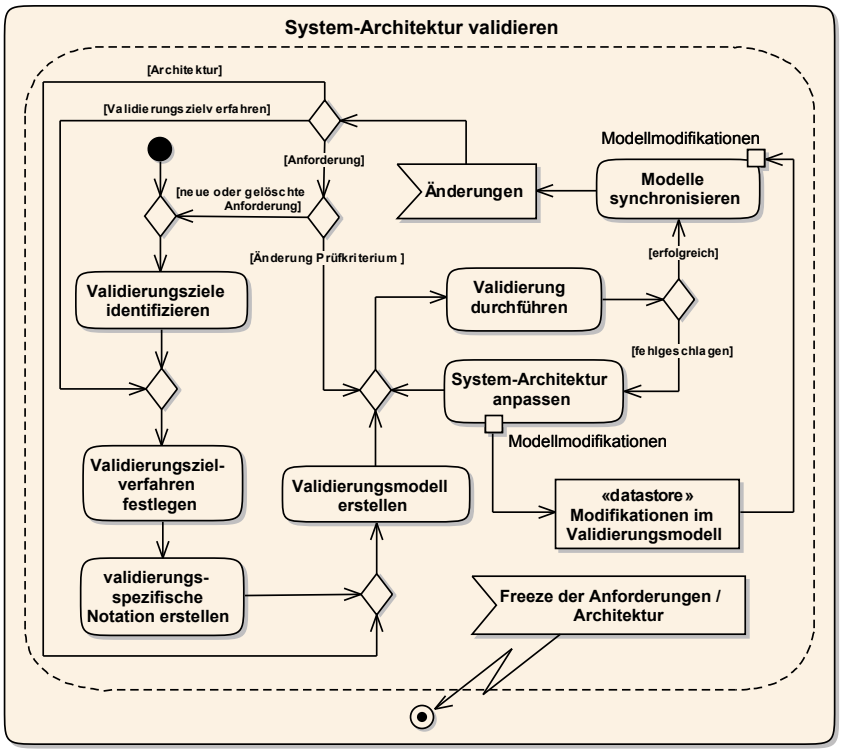


Abbildung 4.3: Ablauf des Validierungsprozesses als Aktivitätsdiagramm

Zu Beginn identifiziert der Architekt die Validierungsziele aus den System-Anforderungen. Dabei entspricht ein Validierungsziel einem architekturspezifischen Aspekt, den der Architekt prüfen möchte. Aus Gründen der Nachverfolgbarkeit werden die Validierungsziele mit den dazugehörigen Anforderungen verbunden. Die Überprüfung der Validierungsziele erfolgt mit Hilfe von Validierungszielverfahren. Der in der Arbeit vorgestellte Ansatz bevorzugt den Einsatz von Simulationen als Validierungszielverfahren.

Sobald diese Verfahren festgelegt sind, kann der Architekt mit der Erstellung der VSN beginnen. Die validierungsspezifische Notation (VSN) erlaubt es dem Architekten, alle für die Validierungszielverfahren benötigten Daten zu modellieren. Dabei werden die validierungsspezifischen Daten getrennt von den entwicklungsspezifischen Daten dokumentiert. Neben dem Entwicklungsmodell entsteht das Validierungsmodell. Das Validierungsmodell dient zum einen als architekturvalidierungsspezifische Sicht auf das System und zum anderen als Datenquelle für die Validierungszielverfahren. Falls alle Validierungsziele identifiziert, deren Verfahren festgelegt und die dafür benötigten Informationen modelliert sind, kann der Architekt mit der Architekturvalidierung beginnen. Die Validierung ist erfolgreich, falls jedes Validierungsziel erfolgreich validiert wurde. Schlägt die Validierung eines Zieles fehl, verbessert der Architekt die Architektur im Validierungsmodell und wiederholt die Validierung der Architektur. Die dabei vorgenommenen Änderungen werden protokolliert. Falls eine valide Architektur vorliegt, überprüft der Architekt, ob die protokollierten Modellmodifikationen für das Entwicklungsmodell relevant sind und überträgt diese gegebenenfalls. An dieser Stelle im Prozess liegt nun eine valide Architektur vor. Werden Änderungen an den Anforderungen, der Architektur oder an den Validierungszielverfahren vorgenommen, ist eine Revalidierung erforderlich. Kommen neue Anforderungen hinzu oder werden vorhandene gelöscht, müssen alle

Schritte noch einmal durchlaufen werden. Hat sich nur das Prüfkriterium geändert, kann direkt mit der Validierung fortgefahren werden. Nach Änderungen an Validierungzielverfahren müssen die Auswirkungen auf andere Verfahren sowie auf die validierungsspezifische Notation und das Modell überprüft werden. Haben sich Änderungen an der Architektur ergeben, müssen diese in das Validierungsmodell übertragen werden, um dann anschließend die Validierung mit den aktualisierten Informationen durchzuführen. Der Validierungsprozess wird in dem Moment[39] unterbrochen, wenn die Anforderungen oder die Architektur durch den Projektfortschritt als nicht mehr veränderbar deklariert werden (engl. *requirements/architecture freeze*).

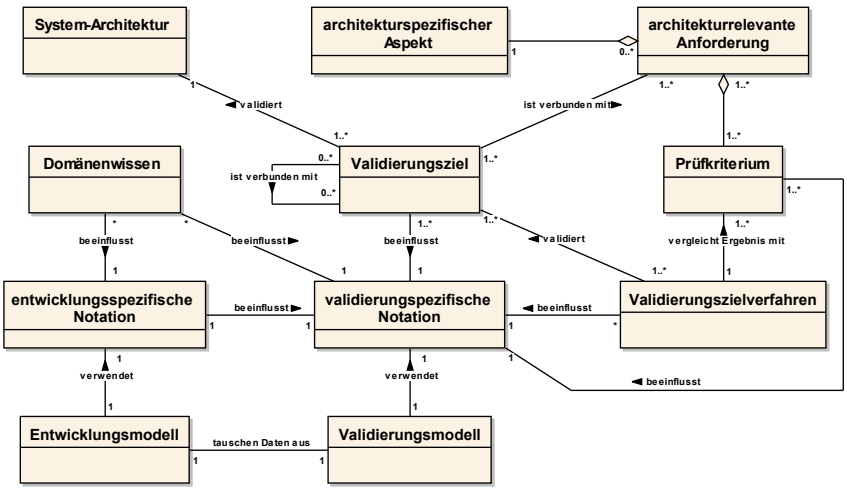


Abbildung 4.4: Begriffsmodell für den Validierungsprozess

[39] Diese Aussage ist zeitlich nicht wortwörtlich zu nehmen. Sinnvollerweise erfolgt ein solcher Schritt nur, wenn eine valide Architektur vorhanden ist und nicht während der Architekt noch Aktivitäten durchgeführt werden.

Das in Abbildung 4.4 mit Hilfe eines Klassendiagramms dargestellte Begriffsmodell[40] zeigt die im Kontext des Prozesses verwendeten Begrifflichkeiten und deren Beziehungen untereinander. Im Mittelpunkt stehen die Validierungsziele, mit deren Hilfe die System-Architektur validiert wird. Ein Validierungsziel wird wiederum durch mindestens ein Validierungszielverfahren validiert. Die Validierungsziele sind mit den architekturrelevanten Anforderungen und ggf. untereinander verbunden. Eine architekturrelevante Anforderung kann einem architekturspezifischen Aspekt zugeordnet werden und enthält ein Prüfkriterium, das für den Vergleich mit den Ergebnissen des Validierungszielverfahrens verwendet wird. Zur Modellierung der validierungsspezifischen Daten im Validierungsmodell wird eine validierungsspezifische Notation verwendet. Diese wird von den Validierungszielen, den Validierungszielverfahren, den Prüfkriterien, der entwicklungsspezifischen Notation und dem Domänenwissen beeinflusst. Letzteres beeinflusst auch die entwicklungsspezifische Notation, die als Basis für das Entwicklungsmodell dient. Zwischen Entwicklungs- und Validierungsmodell findet ein Datenaustausch statt.

4.3.1 Validierungsziele identifizieren

Im ersten Schritt des Validierungsprozesses ist es die Aufgabe des Architekten, die Validierungsziele anhand der System-Anforderungen zu identifizieren.

> Ein Validierungsziel fasst alle Anforderungen eines architekturspezifischen Aspekts zusammen und enthält keine Prüfkriterien.

[40] Weitere Informationen zum Begriffsmodell sind in [Rup09b] zu finden.

Wie Anforderungen die Architektur beeinflussen und was hinter dem architekturspezifischen Aspekt steht, lässt sich anhand der groben Unterscheidung von Anforderungen gemäß der Begriffsdefinition in Kapitel 2.4.2 erklären.

[Rup09b] unterscheidet auf der obersten Ebene zwischen funktionalen und nicht-funktionalen Anforderungen. Die funktionalen Anforderungen geben den Funktionsumfang des Systems vor. Bezogen auf die Architektur beeinflussen sie vor allem die Schnittstellen der einzelnen System-Komponenten und die Datenstrukturen. Handelt es sich um qualitativ hochwertige Anforderungen, wie dies in Kapitel 4.2 gefordert wurde, dann machen sie keine Aussagen, *wie* die Realisierung der Anforderungen aussehen soll. Der Architekt könnte ein beliebiges Design aus einer sehr großen Menge an möglichen Designs wählen. Falls der Kunde aber Einfluss auf die Wahl des Designs nehmen möchte, beispielsweise weil er nicht eine Minute auf das Ergebnis einer Addition zweier ganzzahliger Zahlen warten möchte, dann muss er mit Hilfe der nicht-funktionalen Anforderungen Einschränkungen vorgeben. Für das genannte Additionsbeispiel könnte die Einschränkung lauten: Das System muss das Ergebnis einer Vektoraddition nach spätestens einer Mikrosekunde anzeigen. Nicht-funktionale Anforderungen werden auch als *Architekturtreiber* [TU 13] bezeichnet. Sie reduzieren die Menge der möglichen Designs [Rom85] [BB02], wodurch es für den Architekten schwieriger wird, ein zu den Anforderungen valides Design zu finden.

Um den validierungsspezifischen Aspekt in der Definition von Validierungsziel zu erläutern, ist ein Blick auf die nicht-funktionalen Anforderungen notwendig. NFAs lassen sich in Kategorien unterteilen. Es gibt in der Literatur allerdings keine einheitliche Vorstellung über die Anzahl, Benennung und Unterscheidung der verschiedenen Kategorien. Einige Beispiele sind im

ISO/IEC[41]-Standard 9126 [ISO01] bzw. dessen Nachfolger 25010 [ISO11], im Volere-Ansatz [RR06] oder im IVENA[42]-Ansatz [Rup09b] zu finden. Aus den praktischen Erfahrungen heraus wird empfohlen, sich für einen vorhandenen Kategorienkatalog zu entscheiden oder die zu verwendenden Kategorien projektspezifisch festzulegen. Letzteres setzt Dörr [Dör10] in seinem Ansatz zur Erhebung einer vollständigen Menge nicht-funktionaler Anforderungen ein (siehe Kapitel 2.4.3 und 5.2.5). Dabei entsteht ein sogenanntes Qualitätsmodell, das baumartig aufgebaut ist. Die Baumstruktur setzt die Kategorien[43] der unterschiedlichen Abstraktionsstufen vertikal miteinander in Beziehung. Für die horizontalen Abhängigkeiten verwendet Dörr eine sogenannte Abhängigkeitsmatrix. Ein mögliches Qualitätsmodell für die Kategorie Effizienz ist in Abbildung 4.5 zu sehen. Die hellgrau hinterlegten Kategorien dienen allein der Strukturierung. Nur die Blätter des Baumes, die dunkelgrau hinterlegten Kategorien, werden zur Charakterisierung der nicht-funktionalen Anforderungen verwendet. Ob eine Kategorie ein Blatt oder ein innerer Knoten ist, hängt davon ab, ob der Kategorie mindestens eine spezifische Prüfvorschrift zugewiesen werden kann.

41 International Electrotechnical Commission

42 Integriertes Vorgehen zur Erhebung nichtfunktionaler Anforderungen

43 In [Dör10] wird an Stelle von Kategorie der Begriff Qualitätsattribut verwendet.

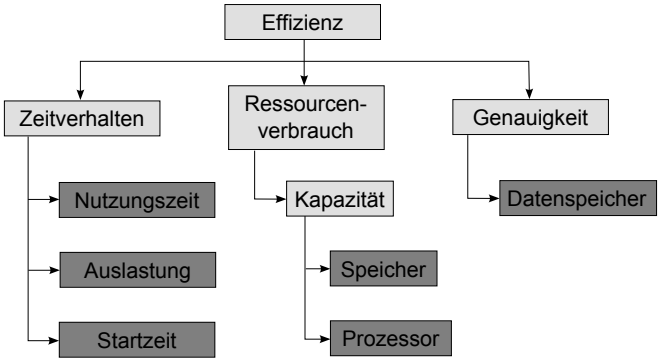


Abbildung 4.5: Auszug aus einem Qualitätsmodell für die Kategorie *Effizienz*

Was nun genau mit dem architekturspezifischen Aspekt in der Definition von Validierungsziel gemeint ist, lässt sich gut anhand des Qualitätsmodells erläutern. Ein architekturspezifischer Aspekt ist eine überprüfbare Kategorie nicht-funktionaler Anforderungen. Im Kontext des Qualitätsmodells sind damit alle Blätter des Baumes gemeint. Die Eigenschaft der Überprüfbarkeit ist für den Validierungsprozess von großer Bedeutung, da die System-Architektur als Ganzes über die Gesamtheit der Validierungsziele validiert wird. Falls ein Validierungsziel nicht überprüft werden kann, weil kein entsprechendes Verfahren existiert, dann ließe sich die Architektur nicht vollständig validieren.

Ein Validierungsziel repräsentiert demnach eine Menge von nicht-funktionalen Anforderungen eines architekturspezifischen Aspekts. Die Gesamtheit der Validierungsziele bildet damit eine Abstraktionsebene über den Anforderungen, die dem Architekten eine architekturspezifische Sicht auf die Anforderungen erlaubt (siehe Abbildung 4.6). Da ein Validierungs-

ziel mehrere Anforderungen zusammenfasst und die Anzahl an Kategorien nicht-funktionaler Anforderungen in der Regel deutlich geringer ist als die Anzahl der Anforderungen insgesamt[44], geht der Überblick auch bei einer hohen Anzahl an Anforderungen nicht verloren. Über die zugeordneten Anforderungen lassen sich automatisiert direkte und indirekte Abhängigkeiten zwischen den Validierungszielen ermitteln.

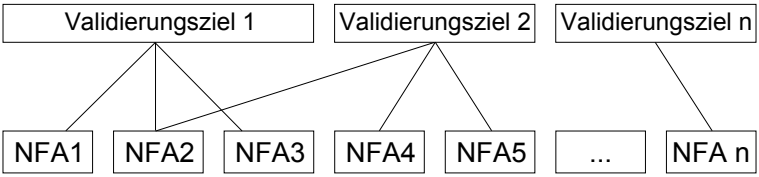


Abbildung 4.6: Schematische Darstellung der architekturspezifischen Sicht

Ein für die Wartung wichtiger Punkt steckt im letzten Teil der Begriffsdefinition von Validierungsziel auf Seite 73: Ein Validierungsziel enthält keine Prüfkriterien. Nach den Voraussetzungen in Kapitel 4.2 muss jede System-Anforderung aber mindestens ein Prüfkriterium besitzen, wobei ein Prüfkriterium aus einem Vergleichsoperator und einem Vergleichswert besteht. Die Prüfkriterien stellen sicher, dass mit dem Ergebnis der Validierungszielverfahren (siehe Kapitel 4.3.2) entschieden werden kann, ob die Architektur die Validierungsziele erfüllt. Die Prüfkriterien eines Validierungsziels ergeben sich implizit über die zugeordneten System-Anforderungen[45]. Sofern

[44] Bezogen auf ein Qualitätsmodell ist die Anzahl der Kategorien sogar begrenzt.

[45] Bei der praktischen Ermittlung der Prüfkriterien ist es entscheidend, wie die Anforderungen vorliegen. Aus natürlichsprachlichen Anforderungen können die Prüfkriterien nur durch einen Menschen ermittelt werden. Mit einem steigenden Grad der Formalisierung, von Satzschablonen [Rup09b] bis zur Controlled Native Language (CNL) [KK06], lässt sich dies auch maschinell erledigen. Je nach Dokumentationsform müssen dem Validierungsziel also ein oder mehrere Prüfkriterien (manuell) zugeordnet werden. Bei Veränderungen an den Anforderungen ist eine Anpassung erforderlich. Das Validierungsziel selbst bleibt aber unverändert.

sich diese nicht über ein gemeinsames Validierungszielverfahren überprüfen lassen, sind auch mehrere Verfahren für ein Validierungsziel denkbar. Durch die Trennung von Prüfkriterien und Validierungszielen führt die Veränderung eines Prüfkriteriums[46], was gerade in den frühen Projektphasen häufiger vorkommt, nicht zu einer Änderung des oder der zugeordneten Validierungsziele. Im weiteren Verlauf der Arbeit wird vereinfachend von den Prüfkriterien der Validierungsziele gesprochen, wobei damit die Prüfkriterien der dem Validierungsziel zugeordneten Anforderungen gemeint sind.

Beispiel

Das Identifizieren eines Validierungszieles wird anhand eines Beispiels verdeutlicht. Für das Radar-System sind unter anderem die folgenden Anforderungen erhoben worden:

- Das System muss die Möglichkeit bieten digitalisierte Signale von der Antenne zu empfangen. (A1)
- Das System muss die Möglichkeit bieten Tracks auszugeben. (A2)
- Das System muss für die Verarbeitung der digitalisierten Radarsignale bis zur Ausgabe der Tracks höchstens 310ms benötigen. (A3)
- Das System muss aus den See-, Luft- und Nahziel-Plots Tracks ermitteln. (A4)
- Das System muss höchstens 220W verbrauchen. (A5)

Für die Validierungsziele sind die nicht-funktionalen Anforderungen von Bedeutung. Bei den Anforderungen A1, A2 und A4 handelt es sich um funktionale Anforderungen, A3 und A5 sind nicht-funktionale Anforderungen.

[46] Beispielsweise die Reduzierung der Antwortzeit für eine Addition zweier ganzzahliger Zahlen auf unter 0,1 Sekunden.

Aus diesen ergeben sich die beiden Validierungsziele: Gesamtverarbeitungszeit (V1) und Energieverbrauch (V2).

A3 gibt eine zeitliche Einschränkung für die Berechnung der Tracks vor. Das Ziel der Validierung ist die Überprüfung der Verarbeitungszeit, genauer der *Gesamtverarbeitungszeit*, da die Tracks das gewünschte Datenobjekt eines Radar-Systems sind. Das Prüfkriterium setzt sich aus dem numerischen Wert *310ms* und dem logischen Vergleichsoperator *kleinergleich* ($\leq$) zusammen. Zur Nachverfolgbarkeit verbindet der Architekt Anforderung A3 mit dem Validierungsziel V1. Die zweite nicht-funktionale Anforderung ist A5. Das Validierungsziel ist hier die Überprüfung des *Energieverbrauchs*. Das Prüfkriterium setzt sich aus dem numerischen Wert *220W* und dem logischen Vergleichsoperator *kleinergleich* ($\leq$) zusammen. Anforderungen A5 wird mit dem Validierungsziel V2 verbunden.

4.3.2 Validierungszielverfahren festlegen

Nachdem alle zu betrachtenden Validierungsziele der System-Anforderungen festgelegt wurden, muss der Architekt ein Verfahren zur Prüfung der verschiedenen architekturspezifischen Aspekte erarbeiten. Da jedes Validierungsziel einen unterschiedlichen Aspekt betrachtet, muss der Architekt für jedes Validierungsziel ein eigenes Verfahren festlegen. Im Kontext des Validierungsprozesses handelt es sich um ein *Validierungszielverfahren*. Mit Hilfe der Verfahrensergebnisse ist eine Prüfung der Validierungsziele gegenüber den Prüfkriterien der zugeordneten Anforderungen möglich. Der Architekt kann als Verfahren eine informelle, statische, dynamische oder formale Prüftechnik (siehe Kapitel 2.8) auswählen, wobei der Validierungsprozess eine Simulation (dynamische Prüftechnik) bevorzugt.

Die Wahl des Validierungszielverfahrens ist eine anspruchsvolle Aufgabe für den Architekten. Die Auswahl ist unter anderem vom Entwicklungsstand des Systems und den zur Verfügung stehenden Daten abhängig. Der Validierungsprozess kann bereits für die Überprüfung des ersten Design-Entwurfs eingesetzt werden, also zu einem frühen Zeitpunkt in der Systementwicklung. Gerade bei langlaufenden Projekten existiert zu diesem Zeitpunkt aber nur ein grobes Verständnis über das zu entwickelnde Gesamtsystem, das sich mit fortschreitendem Entwicklungsverlauf verbessert. Der Wissensstand kann auch sehr differenziert sein. Während die Entwickler über einige interne System-Funktionalitäten detailliertes Wissen besitzen, treten bei deren Zusammenspiel zur Bereitstellung der Gesamtfunktionalität Wissenslücken auf. Diese Lücken spiegeln sich in den nicht-funktionalen Anforderungen als *noch zu definierende* (engl. *to be defined (TBD)*) oder *noch zu überprüfende*[47] (engl. to be checked (TBC)) Prüfkriterien wieder. Da die Systementwicklung nicht einem strikten Wasserfall-Modell folgt, sondern die Entwicklung von Anforderungen und Design quasi parallel verlaufen (siehe COSMOD-RE) in Kapitel 5.2.3), können einige Prüfkriterien erst zu einem späteren Zeitpunkt festgelegt werden.

Der Validierungsprozess berücksichtigt die fortlaufende Weiterentwicklung des Wissens über das System, indem die Validierungsziele unabhängig von Prüfkriterien definiert werden. In Kapitel 4.2 wird zwar die Testbarkeit der Anforderungen vorausgesetzt[48], da dieser Zustand in der Praxis aber meist erst mit fortschreitender Entwicklung erreicht wird, lässt sich der Prozess ohne Mehraufwand in das iterative Vorgehen bei der Designerstellung inte-

[47] Hierbei handelt es sich meist um eine pessimistische Schätzung der numerischen Werte, mit dem Ziel, die Suche der Entwickler nach Realisierungsmöglichkeiten möglichst nicht einzuschränken.

[48] Nur so können die dazugehörigen Validierungsziele und damit die Architektur als Ganzes validiert werden.

grieren. Änderungen an den Prüfkriterien ziehen eine Revalidierung nach sich, nicht aber die Neudefinition des Validierungsziels. Durch den Einsatz des Validierungsprozesses werden implizit TBDs vermieden. Die Anzahl an TBCs steigt zwar an, aber damit existieren zumindest Prüfkriterien, die eine erste Architekturvalidierung ermöglichen. Die Validierungsresultate unterstützen den Architekten dann bei der Festlegung realistischer Werte[49]. Unseren Erfahrungen nach werden zu Beginn eines Projektes relativ simple Validierungszielverfahren eingesetzt, die dann im Laufe des Projektfortschrittes den zur Verfügung stehenden Informationen angepasst oder gegen detailliertere ausgetauscht werden. An dieser Stelle kann ein erarbeitetes Qualitätsmodell nach [Dör10] hilfreich sein, um verschiedene Validierungszielverfahren für eine Anforderungskategorie zu dokumentieren.

Beispiel

Für die beiden Validierungsziele sind in den zugeordneten Anforderungen keine Testverfahren beschrieben. Der Architekt muss sich also ein geeignetes Verfahren überlegen. Die Gesamtverarbeitungszeit wird über die Verarbeitungszeit der einzelnen Software-Komponenten des Systems berechnet. Diese ergibt sich durch die Komplexität (repräsentiert durch Floating Point Operations (Flops)) der Software-Komponente *SW* geteilt durch die Verarbeitungsleistung (in MFlops pro Sekunde) der ausführenden Hardware-Komponente *HW*. Da eine Hardware-Komponente mehrere Software-Komponenten verarbeiten kann, muss dies bei der für die jeweilige Software-Komponente zur Verfügung stehenden Verarbeitungsleistung berücksichtigt werden (siehe Formel 4.1). Vereinfacht wird angenommen, dass sich die Leistung der Hardware-Komponente über den Kehrwert der

[49] Der Grad der Unterstützung hängt natürlich vom Detaillierungslevel der Validierungszielverfahren ab.

zugeordneten Software-Komponenten ergibt. Für beispielsweise zwei Komponenten stehen demnach jeweils die Hälfte der Verarbeitungsleistung der Hardware-Komponente zur Verfügung. Die Flops sind dem Architekten aus früheren Projekten, empirischen Tests oder Schätzungen der Entwickler bekannt.

$$\sum_{i=1}^{n} \frac{complexity_{SW_i}}{performance_{HW_{SW_i}}} \tag{4.1}$$

Für die Berechnung des Energieverbrauchs (V2) werden zunächst die größten Verbraucher, die Verarbeitungseinheiten, vom Architekten berücksichtigt. Der Energieverbrauch ergibt sich über einen Grundverbrauchswert und einen dynamischen, von der Auslastung der Verarbeitungseinheiten abhängigen Wert (siehe Formel 4.2 bis 4.4).

$$\sum_{i=1}^{n}(basicConsumption_{HW_i} + processorLoad_{HW_i} * dynamicValue_{HW_i}) \tag{4.2}$$

$$processorLoad_{HW} = \frac{activeExecutionTime_{HW}}{totalExecutionTime} \tag{4.3}$$

$$dynamicValue_{HW} = maximalConsumption_{HW} - basicConsumption_{HW} \tag{4.4}$$

Den Grund- und den Maximalverbrauchswert entnimmt der Architekt den technischen Datenblättern der Verarbeitungseinheit. Die Auslastung muss abhängig von der gewählten Architektur, d. h. der Zuordnung der Software- zu den Hardware-Komponenten, ermittelt werden. Hier wird die aktive Verarbeitungszeit im Verhältnis zur Gesamtverarbeitungszeit gesetzt (siehe Formel 4.3). Beide Werte lassen sich aus dem Validierungszielverfahren von V1 ermitteln, wobei die aktive Verarbeitungszeit eher ein Nebenprodukt dieses Verfahrens ist. V2 hängt damit direkt von V1 ab. Zur Nachverfolgbarkeit verbindet der Architekt die beiden Validierungsziele. Auf diese Weise

kann er bei Änderungen an einem der Validierungsziele erkennen, dass die Änderungen Auswirkungen auf das andere Validierungsziel haben.

4.3.3 Validierungsspezifische Notation erstellen

Mit der Festlegung der Validierungszielverfahren ist dem Architekten bekannt, welche Eingangsdaten die verschiedenen fachlichen Algorithmen/Testverfahren benötigen. Um eine automatisierte Ausführung des Verfahrens zu ermöglichen, müssen die Eingangsdaten in digitalisierter und einlesbarer Form vorliegen. Der Validierungsprozess setzt hierbei auf die Modellierung dieser Daten in einem UML-Modell[50]. Während sich einige Daten ggf. über die von der UML definierten Modellelemente dokumentieren lassen, muss der Architekt zur Modellierung der übrigen Daten die UML erweitern. Diese Erweiterung der UML-Standardnotation[51] zur Modellierung von Daten für die Architekturvalidierung erfolgt durch die validierungsspezifische Notation. Sie definiert durch welche Modellelemente die validierungsspezifischen Daten im UML-Modell repräsentiert werden.

Bei den verschiedenen Notationsmöglichkeiten muss grundsätzlich zwischen einer einheitlichen oder getrennten Notation für die validierungs- und entwicklungsspezifischen Daten sowie zwischen der Verwendung der UML-Metamodellerweiterung oder des Profilmechanismus für die Modellierung der validierungsspezifischen Informationen unterschieden werden. Eine ausführliche Beschreibung der Möglichkeiten inklusive der Vor- und Nachteile ist in Kapitel 8.2 zu finden.

[50] Genau genommen verwendet der Validierungsprozess zwei UML-Modelle. Dies hat aber auf die Auswahl der validierungsspezifischen Notation keinen Einfluss.

[51] Die Standardnotation der UML hat die Object Management Group (OMG) durch die Infrastructure [Obj11b] und die Superstructure [Obj11a] definiert.

Der Prozess setzt auf getrennte Notationen für die entwicklungs- und validierungsspezifischen Daten, wobei die Definition der validierungsspezifischen Notation in Abhängigkeit von der vorhandenen entwicklungsspezifischen Notation erfolgt. Die Erstellung der validierungsspezifischen Notation erfolgt mit Hilfe des Profilmechanismus. Hauptgründe für die Wahl dieser Notationsmöglichkeit sind der niedrige Erstellungs- und Modellierungsaufwand für den Architekten, die gute Unterstützung des Profilmechanismus durch die Modellierungswerkzeuge sowie die Möglichkeit, den Prozess in laufende Projekte einzusetzen, ohne die vorhandene entwicklungsspezifische Notation zu beeinflussen.

Beispiel

Aus den Validierungszielverfahren von V1 und V2 erarbeitet der Architekt das in Abbildung 4.7 dargestellte UML-Profil. Es enthält die zwei Stereotypen *software* und *hardware*, mit denen domänenspezifische Daten an UML-Komponenten modelliert werden können.

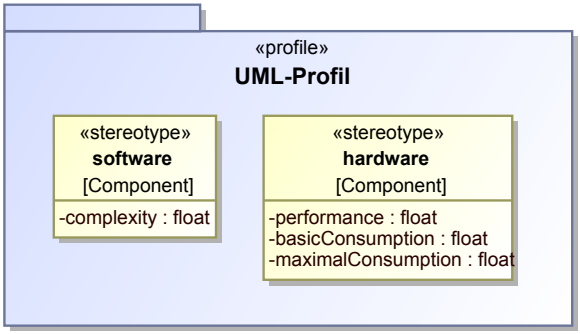


Abbildung 4.7: Beispiel für eine validierungsspezifische Notation

Der Stereotyp *software* hat den TaggedValue *complexity* für die benötigten MFlops der Software-Komponente. Der *hardware*-Stereotyp besitzt die TaggedValues *performance*, *basicConsumption* und *maximalConsumption* für die Verarbeitungsleistung in MFlops pro Sekunde, den Grundenergieverbrauch und den maximalen Energieverbrauch der Hardware-Komponente.

4.3.4 Validierungsmodell erstellen

Nach der Festlegung der Validierungszielverfahren und der validierungsspezifischen Notation[52] modelliert der Architekt im nächsten Prozessschritt alle für die Validierung benötigten Informationen. Wie bereits im Aktivitätsdiagramm des Validierungsprozesses in Abbildung 4.3 zu erkennen ist, wird in der vorliegenden Arbeit eine getrennte Modellierung der Entwicklungs- und Validierungsdaten bevorzugt. Entscheidend hierfür ist die Unabhängigkeit des Validierungsprozesses von der Entwicklungsdokumentation. Sie sollte nur entwicklungsspezifische Informationen enthalten, um die Modellgröße so gering wie möglich zu halten. Dies ist vor allem für den Wartungsaufwand des Modells von Bedeutung. Die Datentrennung hält das Entwicklungsmodell unabhängig vom Validierungsprozess, so dass auch andere Ansätze ohne Rücksicht auf den Validierungsprozess das Entwicklungsmodell als Datenbasis verwenden können.

Eine ausführliche Beschreibung und Diskussion der Möglichkeiten, Modellierung der validierungsspezifischen und entwicklungsspezifischen Daten in einem oder in voneinander getrennten Modellen zu verwenden, ist in Kapitel 8.3 zu finden.

[52] Für den Begriff Notation wird zur besseren Lesbarkeit die Einzahl verwendet, auch wenn es potenziell mehrere Notationen sein können. Falls die Anzahl von Bedeutung ist, wird dies im Text deutlich gemacht.

Beispiel

Ein nicht unerheblicher Anteil der für die Validierung erforderlichen Informationen sind bereits im Entwicklungsmodell[53] enthalten und können in das Validierungsmodell übernommen werden. Dies kann per Hand oder mit Unterstützung eines Werkzeugs geschehen. Ein Prototyp für die Unterstützung des Architekten bei der Transformation der Daten vom Entwicklungs- in das Validierungsmodell ist in Kapitel 7.1.4 beschrieben.

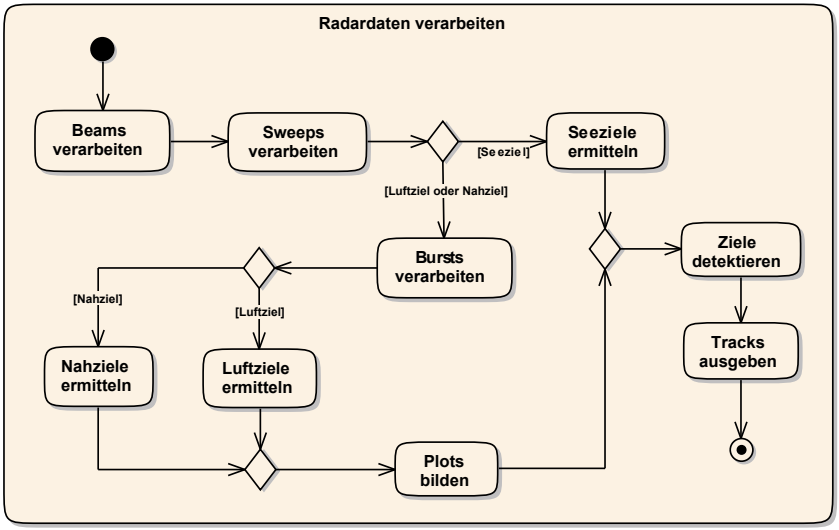


Abbildung 4.8: Vereinfachter Ablauf der Radarsignalverarbeitung

Die übrigen, für die Validierung benötigten Informationen, die nicht im Entwicklungsmodell enthalten sind, muss der Architekt per Hand oder automatisiert aus einer anderen Datenquelle (siehe Kapitel 7.3.5) ergänzen. Die Ab-

[53] Gemäß den in Kapitel 4.2 festgelegten Voraussetzungen für den Validierungsprozess wird eine modellbasierte Dokumentation verwendet.

bildungen 4.8, 4.9 und 4.10 zeigen Diagramme aus dem Entwicklungsmodell. Abbildung 4.8 zeigt den Ablauf der Radardatenverarbeitung mit Hilfe eines Aktivitätsdiagramms. In Abhängigkeit von der Zielart existieren drei Ablaufmöglichkeiten, an deren Ende immer die Ausgabe von Tracks steht. Eine ausführliche Beschreibung der Signalverarbeitung ist in Kapitel 3.2 enthalten. Abbildung 4.9 zeigt die Software-Sicht des Radarprozessors, der die Signalverarbeitung durchführt. Das betrachtete System *Radarprozessor* besteht aus neun Software-Komponenten, deren Abhängigkeit untereinander durch die modellierten Ports und Schnittstellen sowie den Abhängigkeitsbeziehungen zwischen den Required- und ProvidedInterfaces modelliert sind. Über die Schnittstelle *Antenne.Signale*[54] kommen die von der Antenne empfangenen Echoimpulse zur Verarbeitung hinein. Die gewünschten Tracks werden durch den *Radar-Manager* über die Schnittstelle *RM.Tracks* bereitgestellt. Um die Reihenfolge der Software-Komponenten bei der Berechnung der Gesamtverarbeitungszeit ermitteln zu können, ist eine Zuordnung der Aktionen in Abbildung 4.8 zu den Software-Komponenten in Abbildung 4.10 erforderlich. Eine Zuordnung wird durch eine *Dependency*-Verbindung mit dem Stereotyp *realize* zwischen einer Software-Komponente und einer Aktion modelliert. Grundsätzlich kann eine Software-Komponente auch mehrere Aktionen realisieren.

[54] Für die Benennung der Schnittstellen wird an dieser Stelle das Schema <*Quelle*>.<*logische Daten*> verwendet.

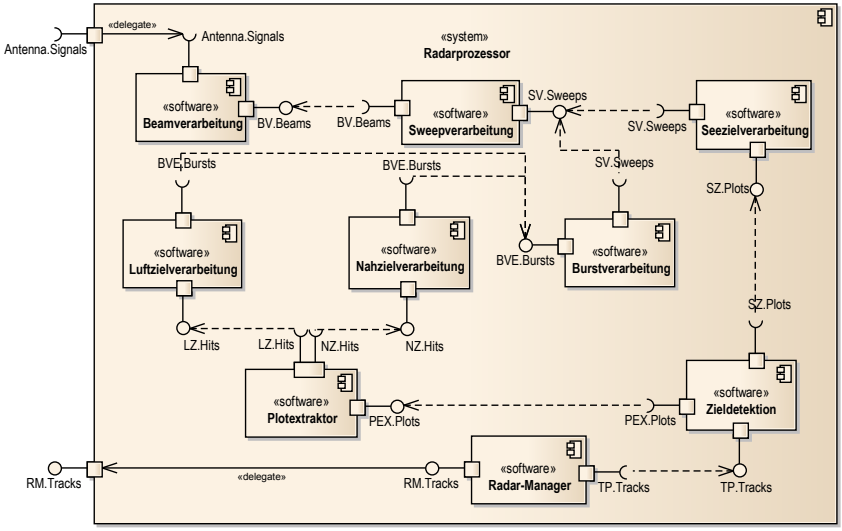


Abbildung 4.9: Vereinfachte Software-Sicht des Radarprozessors

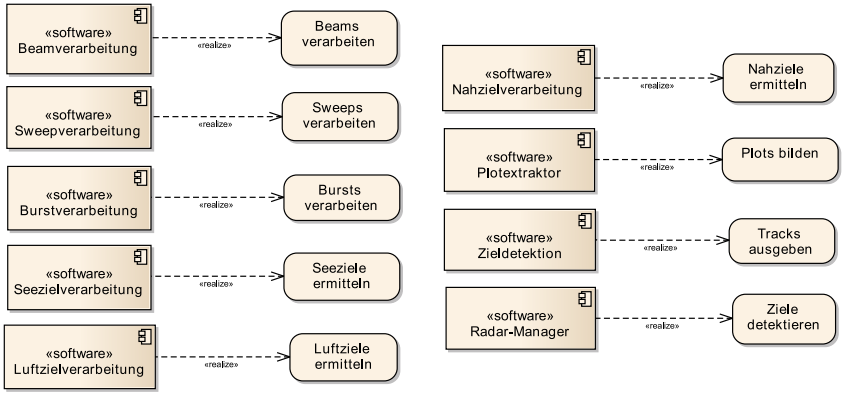


Abbildung 4.10: Zuordnung von Software-Komponenten und Aktionen

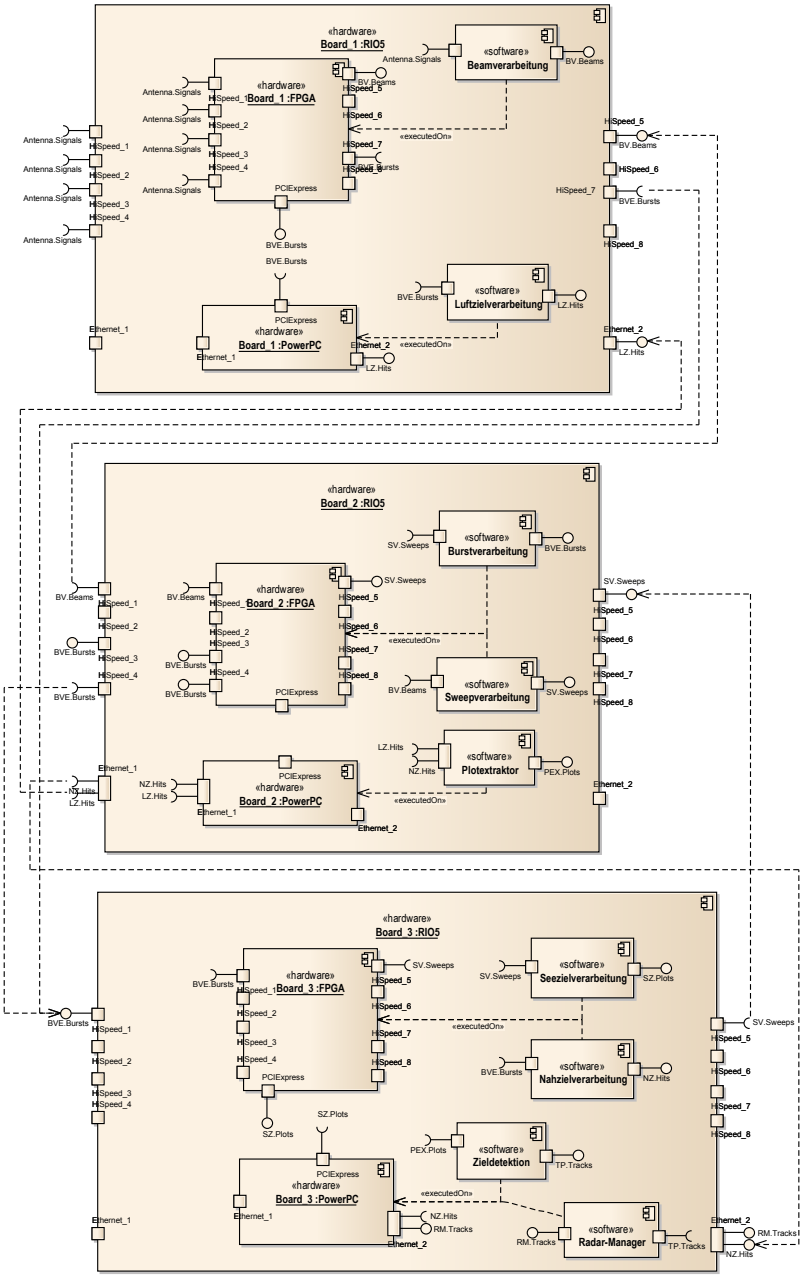


Abbildung 4.11: Deployment-Sicht im Entwicklungsmodell

Abbildung 4.11 zeigt mit Hilfe eines Komponentendiagramms der UML die Zuordnung von Software- und Hardware-Komponenten für das Radar-System. Zur Verarbeitung der empfangenen Signale werden die neun Software-Komponenten auf drei verschiedene Boards verteilt, die jeweils ein *Field Programmable Gate Array (FPGA)* und einen *PowerPC* als verarbeitende Komponenten besitzen. Mit Hilfe der Dependency-Verbindungen mit dem Stereotyp *executedOn* modelliert der Architekt auf welcher Hardware-Komponente eine Software-Komponente ausgeführt wird. Dabei kann eine Hardware-Komponente auch mehrere Software-Komponenten ausführen. Zusätzlich zeigt das Diagramm dem Architekten über welche physikalischen Schnittstellen die Datenkommunikation zwischen den einzelnen Software-Komponenten stattfindet.

Die in den drei Abbildungen 4.8, 4.9 und 4.10 enthaltenen Elemente werden alle für die Validierung der System-Architektur benötigt. Der Architekt transformiert sie deshalb vollständig vom Entwicklungs- ins Validierungsmodell. Von den in Abbildung 4.11 dargestellten Elementen wird nur eine Teilmenge für die Validierung benötigt. Da die Datenkommunikation in diesem Beispiel nicht für die Berechnung der Gesamtverarbeitungszeit berücksichtigt wird, übernimmt der Architekt auch nicht die im Entwicklungsmodell enthaltenen Ports, die Required- und Provided-Interfaces sowie deren Abhängigkeitsbeziehungen in das Validierungsmodell. Das resultierende Diagramm ist in Abbildung 4.12 zu sehen. Neben den Boards, den Hardware- und Software-Komponenten sowie den *executedOn*-Beziehungen sind auch die validierungsspezifischen Informationen zu sehen, die der Architekt mit Hilfe der VSN hinzugefügt hat. Die Werte sind in dem mit *tags* bezeichneten Bereich der Komponenten als TaggedValues dargestellt, z. B. *performance = 700* für die Software-Komponente *Beamverarbeitung*.

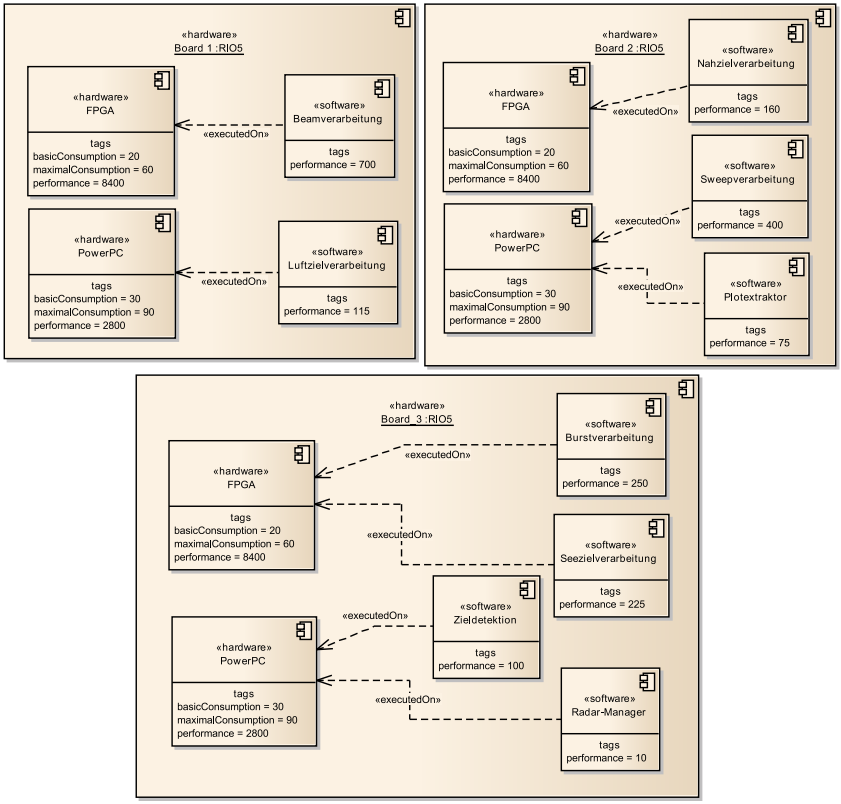


Abbildung 4.12: Zuordnung von Software- zu Hardware-Komponenten im Validierungsmodell

4.3.5 Validierungszielverfahren durchführen

Zur Durchführung der Architekturvalidierung muss der Architekt für jedes Validierungsziel ein Verfahren festgelegt und die dafür benötigten Daten im Validierungsmodell modelliert haben. Sind diese Voraussetzungen erfüllt, werden die Validierungszielverfahren durchgeführt. Dabei ist es für

den Prozess nicht von Bedeutung, ob die Ausführung manuell oder automatisiert (siehe Kapitel 7.1) abläuft.

Das Ergebnis eines Validierungszielverfahrens wird mit den Prüfkriterium des Validierungsziels verglichen. Bei dem Vergleich handelt es sich um einen logischen Ausdruck, durch dessen Auswertung das Validierungsziel entweder als valide oder nicht valide in Bezug auf die gewählte Architektur charakterisiert wird. Falls die Validierung eines oder mehrerer Ziele fehlschlägt, dann gilt die gesamte Architektur als nicht valide gegenüber allen architekturrelevanten Anforderungen. Der Architekt muss die Architektur anpassen. Die Ergebnisse der Validierungszielverfahren, vor allem die Zwischenergebnisse der Simulationen, unterstützen ihn dabei.

Beispiel

Für die Durchführung der Simulation zur Ermittlung der Gesamtverarbeitungszeit werden Testdatensätze verwendet. Diese können entweder vom Kunden bereitgestellt oder von den Entwicklern anhand häufig vorkommender Szenarios erarbeitet werden. An dieser Stelle wird nur ein Testdatensatz betrachtet. Der Architekt hat zwei Validierungsziele identifiziert: Gesamtverarbeitungszeit und Energieverbrauch. Durch die zugeordneten Anforderungen ergeben sich die Grenzwerte 310ms und 220W. Für die See-, Luft- und Nahziele ergeben sich Gesamtverarbeitungszeiten von 285ms, 306ms und 299ms[55]. Die Aussagen $285ms \leq 310ms$, $306ms \leq 310ms$ und $299ms \leq 310ms$ sind alle wahr und V1 damit valide. Tabelle 4.1 zeigt die Auslastung der jeweiligen Verarbeitungseinheiten mit dem daraus resultierenden Energieverbrauch. Die Aussage $226W \leq 220W$ ist falsch und V2

[55] Auf eine genauere Berechnung wird hier verzichtet, da diese durch die parallele Datenverarbeitung etwas aufwändiger ist. Für das Verständnis des Validierungsprozesses sind die Ergebnisse ausreichend. Berechnungsdetails sind in Kapitel 7.2.3.3 beschrieben.

deshalb nicht valide. Die Validierung der Architektur ist damit fehlgeschlagen und der Architekt muss die System-Architektur korrigieren.

Tabelle 4.1: Auslastung der Verarbeitungseinheiten mit zugehörigem Energieverbrauch

Verarbeitungseinheit	**Auslastung [%]**	**Energie [W]**
B1_FPGA	86	34
B1_PowerPC	8	38
B2_FPGA	66	30
B2_PowerPC	28	52
B3_FPGA	15	25
B3_PowerPC	21	47
Energieverbrauch gesamt		**226**

4.3.6 System-Architektur anpassen

Die Anpassung der System-Architektur erfolgt durch den Architekten im Validierungsmodell. Das Validierungsmodell bietet dem Architekten eine validierungsspezifische Sicht auf die Architektur an, so dass der Architekt nicht durch irrelevante Informationen den Überblick verliert und sich vollständig auf die Anpassung konzentrieren kann. Da es sich beim Architekturdesign nicht nur um eine analytische, sondern auch um eine kreative Tätigkeit handelt [MR09], kann der Validierungsprozess dem Architekten diesen Arbeitsschritt nicht abnehmen, sondern bestenfalls unterstützen. Die hier vom Prozess angebotene Unterstützung ist die Anzeige der Abhängigkeiten zwischen den verschiedenen architekturspezifischen Aspekten, d.h. die statische Analyse von Veränderungen auf die Architektur. Statisch bedeutet an dieser Stelle, dass Veränderungen Auswirkungen an einer anderen Stelle in

der Architektur erzeugen können. Ob die Veränderungen sich wirklich auf andere Stellen auswirken und wie diese im Detail aussehen, ist das Ergebnis einer dynamischen Analyse, d. h. der Architekturvalidierung.

Die Unterstützung des Architekten bei der Impact-Analyse ermöglicht der Ansatz durch die Verbindung der Validierungsziele mit den dazugehörigen System-Anforderungen. Entsprechen die Ergebnisse eines Validierungszielverfahrens nicht den Prüfkriterien des Validierungszieles, erlaubt der Ansatz die betroffenen Anforderungen über das dem Verfahren zugeordnete Validierungsziel zu identifizieren. Dem Architekten stehen damit detaillierte Informationen zu dem Validierungsziel zur Verfügung, die er für die Problemanalyse verwenden kann. Darüber hinaus können die Verbindungen der Anforderungen zu anderen Validierungszielen ermittelt werden, so dass dem Architekten bewusst wird, welche architekturspezifischen Aspekte von Änderungen betroffen sind. So bewirkt beispielsweise die Neuverteilung von Software-Komponenten auf die vorhandenen Verarbeitungseinheiten eine auf das ganze System gesehene höhere Auslastung der Verarbeitungseinheiten, was die Gesamtverarbeitungszeit verbessert. Die höhere Auslastung erhöht aber auch den Energieverbrauch und beeinflusst die Betriebstemperatur der Verarbeitungseinheiten. Wie an diesem Beispiel deutlich wird, gibt es bei Architekturen viele *Stellschrauben*, die sich auf verschiedene Bereiche der Architektur auswirken. Vorhersagen über die Auswirkungen der Veränderungen auf die Validierungsergebnisse lassen sich schon für ein kleines System mit einer geringen Anzahl an Validierungszielen und relativ simplen Verfahren nicht mehr treffen. Die Nachverfolgbarkeit durch die Verbindungen zwischen Anforderungen, Validierungszielen und Validierungszielverfahren erlaubt es dem Architekten **vor** dem *Stellen der Schrauben* die davon betroffenen Bereiche der Architektur zu identifizieren.

Beispiel

Da die Validierung von V2 fehlgeschlagen ist, muss der Architekt die System-Architektur anpassen. Er verändert im Validierungsmodell die Zuordnung der Software-Komponenten zu den Verarbeitungseinheiten.

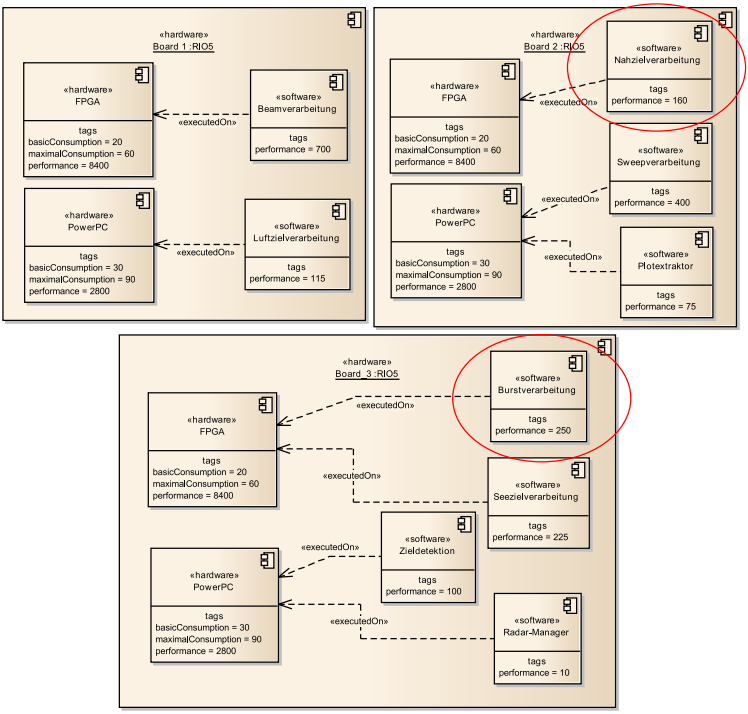


Abbildung 4.13: Veränderte Zuordnung der Software- und Hardware-Komponenten im Validierungsmodell

Die Simulationsdaten helfen ihm bei der Analyse des Problems und bei der Suche nach einer Lösung (siehe Tabelle 4.1). Anhand der Auslastungswerte der Verarbeitungseinheiten ist erkennbar, dass die FPGAs auf Board 1 und 2

eine hohe Auslastung haben, die PowerPCs, insbesondere auf Board 1, aber eine sehr niedrige. Auch das FPGA auf Board 3 hat noch Kapazitäten frei. Der Architekt vertauscht deshalb die Software-Komponenten *Burstverarbeitung* und *Nahzielverarbeitung* miteinander (siehe Abbildung 4.13).

Nach der Anpassung wird die Validierung erneut durchgeführt. Für die Verarbeitungszeit der drei Zielarten ergeben sich die Zeiten 301ms, 309ms und 306ms. Alle Werte sind kleiner gleich 310W, weshalb V1 valide ist. Für den Energieverbrauch ergeben sich die in Tabelle 4.2 dargestellten Werte. Die Aussage $217W \leq 220W$ ist wahr und V2 deshalb valide. Da beide Ziele erfolgreich validiert wurden, ist auch die Gesamtarchitektur gegenüber den den Validierungszielen zugeordneten Anforderungen konform.

Tabelle 4.2: Auslastung der Verarbeitungseinheiten mit zugehörigem Energieverbrauch nach der Anpassung der Architektur

Verarbeitungseinheit	**Auslastung [%]**	**Energie [W]**
B1_FPGA	86	34
B1_PowerPC	8	38
B2_FPGA	84	33
B2_PowerPC	8	38
B3_FPGA	17	26
B3_PowerPC	23	48
Energieverbrauch gesamt		**217**

4.3.7 Modelle synchronisieren

Das den Validierungsprozess beschreibende Aktivitätsdiagramm in Abbildung 4.3 zeigt einen Activity-Parameter mit der Bezeichnung *Modellmodifi-*

kationen für die Aktivität *System-Architektur anpassen* (siehe Auszug in Abbildung 4.14). Die Änderungen werden in einem *datastore* abgelegt und entsprechen den Eingangsdaten der Aktivität *Modelle synchronisieren*. Diese wird nach einer erfolgreichen Validierung der Architektur durchgeführt und überträgt die vorgenommenen Modellmodifikationen, die zu einer validen Architektur geführt haben, in das Entwicklungsmodell. Auf diese Weise wird die Architekturdokumentation im Entwicklungsmodell auf den neuesten Stand gebracht.

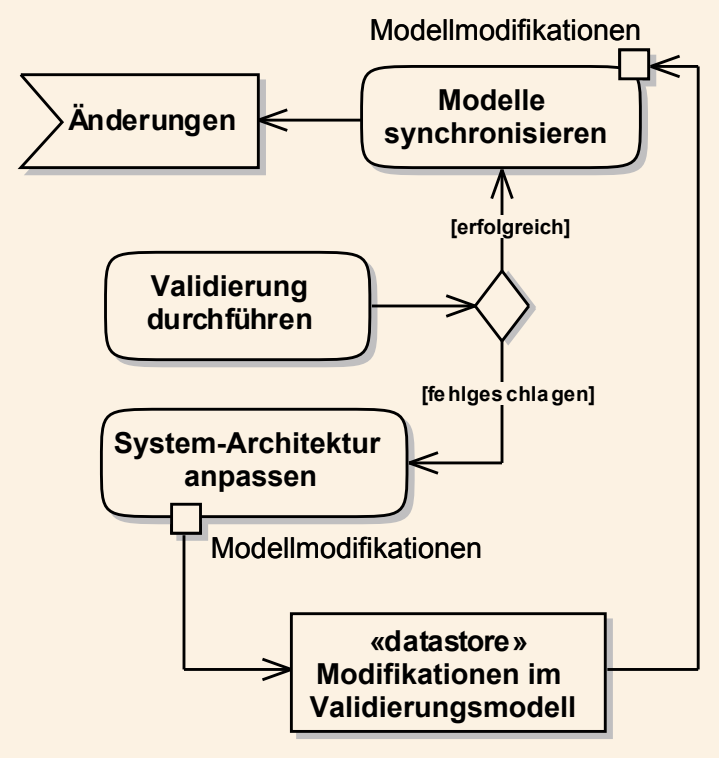


Abbildung 4.14: Auszug aus dem Validierungsprozess

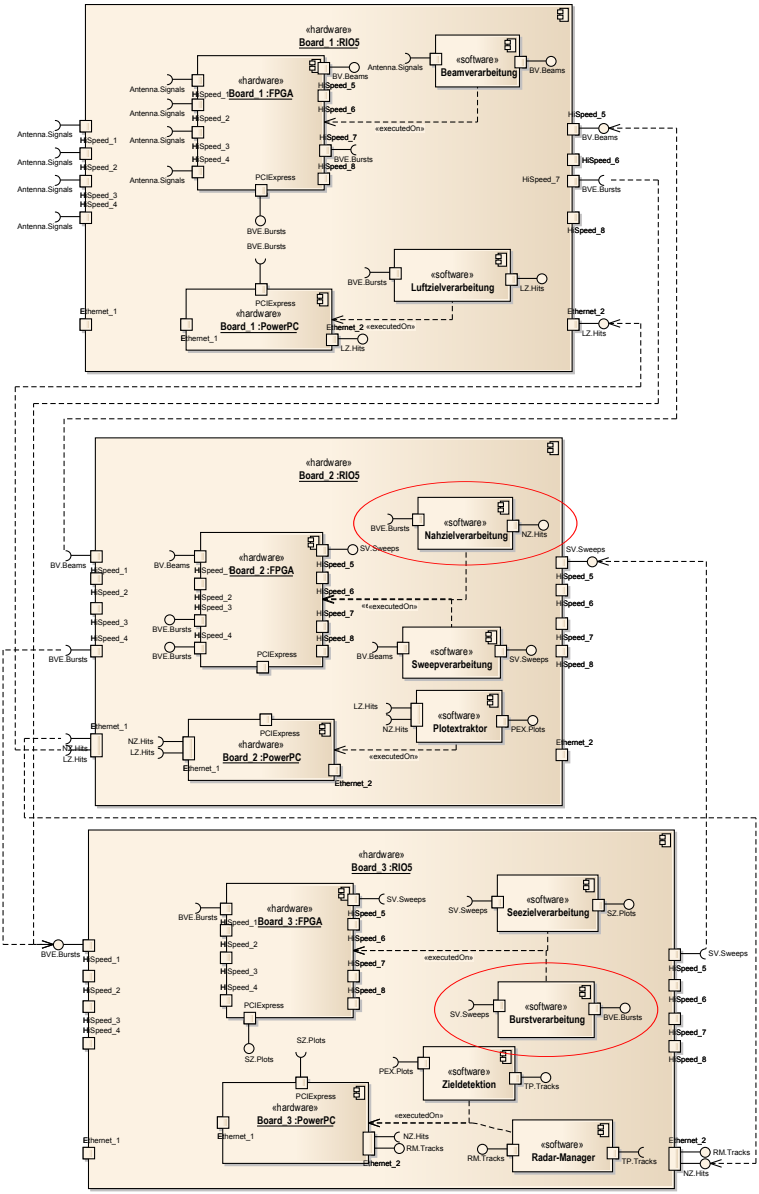


Abbildung 4.15: Veränderte Zuordnung der Software- und Hardware-Komponenten im Entwicklungsmodell

Die explizite Modellierung dieser Änderungsdaten durch die Activity-Parameter und den Datastore dient der Verdeutlichung, dass die veränderten und für das Entwicklungsmodell relevanten Daten aus dem Validierungsmodell in das Entwicklungsmodell übertragen werden. Es ist keine Vorgabe für die Realisierung dieser Datensynchronisation[56].

Beispiel

Der Architekt hat nach der ersten, fehlgeschlagenen Validierung die Zuordnung der Software- und Hardware-Komponenten im Validierungsmodell verändert. Da diese Zuordnung auch im Entwicklungsmodell modelliert ist, müssen zur Synchronisation der beiden Modelle die veränderten Zuordnungen auf das Entwicklungsmodell übertragen werden. Abbildung 4.15 zeigt einen Auszug des an die Änderungen im Validierungsmodell angepassten Entwicklungsmodells.

4.3.8 Architektur revalidieren

Während die letzten Abschnitte jeweils einer Aktivität aus dem Aktivitätsdiagramm in Abbildung 4.3 gewidmet sind, beschreibt dieses Kapitel das Vorgehen bei auftretenden Änderungen bis zur Revalidierung der Architektur. Im Aktivitätsdiagramm wird dies durch das empfangene Signal *Änderungen* und die darauf folgenden Entscheidungsknoten repräsentiert (siehe Abbildung 4.16). In Abhängigkeit von dem was sich geändert hat, muss der Architekt einige Schritte des Prozesses erneut durchführen, um die vorhandenen Artefakte (Validierungsziele, Verfahren, Notationen, Validierungsmodell) entsprechend anzupassen.

[56] Kapitel 7.1.4 beschreibt eine mögliche Realisierung der Modelltransformationen, die als Basis für eine Synchronisation der beiden Modelle verwendet werden kann.

Wird ein *Validierungszielverfahren* verändert, beispielsweise durch die Wahl einer anderen, detaillierteren Berechnung, muss der Architekt im Schritt *Validierungszielverfahren festlegen* überprüfen, ob diese Änderung Auswirkungen auf andere Verfahren hat und diese ggf. anpassen. Anschließend muss er überprüfen, ob die validierungsspezifische Notation angepasst werden muss, weil einige Eingangsdaten nicht mehr benötigt werden oder neue zur Durchführung der Validierung erforderlich sind. Die fehlenden validierungsspezifischen Daten werden im Validierungsmodell ergänzt, überflüssige entfernt. Auf Basis des aktualisierten Validierungsmodells kann der Architekt dann die *Validierung durchführen*.

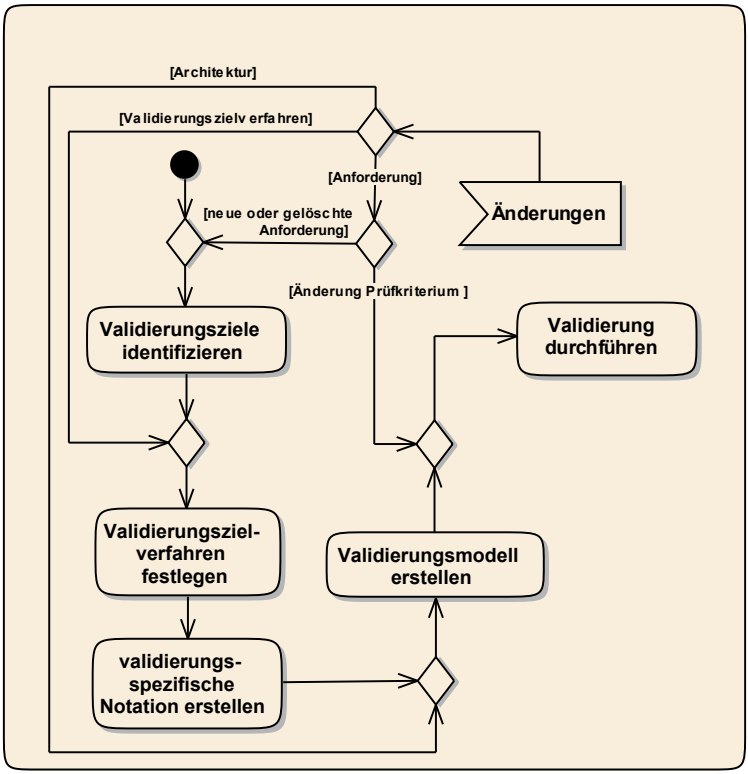


Abbildung 4.16: Ausschnitt aus dem Validierungsprozess

Wurden Änderungen am Enwicklungsmodell vorgenommen, beispielsweise um eine Architekturalternative zu erarbeiten oder aufgrund des Fortschritts in der Systementwicklung[57], muss der Architekt die Änderungen in das Validierungsmodell übernehmen und anschließend die *Validierung durchführen*.

Wurden Änderungen an einer oder mehreren Anforderungen vorgenommen, sind zwei Fälle zu unterscheiden:
Falls lediglich ein oder mehrere Prüfkriterien[58] geändert wurden, kann der Architekt direkt die *Validierung durchführen*. Das Ändern eines Prüfkriteriums hat laut Definition (siehe Seite 73) keinen Einfluss auf das Validierungsziel, so dass weder neue hinzukommen oder vorhandene gelöscht werden[59]. Dies schließt auch Veränderungen an Validierungszielverfahren und am Validierungsmodell aus.

Der zweite Fall verursacht den größten Aufwand und tritt vor allem zu Beginn der Systementwicklung auf. Das Hinzufügen oder Löschen von Anforderungen kann Auswirkungen auf die Validierungsziele und damit auch auf die Validierungszielverfahren, die Notationen sowie das Validierungsmodell haben. Beim Hinzufügen einer neuen Anforderung muss diese einem neuen oder einem bzw. mehreren vorhandenen Validierungszielen zugeordnet werden. Beim Löschen kann der Architekt die betroffenen Validierungsziele anhand der Verbindungen zwischen Anforderung und den Validierungszielen direkt identifizieren. Danach ermittelt er für die betroffenen

[57] Hiermit sind neben Architketuranpassungen auch Änderungen an den Analyseergebnissen gemeint.

[58] Die Prüfkriterien sind Teil der Anforderungen.

[59] Das Verändern eines Validierungsziels kommt nur selten vor. Bei den bekannten Fällen handelt es sich meist um die Wahl eines anderen Validierungszielnamens, was aber keine weiteren Veränderungen im Verfahren und Modell nach sich zieht. Bei Änderungen, die darüber hinaus gehen, kann auch vom Löschen oder vom Neuanlegen ausgegangen werden.

Validierungsziele die erforderlichen Veränderungen an der validierungsspezifischen Notation und dem Validierungsmodell. Mit dem aktualisierten Modell kann er die *Validierung durchführen*.

Beispiel

Als Beispiel für eine notwendige Revalidierung wird der Austausch von Hardware-Komponenten im Entwicklungsmodell herangezogen. Dieser Fall tritt besonders häufig zu Beginn eines Projekts auf, wenn sich der Architekt noch nicht auf bestimmte Hardware-Bauteile festgelegt hat oder sich die Festlegung im Laufe des Projekts verändert. Auch die fortschreitende Entwicklung im Hardwarebereich während der Projektlaufzeit kann eine Anpassung der verwendeten Hardware zur Folge haben. Höhere Leistung, geringerer Stromverbrauch, höhere Kommunikationsbandbreiten oder schnellere Speicherzugriffszeiten schaffen ggf. mehr Freiräume für den Architekten und ermöglichen neue oder vorher nicht realisierbare Architekturvarianten. Weitere nicht zu vernachlässigende Faktoren sind die Kosten, der Verschleiß und der Einsatzbereich der Hardware, z. B. in besonders kalten oder warmen Umgebungen. Der Architekt kann durch den Validierungsprozess systematisch und gezielt nach möglichen Architekturalternativen suchen. Die Entscheidung, welche der validen Architekturen verwendet wird, liegt allerdings außerhalb des beschriebenen Prozesses.

5 Abgrenzung und Vergleich mit anderen Arbeiten

Dieses Kapitel ordnet den in der Arbeit vorgestellten Ansatz in die Systementwicklung ein und grenzt ihn gegenüber anderen Forschungsarbeiten, Methoden und Werkzeugen ab. Einige Grundlagen hierfür werden in Kapitel 5.1 beschrieben. Kapitel 5.2 stellt verschiedene Entwicklungsmodelle vor und beschreibt, wie sich der Ansatz in diese einordnen lässt. Des Weiteren beschreibt Kapitel 5.2 am Beispiel eines Ermittlungsansatzes für NFAs, wie sich die Ergebnisse vorausgehender Tätigkeiten für den Validierungsprozess nutzen lassen. Die Struktur der Kapitel 5.3 bis 5.5 ergibt sich aus der Gruppierung der verschiedenen Forschungsarbeiten, Methoden und Werkzeuge, die in Abbildung 5.1 dargestellt ist.

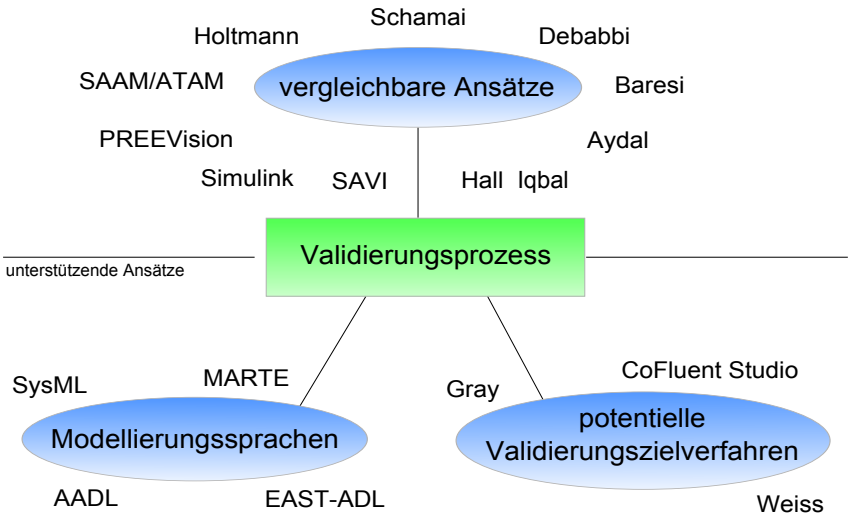


Abbildung 5.1: Einordnung der verschiedenen Literaturquellen

Im oberen Bereich der Abbildung sind Ansätze über ihren Namen oder den Hauptautor der veröffentlichten Forschungsarbeit dargestellt, die zur Abgrenzung des Validierungsprozesses verwendet werden (siehe Kapitel 5.3). Im unteren Bereich sind Ansätze aufgeführt, die bei der Durchführung einzelner Prozessschritte unterstützend eingesetzt werden können. Hier wird zwischen potenziellen Validierungszielverfahren (siehe Kapitel 5.4) und Modellierungssprachen (siehe Kapitel 5.5) unterschieden.

Zusammenfassend lässt sich an dieser Stelle schon sagen, dass bisher kein Validierungsprozess existiert, der

- auf Basis von dynamischen Prüftechniken, genauer Simulationen, und der UML, als Quasi-Standard in der Industrie für die modellbasierte Dokumentation, basiert,
- eine systematische Prüfung der System-Architektur gegenüber den architekturrelevanten System-Anforderungen anbietet und
- den System-Architekten bei der Analyse von Anforderungsänderungen auf eine vorhandene System-Architektur oder bei der Analyse von Architekturänderungen auf die Einhaltung der architekturrelevanten System-Anforderungen unterstützt.

5.1 Grundlagen

Dieses Kapitel beschreibt Methoden und Verfahren, die bei der Abgrenzung des Validierungsprozesses verwendet werden. Kapitel 5.1.1 beschreibt das Anforderungsmetamodell nach Dörr [Dör10], auf das Kapitel 5.2.5 näher eingeht. Kapitel 5.1.2 beschreibt das Model-Checking, eine häufig eingesetzte

formale Prüftechnik. Es wird von einigen Ansätzen in Kapitel 5.3 verwendet.

5.1.1 Anforderungsmetamodell

Kapitel 2.4.3 beschreibt kompakt was Dörr unter dem Begriff Anforderung versteht und wie diese klassifiziert werden. Dieses Kapitel geht detaillierter auf die Definitionen ein, indem es das Anforderungsmetamodell vorstellt. Für die Beschreibung des vollständigen NFA-Prozesses wird auf Anhang A.1 verwiesen.

Überblick über die Arbeit In seiner Dissertation [Dör10] beschreibt Jörg Dörr einen systematischen Ansatz zur Erhebung, Analyse und Spezifikation einer vollständigen Menge[60] nicht-funktionaler Anforderungen. Grundlage für diesen NFA-Prozess (siehe Kapitel A.1) ist ein in der Arbeit vorgestelltes Anforderungsmetamodell. In ihm werden die verwendeten Artefakte, z. B. die NFAs, mit ihren Beziehungen untereinander erfasst, wodurch sich beispielsweise Widersprüche identifizieren lassen. Der verwendete Erhebungsalgorithmus bringt u. a. den Vorteil mit sich, dass er eine iterative Durchführung unterstützt, was den immer später im Entwicklungszeitraum stattfindenden Architekturentscheidungen entgegenkommt. Für die Erstellung des Ansatzes wurden unterschiedliche Quellen und Ansätze aus dem Bereich Requirements Engineering ausgewertet und die Ergebnisse in einer Arbeitsgruppe der Gesellschaft für Informatik (GI) diskutiert. Das Metamodell spiegelt also die Vorstellungen verschiedener Experten wieder. Der Mehrwert des Ansatz wurde anhand verschiedener praktischer Projekte gezeigt.

[60] Im Dissertationstitel ist von *complete set* die Rede, wobei in der Arbeit darauf hingewiesen wird, dass der vorgestellte Ansatz in der Theorie Vollständigkeit anvisiert, dies aber in der Praxis nicht oder nur sehr schwierig umzusetzen ist.

Anforderungsmetamodell Abbildung 5.2 zeigt einen Ausschnitt aus dem Anforderungsmetamodell. In ihm sind die bereits in Kapitel 2.4.3 angesprochenen Product Requirements, Functional Requirements, Non-Functional Requirements und die Architectural Constraints zu sehen.

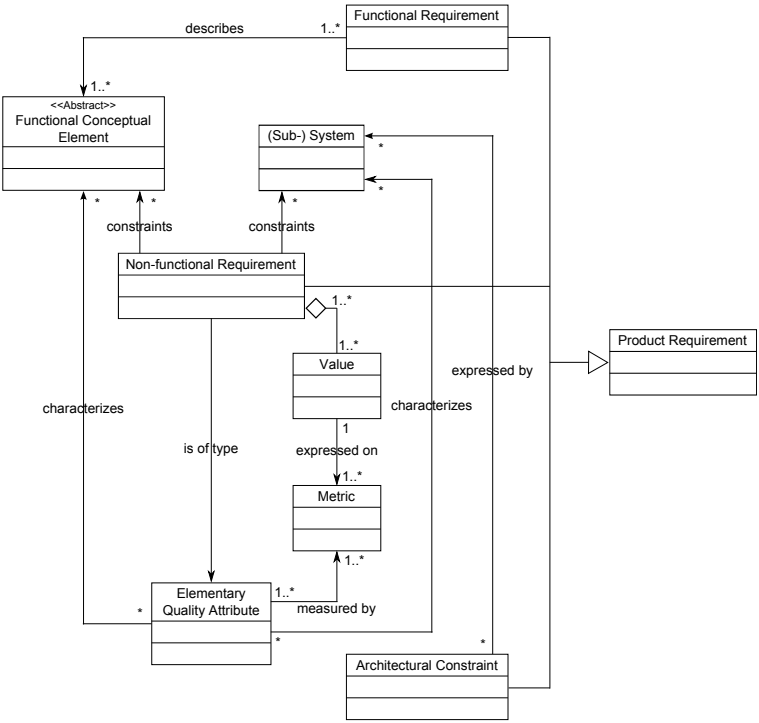


Abbildung 5.2: Vereinfachtes Anforderungs-Metamodell, nach [Dör10, S. 24]

Die Architectural Constraints sind eine für den vorgestellten Ansatz zur Ermittlung der nicht-funktionalen Anforderungen relevante Untermenge der Product Constraints. Die Unterteilung der Product Requirements ist hier

durch eine Vererbungsbeziehung modelliert. Die Functional Requirements beschreiben mindestens ein FCE, das folgendermaßen definiert ist:

> "A Functional Conceptual Element is an abstract concept that serves as a placeholder for different kinds of functional conceptual elements, like system functions or data items." [Dör10, S. 23]

Mit dem Begriff (Sub-)System[61] ist eine Menge von Objekten gemeint, die das System selbst und all seine (bereits bekannten) physikalischen Komponenten enthält. Durch die Architectural Constraints ergeben sich die vorgegebenen Objekte in der Menge der Subsysteme, z. B. physikalische Komponenten wie PCs, Handys, Smartphones usw. Diese Constraints sind vor der Systementwicklung festgelegte Vorgaben; sie entstehen nicht während der Design-Phase. Die nicht-funktionalen Anforderungen haben eine einschränkende Wirkung auf die Subsysteme und die FCEs. Sie besitzen einen Typ, das Elementary Quality Attribute (EQA), und mindestens ein zugewiesenes Value-Element. Das abstrakte EQA ist ein Schlüsselelement des Metamodells. EQAs sind die "quality characteristics" [Dör10, S. 24] der FCEs und der Subsysteme (siehe *characterizes*-Verbindungen in Abbildung 5.2), wodurch sie mit Hilfe der dazugehörigen Metrik messbar und damit analytisch auswertbar sind. Die Definition lautet:

> "An Elementary Quality Attribute is a measurable characteristic of one or more functional conceptual elements or subsystems that describes any other characteristic of this element but not its functionality (i.e. inputs, outputs, or input-output-

[61] Für einen besseren Lesefluss wird Subsystem als Synonym für (Sub-)System verwendet.

> relationship). These other characteristics are often called quality characteristics." [Dör10, S. 25]

Die Value-Elemente, die den nicht-funktionalen Anforderungen zugeordnet sind, beinflussen die Metriken. Sie sind indirekt über das EQA den nicht-funktionalen Anforderungen zugeordnet und sind im Grunde die zur Überprüfung der Anforderung notwendigen Testverfahren. Das erwartete Testergebnis ist durch die Values vorgegeben, für die eine passende Metrik gefunden werden muss. Zusammengefasst ist dies in der folgenden Definition der nicht-funktionalen Anforderungen:

> "A non-functional requirement (NFR) constraints one or more functional conceptual elements or subsystems by determining values (or value domains) for one or more metrics of a specified elementary quality attribute that should be achieved by the functional conceptual element or subsystem. The non-functional requirement is of the type of the elementary quality attribute that characterizes the functional conceptual element or subsystem." [Dör10, S. 26]

Nach diesem groben Überblick über das Anforderungsmetamodell folgt nun eine schrittweise Vertiefung. Zunächst werden die beiden abstrakten Elemente FCE und EQA beschrieben, anschließend werden die für die System-Architektur relevanten *Means*-Elemente vorgestellt. Mit diesem Wissen lässt sich das in Abbildung 5.6 vollständig dargestellte Metamodell erläutern.

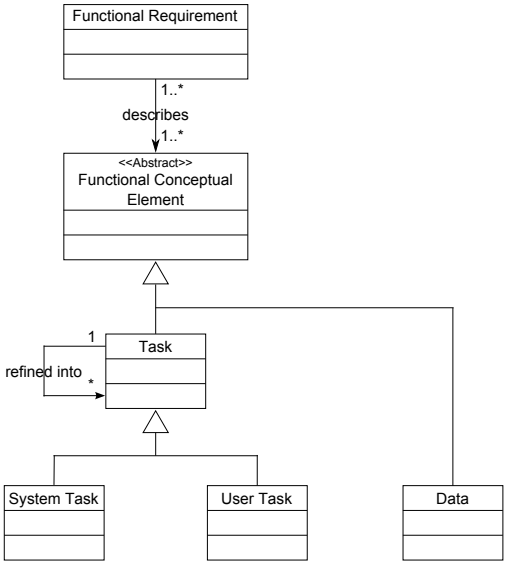


Abbildung 5.3: Ausprägungen der Functional Conceptual Elements, nach [Dör10, S. 28]

Abbildung 5.3 zeigt die möglichen Ausprägungen der FCEs, die von den funktionalen Komponenten beschrieben werden. Grundsätzlich wird zwischen *Data* und *Tasks* unterschieden; ein *Task* wird noch weiter in *User Task* und *System Task* unterteilt. Unter dem Begriff *Data* werden in das System eingegebene, vom System manipulierte und an andere Systeme oder die Systemumgebung weitergegebene Daten verstanden. User Tasks sind Aufgaben, die vom Benutzer mit dem System durchgeführt werden. Das System wirkt unterstützend bei der Abarbeitung der Aufgabe, wobei immer, in irgendeiner Form, eine Beteiligung des Benutzers stattfindet. User Tasks werden im Bereich der Business-Prozesse, Szenario- oder Use Case-Beschreibungen verwendet. Findet keine Benutzerbeteiligung statt, das System arbeitet die Aufgabe also selbstständig ab, handelt es sich um einen *Sys-*

tem Task. Synonyme für einen System Task sind die Begriffe *System Function* und *Automated Task*. Ein Task lässt sich durch weitere Tasks verfeinern, wobei ein User Task durch eine Kombination von System Tasks und User Tasks verfeinert werden kann, ein System Task aber nur durch weitere System Tasks.

Abbildung 5.4: Ausprägungen der Elementary Quality Attributes, nach [Dör10, S. 31]

Das EQA beschreibt den Typ einer nicht-funktionalen Anforderung und charakterisiert FCEs und Subsysteme. Wie in Abbildung 5.4 zu sehen ist, handelt es sich hierbei um eine spezielle Form des abstrakten Elements QA. Hierarchisch über den QAs liegen die High Level Quality Attributes, die Vorgaben auf Systemebene geben und nicht messbar sind, z. B. Effizienz,

Sicherheit, Portabilität oder Zuverlässigkeit. Diese High Level Quality Attributes werden durch QAs verfeinert, die folgendermaßen definiert sind:

> "A quality attribute (QA) is a quality (non-functional) characteristic of a functional conceptual component or subsystem. As in the definition of elementary QA, the term quality characteristic refers to any characteristic other than functionality." [Dör10, S. 32]

QAs können sich gegenseitig beeinflussen, sowohl im positiven als auch im negativen Sinne. Eine Performanzsteigerung kann sich beispielsweise positiv auf die Antwortzeit eines Systems auswirken, die Steigerung geht aber einher mit einem höheren Energieverbrauch des gesamten Systems. Das *Structural Quality Attribute* ist ein Strukturierungselement zum Aufbau eines *Quality Models*[62]. Es besitzt keine Metrik und muss durch ein oder mehrere EQAs verfeinert werden. Die Bedeutung der vier Subtypen der EQAs ergibt sich durch die Elemente, die von den EQAs charakterisiert werden und deren Name Teil der Subtypenbezeichnung sind. Das *System QA* bezieht sich auf die Subsysteme, zu denen das System selbst und die durch die Architectural Contraints vorgegebenen physikalischen Komponenten gehören. Sie werden typischerweise für das ganze System oder eine der Subkomponenten erhoben. Ein Beispiel hierfür ist die Kapazität einer Datenbank. Ein *User Task QA* charakterisiert einen User Task und wird für Business-Prozesse oder vollständige Use Cases erhoben. Als Beispiel für ein User Task QA kann die benötigte Zeit zur Durchführung der Aufgabe (engl. *usage time*) genannt werden. Der System Task wird durch das namentlich dazu passen-

[62] Ein Quality Model entspricht einer Liste oder Hierarchie von Quality Attrributen, die zur Qualitätsbeschreibung von Systemkomponenten dienen. In der Praxis gibt es unterschiedliche Modelle, beispielsweise die ISO 830, 1326, 9126, 25000 oder die Volere Shell.

de *System QA* näher beschrieben, wobei diese QAs für System-Funktionen oder System-Features sowie einzelnen nur vom System durchgeführten Use Case-Schritten erhoben werden. Ein Beispiel hierfür ist die Antwortzeit eines bestimmten Systemschritts. Das *Data QA* charakterisiert Data-Elemente des Systems, beispielsweise die Genauigkeit bei der Datenspeicherung.

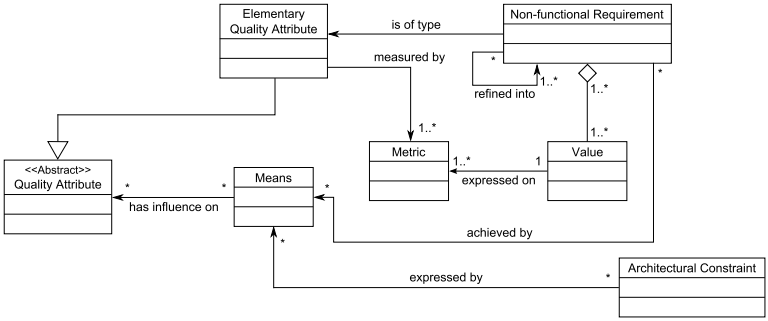


Abbildung 5.5: Beziehungen des Means-Element, nach [Dör10, S. 35]

Für die Menge der architekturrelevanten Anforderungen, die für den Validierungsprozess benötigt werden, sind die im Metamodell als *Means* bezeichneten Elemente von Bedeutung. Dörr definiert den Begriff wie folgt:

> "A Means describes an architectural decision that can be applied to the architecture to influence a certain QA and therefore achieve certain NFRs. [...] [It] should have positive influence on the target QAs but as a side effect it could have negative influence on other QAs." [Dör10, S. 36]

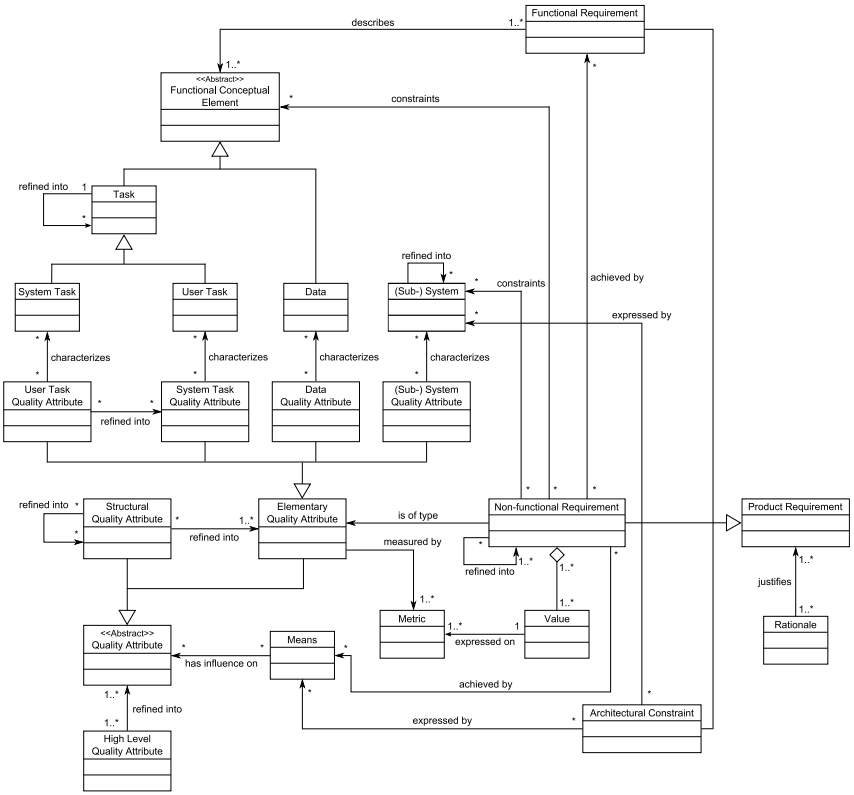


Abbildung 5.6: Das vollständige Anforderungsmetamodell, nach [Dör10, S. 35]

Means sind keine Anforderungen, sondern architekturspezifische Entscheidungen, die während der Design-Phase definiert werden und zum Erreichen von nicht-funktionalen Anforderungen dienen. Sie sind nicht mit den Architectural Constraints zu verwechseln, die zu den Anforderungen zählen und vordefinierte Architektureinschränkungen beinhalten. Ein Beispiel für ein Means ist die Verwendung eines Design-Patterns. Es wirkt positiv auf ein oder mehrere QAs, allerdings kann deren Anwendung auch einen ne-

gativen Seiteneffekt mit sich bringen, der in Kauf genommen wird. Means können sich durch die formulierten Architectural Contraints ergeben, was durch die *expressedBy*-Beziehung in Abbildung 5.5 ausgedrückt wird. Über die Means lässt sich der Einfluss der nicht-funktionalen Anforderungen auf die Architektur im Metamodell modellieren. Dabei sollte bei der Auswahl eines Means-Elements berücksichtigt werden, dass der EQA-Typ der nicht-funktionalen Anforderung in der Menge der von dem Means-Element beeinflussten QAs enthalten ist. Das vollständige Anforderungsmetamodell von Dörr ist in Abbildung 5.6 dargestellt. Es setzt sich in etwa aus den Abbildungen 5.2, 5.3 bis 5.5 zusammen. Neben den bereits in den Beschreibungen implizit beschriebenen *characterizes*-Verbindungen zwischen den vier Verfeinerungen des EQAs und den dazugehörigen FCE-Ausprägungen und Subsystemen, kommt noch das *Rationale*-Element hinzu. Hiermit ist eine Begründung für die Existenz von Produkt-Anforderungen gemeint, wodurch diese überhaupt erst gerechtfertigt wird. Durch solche Rationales werden Goldrandlösungen[63] vermieden.

5.1.2 Model-Checking

Die Technik des Model-Checkings [BK08] [Bér01] ist ein formales Verfahren zur Prüfung einer Systembeschreibung. Der Begriff *formal* drückt an dieser Stelle aus, dass das zu untersuchende System mit Hilfe von mathematischen Ausdrücken beschrieben wird, was eine präzise und eindeutige Beschreibung des Systems ermöglicht. Da es sich bei der Systembeschreibung um eine Abstrahierung der Realität handelt, wird an Stelle von Beschreibung auch der Begriff Modell verwendet. Bei den mathematischen Ausdrücken

[63] Unter Goldrandlösungen fallen Ausschmückungen oder Funktionen, für die keine Anforderungen innerhalb der Spezifikation existieren. Der Benutzer kann diese positiv aufnehmen oder als nicht nützlich bewerten, wodurch der Realisierungs- und Testaufwand überflüssig war. [Rup09b, S. 23]

handelt es sich meist um temporale Logiken [Das05] [Bér01]. Sie ergeben zusammen eine endliche Menge von Zuständen, die in ihrer Gesamtheit das Systemverhalten beschreiben. Bei der Überprüfung einer Eigenschaft, die aus den Anforderungen abgeleitet und formalisiert wurde, werden alle möglichen Systemzustände untersucht. Wird ein Zustand entdeckt, in dem die Eigenschaft nicht eingehalten wird, ist die Prüfung fehlgeschlagen.

Model-Checking basiert auf Graphtheorien und Logik und somit auf einem soliden mathematischen Fundament. Dadurch ist eine präzise Modellierung des Systems möglich, wobei durch die Modellierung selbst bereits Inkonsistenzen und Lücken in der Beschreibung aufgedeckt werden können. Es kann für verschiedene Anwendungsgebiete, beispielsweise eingebettete Systeme, Software Engineering oder Hardware-Design, angewendet werden. Im Gegensatz zum Testen liegt der Fokus des Model-Checkings nicht auf dem Finden eines bestimmten oder des wahrscheinlichsten Fehlers. Durch das systematische Ausprobieren aller Möglichkeiten werden auch Fehler in nicht bedachten Systemzuständen gefunden. Das ist ein bedeutender Vorteil des Model-Checkings. Durch die Wahl der zu überprüfenden Eigenschaften kann der Fokus auch auf bestimmte, z. B. für das System essentielle, Eigenschaften gelegt werden. Dies erlaubt eine partielle Überprüfung des Systems [BK08].

Neben den genannten Vorteilen des Model-Checkings gibt es aber auch einige Nachteile. Einer ergibt sich durch die Charakteristik des Verfahrens selbst. Das System-Modell besteht aus einem endlichen Zustandsautomat, was besonders für kontrollintensive Systeme geeignet ist. Für datenintensive Systeme, wie beispielsweise ein Radar-System, ist es weniger geeignet, da der Definitionsbereich der Daten typischerweise in Richtung unendlich geht [BK08]. Solche Systeme sind auf Grund des Entscheidbarkeitsproblems

(siehe [Weg03]) nicht effizient berechenbar. Zur Verkleinerung des Definitionsbereichs sind Abstraktionstechniken erforderlich [CGL94] [Pel06].

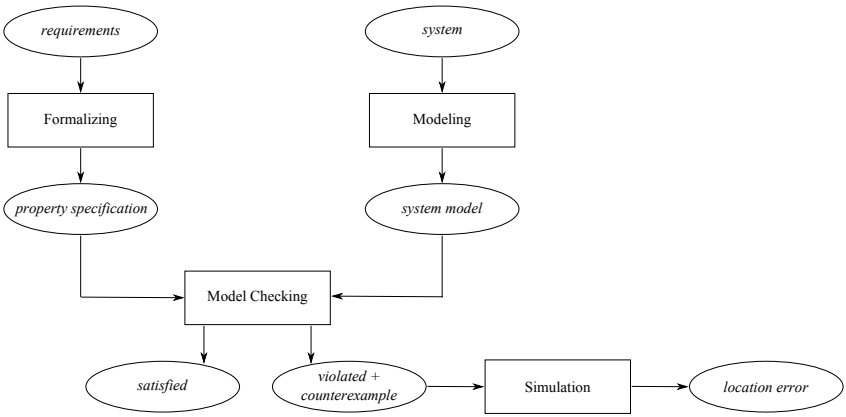


Abbildung 5.7: Schematische Sicht des Model-Checking-Ansatzes, nach [BK08]

Abbildung 5.7 zeigt die schematische Sicht des Model-Checking-Ansatzes. Die Model-Checking-Technik hat als Eingaben die *property specification* und das *system model*. Hier ist eine klare Trennung zu erkennen zwischen dem Systemverhalten (system model) und den zu überprüfenden Eigenschaften, die beschreiben, was das System machen kann und was nicht (property specification). Die Ergebnismenge des Model-Checkings ist übersichtlich. Entweder das System-Modell erfüllt eine Eigenschaft oder es wird ein Gegenbeispiel ausgegeben, in dem die Eigenschaft verletzt wird. Das Gegenbeispiel besteht aus dem Pfad vom initialen Systemzustand bis zu einem Zustand, der die Eigenschaft verletzt. Zusammen mit dem System-Modell dient es als Eingabe für eine Simulation, mit deren Hilfe der Anwender die genaue Ursache des Fehlers herausfinden und anschließend das System-Modell oder die Property-Spezifikation anpassen kann.

Die Anwendung des Model-Checkings zur Prüfung eines Designs wird in [BK08] in die Phasen *Modeling, Running* und *Analysis* unterteilt. Zusätzlich sollten die Planung, Verwaltung und Organisation der Prüfung nicht außer Acht gelassen werden. Die Phase *Modeling* umfasst die Erstellung des System-Modells mit Hilfe der vom Model-Checking-Werkzeug bereitgestellten Modellierungssprache. Hierbei kann es sich beispielsweise um Erweiterungen von C, Java oder der Very High Speed Integrated Circuit Hardware Description Language (VHDL) handeln. Die hiermit erstellten endlichen Zustandsautomaten sollten einer Plausibilitätsprüfung in Form von Simulationen unterzogen werden [BK08]. Auf diese Weise können grobe Modellierungsfehler bereits vor der Durchführung des Model-Checkings identifiziert und beseitigt werden. Neben dem Systemverhalten müssen auch die zu überprüfenden Eigenschaften formalisiert werden. Hierfür wird eine Property Specification Language (PSL) [IEE10] verwendet, beispielsweise die lineare temporale Logik (LTL) [BK08]. Mit der LTL ist es möglich, Bedingungen für eine zeitliche Abfolge[64] zu definieren. Die Bedingungen oder Eigenschaften werden mit Hilfe der Aussagenlogik definiert und entsprechen den Zuständen eines Systems. Mit Hilfe von temporalen Operatoren können diese verknüpft werden, wodurch Vorgaben an das (zeitliche) Verhalten eines Systems beschrieben werden. Beispiele hierfür sind, dass ein Zustand X niemals eingenommen werden darf oder dass ein Zustand Z nur eingenommen werden darf, falls das System vorher im Zustand Y war. Auf Basis des System-Modells und der formalisierten Eigenschaften wird in der Phase *Running* das Model-Checking-Werkzeug die Gültigkeit einer Eigenschaft für alle Zustände des Systems prüfen. Die Auswertung des Ergebnisses erfolgt in der Phase *Analysis*. Die Ergebnismenge besteht aus drei Werten. Entweder die Eigenschaft ist in dem gegebenen System-Modell gültig, sie ist es nicht

[64] In der Mathematik wird an dieser Stelle der Begriff *Spur* verwendet.

oder das System-Modell passt nicht in den Speicher des Computers. Der zuletzt genannte Fall wird auch *State Explosion-Problem* genannt. Zur Vermeidung muss das System-Modell verkleinert werden, beispielsweise durch die Wahl einer anderen Abstraktionsebene [CGL94]. Hierzu können verschiedene Abstraktionstechniken eingesetzt werden [Pel06]. Ist die Eigenschaft gültig, kann die nächste zu überprüfenden Eigenschaft geprüft werden. Ist die Eigenschaft nicht gültig, gibt es mehrere Fehlerursachen. Bei der Analyse des Fehlers kann sich herausstellen, dass das System-Modell nicht dem realen Design oder die formalisierte Eigenschaft nicht der realen Anforderung entspricht. In beiden Fällen ist eine Anpassung notwendig. Nach einer Anpassung des System-Modells muss die Prüfung für alle zuvor geprüften Eigenschaften wiederholt werden. Können beide Fehlerfälle ausgeschlossen werden, liegt ein Designfehler vor. Das Design muss angepasst werden und damit auch das System-Modell, welches das Design repräsentiert. Das Design ist gültig, falls alle Eigenschaften für das (valide) System-Modell gültig sind. Da beim Model-Checking sehr viele Informationen, z. B. verschiedene System-Modelle aufgrund unterschiedlicher Abstraktionslevel, Ergebnisse sowie unterschiedliche Parameter für das Model-Checking-Werkzeug, entstehen, ist eine gute Organisation des Model-Checking-Prozesses notwendig. Nur durch eine sorgfältige Verwaltung und Versionierung dieser Daten ist eine effiziente Anwendung möglich, die auch reproduzierbarer Ergebnisse liefert.

Des Weiteren ist die Wahl der Abstraktionsebene für das System-Modell entscheidend für eine erfolgreiche und sinnvolle Prüfung. Eine zu detaillierte Modellierung kann zum State-Explosion-Problem führen, eine zu geringe Detailstufe kann das Ergebnis unbrauchbar machen, weil entscheidende Einflussfaktoren nicht berücksichtigt werden. Die Formalisierung der Eigenschaften mit Hilfe der PSL ist für einen ungeübten Anwender schwierig und

zeitaufwändig. Selbst nach einer Einarbeitungszeit existiert bei der Umsetzung einer Eigenschaft in die PSL eine erhöhte Fehlerwahrscheinlichkeit, die über Flüchtigkeitsfehler hinausgeht. Da aus einer falschen Annahme, d. h. einer fehlerbehafteten formalisierten Eigenschaft, Beliebiges gefolgert werden kann, ist es schwierig die Korrektheit des Ergebnisses einzuschätzen. Die Wahl der Abstraktionsebene und die Formalisierung der Eigenschaften erfordern ein hohes Maß an praktischer Erfahrung. Dies ist zwar auch bei nicht-formalen Verfahren der Fall, allerdings ist in der Praxis zu beobachten, dass die Akzeptanz nicht-formaler Verfahren höher ist und damit auch die Bereitschaft sich in die Verfahren einzuarbeiten. Eine Ursache für dieses Akzeptanzproblem ist die für viele Anwender ablehnende Wirkung der mathematischen (logischen) Ausdrücke. Die theoretischen Grundlagen, Logik und Graphtheorie, sind den potenziellen Anwendern nicht unbekannt, die Schwierigkeit ist aber die Anwendung dieses Wissens auf das konkrete Problem. Hier bedarf es praktischer Erfahrung mit dem Model-Checking-Werkzeug und der Formalisierung der zu überprüfenden Eigenschaften. Sofern diese nicht vorhanden ist oder die Projektrandbedingungen, Anforderungen oder hohe zu erwartende Kosten im Fehlerfall den Einsatz formaler Methoden nicht erzwingen, bleibt im Projektgeschäft selten Zeit, ausreichend Erfahrung mit neuen Verfahren zu sammeln. So sind die formalen Prüftechniken bisher größtenteils nur in der Entwicklung kritischer Systeme Teil des Entwicklungsprozesses [WLBF09].

5.2 Einordnung in die Systementwicklung

Dieses Kapitel ordnet den in der Arbeit vorgestellten Validierungsprozess in unterschiedliche Prozessmodelle der Systementwicklung ein (Kapitel 5.2.1 bis 5.2.4). Kapitel 5.2.5 beschreibt wie sich die Ergebnisse des NFA-Prozesses

nach Dörr (siehe Kapitel 5.1.1 und Anhang A.1) als Datenbasis für die Durchführung des Validierungsprozesses verwenden lassen.

Zusammenfassend und etwas verallgemeinert lässt sich sagen, dass sich der in der Arbeit vorgestellte Validierungsprozess während und am Ende der Design-Phase einordnet, um die Architektur gegenüber den Anforderungen zu validieren. Die Validierung kann dabei auf Informationen mit unterschiedlichem Detaillierungsgrad basieren, so dass sie beispielsweise mit Annahmen über die Laufzeit einer Software-Komponente auf einer noch nicht verfügbaren Hardware-Komponente oder mit Laufzeitberechnungen durch die Ausführung konkreter fachlicher Algorithmen durchgeführt werden kann. Das Identifizieren der Validierungsziele sowie die Festlegung der Validierungszielverfahren und Prüfkriterien kann durch die Bereitstellung gut aufbereiteter und qualitativ hochwertiger Anforderungen unterstützt werden.

5.2.1 ISO 15288

Die ISO/IEC 15288 [ISO08] mit dem Titel *Systems and software engineering - System life cycle processes* definiert vier Prozessgruppen zur Unterstützung des Systems Engineering: Technical Processes, Project Processes, Agreement Processes und Organizational Project-Enabling Processes. Der Validierungsprozess ist den *Technical Processes* zuzuordnen, zu denen unter anderen die Anforderungsanalyse, der Architekturentwurf, die Implementierung sowie die Validierung und Verifikation gehören. Da es sich um einen Prozess zur Prüfung der Architektur gegenüber den Anforderungen handelt, ist er dem Architekturentwurf zuzuordnen und kann als integraler Bestandteil verwendet werden. Der Kontext für die Validierung und Verifikation ist das System als Ganzes.

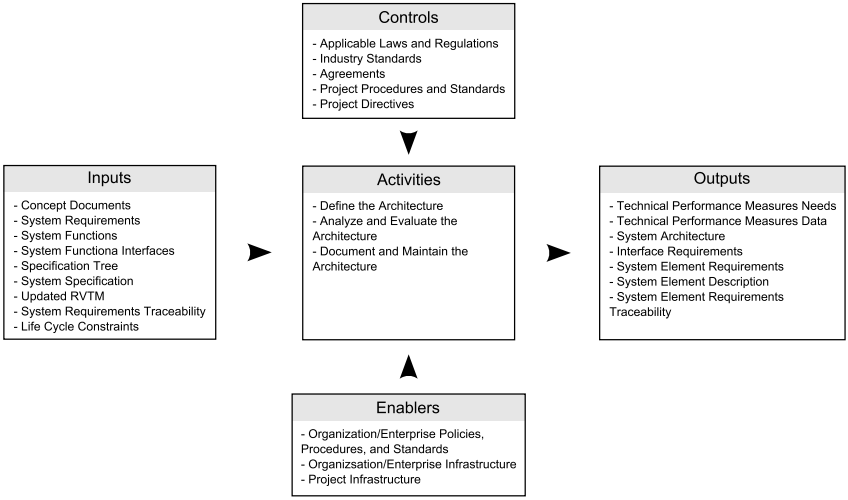


Abbildung 5.8: Kontext-Diagramm für den Architekturentwurfsprozess, nach [INC11, S. 94]

Abbildung 5.8 zeigt das Kontext-Diagramm für den Architekturentwurfsprozess aus dem INCOSE-Handbuch. Die *Activities* werden von drei verschiedenen Faktoren, den *Inputs*, den *Controls* und den *Enablers*, beeinflusst und erzeugen *Outputs*. Bei den *Outputs* handelt es sich um verarbeitete Daten, Dokumentation, Produktinkremente und/oder Dienste. Mit *Inputs* sind Daten und Materialien gemeint, *Controls* ist der Sammelbegriff für Anweisungen und Einschränkungen, z. B. rechtliche, und unter *Enablers* werden Ressourcen wie Infrastruktur und Personal, Werkzeuge, Technologien und organisationsspezifische Richtlinien verstanden.

Eines der Kriterien, das eine System-Architektur erfüllen muss, ist die Erfüllung der an sie gestellten Anforderungen. Dabei stellt die Erfüllung der funktionalen Anforderungen selten ein Problem dar, falls keine Einschränkungen an das System vorgegeben sind. Es sind aber genau diese Einschrän-

kungen, seien sie in nicht-funktionalen Anforderungen festgehalten oder durch *Controls* oder *Enablers* vorgegeben, welche die Menge der in Frage kommenden Architekturen verkleinern. Die Vorgabe der Reaktionszeit auf ein Ereignis, die Einhaltung eines Budgets, der maximale Energieverbrauch oder die Verwendung einer bereits im Unternehmen vorhandenen Infrastruktur sind Beispiele für architekturrelevante Einschränkungen.
Es ist die Aufgabe des Architekten, eine oder mehrere Architekturen zu finden, die alle Randbedingungen erfüllt. Der Validierungsprozess unterstützt den Architekten bei der systematischen Überprüfung, ob die von ihm erstellte Architektur den Randbedingungen genügt. Die initiale Erarbeitung der Architektur liegt zeitlich vor dem Validierungsprozess. Er kann aber dazu beitragen, dass eine initial nicht vollständig valide Architektur schrittweise durch den Architekten verbessert wird, wobei der Prozess Hinweise auf die verletzten Anforderungen gibt. Analog dazu kann die Veränderung von Anforderungen gesehen werden, die eine partiell invalide Architektur erzeugt.
In der detaillierten Beschreibung der Aktivitäten des Architekturentwurfprozesses empfiehlt das INCOSE-Handbuch die Entwicklung mehrere geeigneter Architekturen. Aus dieser Menge möglicher Kandidaten kann dann die finale Version ausgewählt werden [INC11, S. 104]. An dieser Stelle unterstützt der Validierungsprozess den Architekten, für die unterschiedlichen Entwürfe reproduzierbare Validierungsergebnisse auf Basis eines vorgegebenen und für alle identischen Verfahren zu erstellen. Eine solche Reproduzierbarkeit ist bei der oft verwendeten Review-Methode aufgrund des menschlichen Faktors nicht zu erzielen.

5.2.2 V-Modell XT

Das V-Modell XT ist eine Weiterentwicklung des V-Modells 97. Es wird vom Amt des Beauftragten der Bundesregierung für Informationstechnik,

dem Chief Information Officer (CIO) Bund, bereitgestellt; die Weiterentwicklung wird vom WEIT e.V.[65] übernommen. Das XT steht für eXtreme Tailoring, die Anpassung der durchzuführenden Aktivitäten und des zu erstellenden Produkts an das jeweilige Projekt. Dabei gibt das V-Modell XT nur indirekt über Entscheidungspunkte einen groben zeitlichen Ablauf in Form von Abschnitten vor. In diesen Abschnitten stehen die den Entscheidungspunkten zugeordneten Produkte im Vordergrund. Es wird außerdem ein standardisiertes Vorgehen für Auftraggeber- (AG) und Auftragnehmer-Projekte (AN) inklusive einer AG/AN-Schnittstelle definiert, die sogenannten Produktdurchführungsstrategien [TU 13] [Kli07]. Abbildung 5.9 zeigt die Projektdurchführungsstrategie des V-Modells XT für ein Auftragnehmer-Projekt. In der Abbildung sind den Entscheidungspunkten, dargestellt als Parallelogramme, die zu erstellenden Produkte zugeordnet.

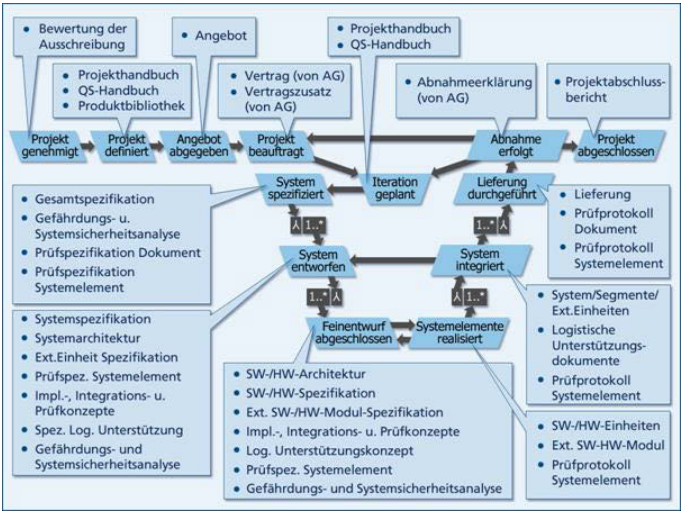


Abbildung 5.9: Produktdurchführungsstrategie eines Auftragnehmer-Projektes, nach [Kli07]

[65] http://www.weit-verein.de/

Der Validierungsprozess unterstützt den Architekten bei der Überprüfung der System-Architektur gegenüber den Anforderungen. Da die Erstellung der System-Architektur dem Entscheidungspunkt *System entworfen* zugeordnet ist, lässt sich die Anwendung des Validierungsprozesses diesem Abschnitt zuordnen.

5.2.3 COSMOD-RE

[Bro06] weist darauf hin, dass sowohl das Requirements Engineering als auch der Architekturentwurf Innovationsquellen für die Systementwicklung sind. Diesen Fakt greifen [PS07b] und [PS07a] auf und stellen einen Ansatz vor, der die strikte Trennung bei der Erarbeitung von Anforderungen und Architektur im Entwicklungsprozess auflöst und die Erarbeitung in Form einer nebenläufigen Entwicklung ermöglicht.

Es wird darauf hingewiesen, dass gerade im Bereich der Entwicklung von innovativen softwareintensiven (eingebetteten) Systemen, die auch das Anwendungsfeld des Validierungsprozesses sind, akzeptiert wird, dass eine solche nebenläufige Entwicklung (engl. *co-design*) notwendig ist. Der in [PS07a] als sCenario and gOal based SysteM delOpment methoD (COSMOD)-Requirements Engineering (RE) vorgestellte Ansatz definiert vier Abstraktionsebenen: *System*, *funktionale Komponenten*, *Hardware-/Software-Komponenten* und *Software-Komponenten*. In jeder Ebene können Artefakte parallel erzeugt werden, die dann in den darunterliegenden Ebenen sukzessive verfeinert werden. Zu den vier Abstraktionsebenen definiert COSMOD-RE drei Co-Design-Prozesse, deren Fokus auf einer bestimmten Ebene und deren Nachfolger liegt. Abbildung 5.10 zeigt die enstehenden Artefakte der drei Co-Design-Prozesse und deren Zugehörigkeit zu den Abstraktionseebenen.

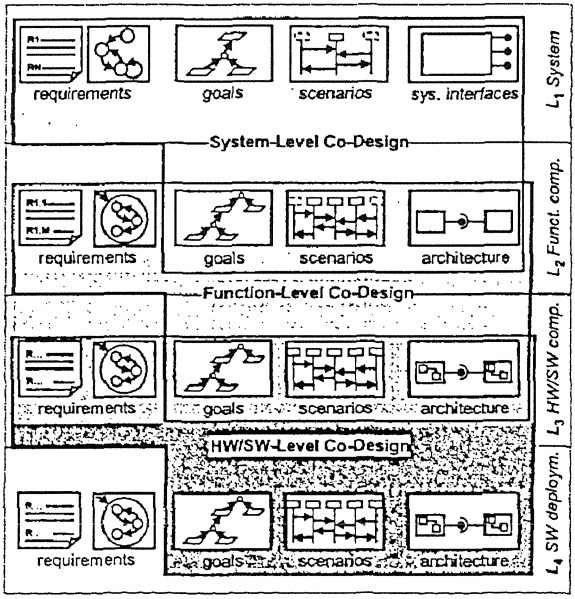


Abbildung 5.10: Abstraktionsebenen des COSMOD-RE-Ansatzes mit den Entwicklungsprozessen und den erzeugten Artefakten, nach [PS07a]

Jeder Co-Design-Prozess wird noch einmal in fünf Sub-Prozesse unterteilt, die in Abbildung 5.11 dargestellt sind. Zunächst findet parallel eine Entwicklung der Anforderungssicht und der Architektursicht statt. Im nächsten Schritt werden die Ergebnisse dieser beiden Prozesse verglichen, anschließend konsolidiert und basierend auf diesen Ergebnissen detaillierte Anforderungen definiert. Für eine genaue Beschreibung der Co-Design-Prozesse und der Sub-Prozesse wird auf [PS07b] und [PS07a] verwiesen. Der für die Einordnung des Validierungsprozesses interessante Sub-Prozess ist *SP3*. Die zwei Hauptziele dieses Sub-Prozesses sind:

1. Einhaltung der Anforderungen durch die Architektur
2. Identifizierung neuer Anforderungen auf Basis der vorgeschlagenen Architektur durch Vergleich der Ergebnisse von *SP1* und *SP2*.

Der Validierungsprozess kann zur Erreichung des ersten Ziels verwendet werden. Da der Validierungsprozess mit den frei wählbaren Verfahren zur Überprüfung der Einhaltung der architekturrelevanten Anforderungen keine Vorgaben zum Detaillierungsgrad der Architektur macht, ist ein Einsatz in allen drei Co-Design-Prozessen denkbar. In Abhängigkeit vom Detaillierungslevel ist eine sukzessive Anpassung der Verfahren und damit der für die Validierung notwendigen Informationen erforderlich. Der Detaillierungsgrad des HW/SW-Level-Co-Design-Prozesses wird vermutlich[66] dem Grad des in der vorliegenden Arbeit verwendeten Beispiels entsprechen.

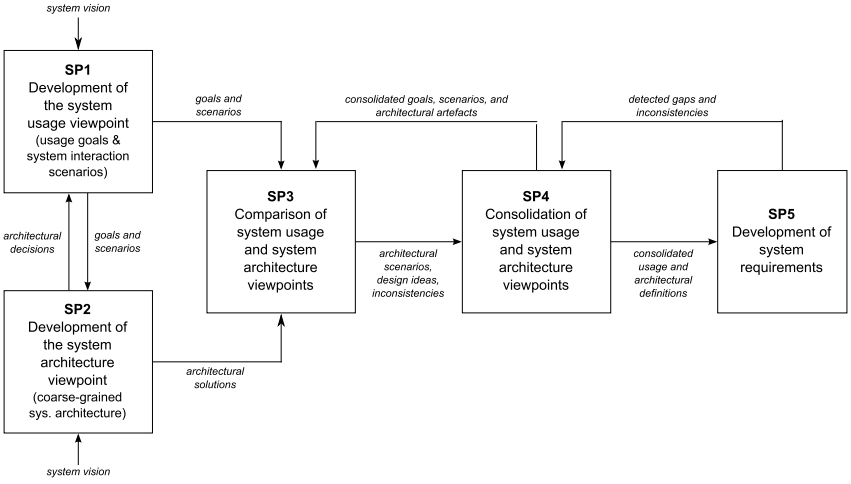


Abbildung 5.11: 5 Subprozesse der Co-Design-Ansatzes, nach [PS07a]

66 Die in den Papern enthaltenen Beispiele sind leider nicht so detailliert, dass dadurch die Vermutung bestätigt werden kann. Die Vermutung basiert auf dem aus den Papern erarbeiteten Verständnis.

5.2.4 Competitive Engineering

Tom Gilb beschreibt in [GB05] das Competitive Engineering, eine auf der Planguage (plan language) basierende Methodik, um Systeme zu erstellen, die wettbewerbsfähig (engl. *competitive*) sind. Die Planguage besteht aus der *Planugage Specification Language* und den *Planguage Process Descriptions*. Sie ist eine formale Spezifizierungssprache [Emm10], die für Anforderungen, Designs und die Projektpläne verwendet werden kann. Gilb klassifiziert Anforderungen in *Visions*, *Function Requirements*, *Performance Requirements* und *Resource Requirements* sowie *Design Constraints* und *Condition Constraints*. Die Planguage bietet eine Menge empfohlener Vorgehensweisen für die Ausführung bestimmter Aufgaben an, die projektspezifisch angepasst werden können und sollten. Die Menge der *Planguage Process Descriptions* besteht aus:

- Requirements Specification (RS),
- Design Engineering (DE),
- Specification Quality Control (SQC),
- Impact Estimation (IE) und
- Evolutionary Project Management (Evo).

Der für die Einordnung des Validierungsprozesses entscheidende Prozess ist das Design Engineering. Er besteht grundlegend aus vier Prozessschritten, die aus weiteren Subprozessen bestehen:

- P1: Analyze the Requirements
- P2: Find and Specify Design Ideas
- P3: Evaluate Design Ideas
- P4: Select Design Ideas and Produce Evo Plan

Der Validierungsprozess kann zur Unterstützung von *P3: Evaluate Design Ideas* eingesetzt werden, der in seinen Subprozessen die bereits genannte *Impact Estimation* enthält:

- P3.1: Filter for Violation of any Constraint
- P3.2: Estimate all Impacts
- P3.3: Consider Side Effects
- P3.4: Consider Safety Margins

Das Herausfiltern von Anforderungsverletzungen ist die Hauptaufgabe des Validierungsprozesses, weshalb er *P3.1* zuzuordnen ist. Die Impact Estimation wird zur Evaluierung von Design-Vorschlägen gegen die Anforderungen mit Hilfe numerischer Werte eingesetzt und unterstützt bei der Risikobewertung. Auf diese Weise lassen sich verschiedene Design-Vorschläge miteinander vergleichen. Durch die Validierungsverfahren und die zu erfüllenden Grenzwerte in den Anforderungen[67] wird auch die Impact Estimation, P3.2, vom Validierungsprozess in einer ähnlichen Form angeboten.

5.2.5 Nutzung von Informationen aus dem NFA-Prozess

Während die vorherigen Kapitel von der Einordnung des in der Arbeit vorgestellten Validierungsprozesses in verschiedene Prozessmodelle der Systementwicklung handeln, beschreibt dieses Kapitel, wie die Informationen aus dem NFA-Prozess nach Jörg Dörr (siehe Kapitel 5.1.1 und Anhang A.1) für den in der Arbeit vorgestellten Ansatz verwendet werden können.

[67] Anforderungen müssen testbar sein, weshalb nicht-funktionale Anforderungén meist quantifizierte Vergleichswerte für einen Test enthalten. Falls eine Anforderung nicht unmittelbar testbar ist (z. B. 24x7-Verfügbarkeit), muss hierfür ein Abnahmekriterium definiert werden.

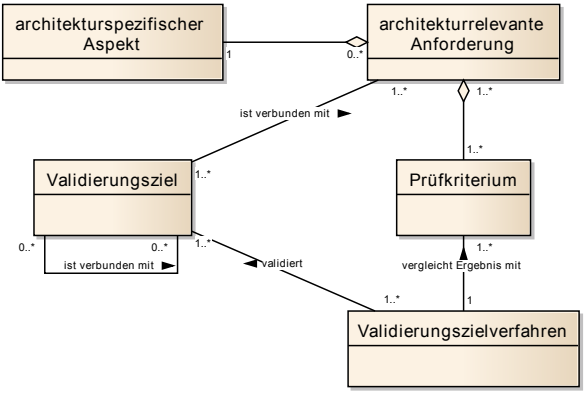


Abbildung 5.12: Beziehungen zwischen ausgewählten Elementen des Validierungsprozesses

Der erste Schritt des Validierungsprozesses ist die Identifizierung der Validierungsziele. Sie ergeben sich aus den für die Architektur relevanten Anforderungen. Hier müssen (vor allem) die nicht-funktionalen Anforderungen betrachtet werden, da sie die Menge der möglichen Architekturen zur Realisierung der durch die funktionalen Anforderungen geforderten Funktionalitäten einschränken [Rom85]. Für die Ermittlung dieser Anforderungen wird auf bereits vorhandene Ansätze verwiesen, beispielsweise [Dör10] oder [Rup09b]. Einen guten Überblick gibt auch [HKD09].

Die Zusammenhänge zwischen den Begriffen architekturrelevante Anforderung, architekturspezifischer Aspekt, Validierungsziel, Validierungszielverfahren und Prüfkriterium (siehe Begriffsmodell in Kapitel 4.3) sind in Abbildung 5.12 in Form eines UML-Klassendiagramms dargestellt. Ein Validierungsziel repräsentiert einen bestimmten architekturspezifischen Aspekt und ist mit mindestens einer architekturrelevanten Anforderung verbunden, die wiederum die Ziele beeinflusst. Es wird durch ein Validierungszielver-

fahren validiert, wobei ein Verfahren für mehrere Ziele verwendet werden kann. Die Ergebnisse des Verfahrens werden mit den Prüfkriterien verglichen, die in den architekturrelevanten Anforderungen enthalten sind. Falls sich Validierungsziele direkt oder indirekt über die zugeordneten architekturrelevante Anforderungen beinflussen, wird eine Verbindungen zwischen diesen erstellt.

Der NFA-Prozess verwendet ein Anforderungsmetamodell, in dem die verwendeten Artefakte und deren Beziehungen untereinander enthalten sind. Ein Ausschnitt ist in Abbildung 5.13 dargestellt. Um die Testbarkeit zu gewährleisten, ist jeder NFA ein messbarer Vergleichswert (Value) zugeordnet, wobei ein Value auch zu mehreren NFAs gehören kann. Die Values beeinflussen die Metriken. Jeder NFA ist ein EQA zugewiesen, das anhand einer oder mehrerer Metriken gemessen werden kann.

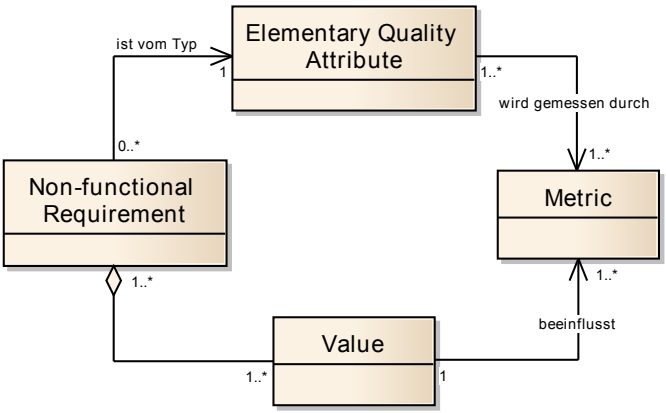


Abbildung 5.13: Beziehungen zwischen ausgewählten Elementen des Anforderungsmetamodells nach Dörr

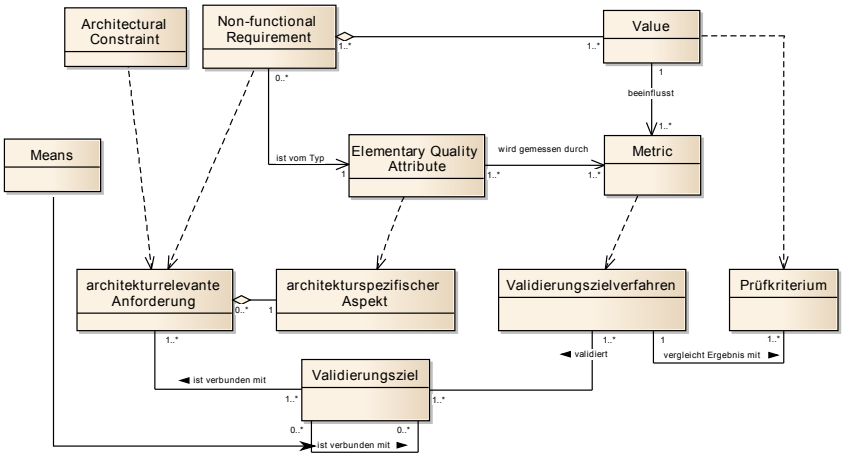


Abbildung 5.14: Nutzung von Elementen des Anforderungsmetamodells für den Validierungsprozess

Abbildung 5.14 illustriert, wie die Informationen aus dem Anforderungsmetamodell (oberer Bereich) für den Validierungsprozess (unterer Bereich) genutzt werden können. So lässt sich beispielsweise aus dem EQA der architekturspezifische Aspekt ableiten. Dabei muss es sich aber nicht unbedingt um eine 1:1-Abbildung handeln, beispielsweise bei der Wahl einer unterschiedlichen Detaillierungsebene für Qualitätsattribute und Aspekte. Falls eine Metrik vorhanden ist, kann dies dem Architekten als Hinweis für die Erstellung eines konkreten Validierungszielverfahren dienen. Hier ist vor allem anhand der Verfügbarkeit von Testdaten zu berücksichtigen, ob eine Übernahme möglich oder eine Anpassung erforderlich ist. Aus dem Value-Element lässt sich ein Teil des Prüfkriteriums herleiten. Der logische Operator muss vom Architekten ergänzt werden. Aus der Menge der NFAs kann der Architekt die zu einem Validierungsziel gehörenden architekturrelevanten Anforderungen herausfiltern. Da die beiden Ansätze unterschiedliche

Definitionen für den Begriff der nicht-funktionalen Anforderung verwenden (siehe Kapitel 2.4), lassen sich auch aus den Architectural Constraints architekturrelevante Anforderungen für ein Validierungsziel ableiten. Means werden eingesetzt, um NFAs erfüllen zu können. Sie haben Einfluss auf EQAs, die wiederum NFAs zugeordnet sind. Aus diesen Beziehungen lassen sich direkte und indirekte (über gemeinsame architekturrelevante Anforderungen) Abhängigkeiten zwischen Validierungszielen herausarbeiten.

Das Beispiel zeigt welche Daten aus einer qualitativ hochwertigen und strukturierten Anforderungsspezifikation für die ersten Prozessschritte der Architekturvalidierung genutzt werden können. Zwar lassen sich die Daten nicht automatisiert übertragen, es lässt sich aber der Aufwand für den Architekten zur Durchführung der Schritte reduzieren.

5.3 Vergleichbare Ansätze

Dieses Kapitel grenzt den in der Arbeit vorgestellten Validierungsprozess gegenüber anderen Methoden und Werkzeugen ab. Dazu werden zunächst die verschiedenen Ansätze vorgestellt und anschließend die Unterschiede zum hier vorgestellten Validierungsprozess herausgestellt. Dabei werden die Begriffe Validierung und Verifizierung so verwendet, wie sie in den jeweiligen Ansätzen genutzt werden. Dies gilt sowohl für die Beschreibung der Ansätze als auch für den Vergleich und die Abgrenzung zu dem in der Arbeit vorgestellten Ansatz. Wenn von der Verifizierung einer Architektur gesprochen wird, entspricht dies der in der vorliegenden Arbeit verwendeten Semantik für den Begriff Validierung (siehe Kapitel 2.5).

Zusammenfassend lässt sich sagen, dass viele Ansätze auf formale Prüftechniken zur Validierung der Architektur setzen. Die formalen Prüftechniken sind aber weniger für datenintensive Systeme (State-Explosion-Problem beim Model-Checking (siehe 5.1.2), zur Ermittlung von Zwischenergebnissen in der Simulation und für die Berücksichtigung komplexer Abhängigkeitsbeziehungen (siehe 5.3.10) geeignet. Sie besitzen trotz ihres Potentials eine geringe Akzeptanz in der praktischen Anwendung [WLBF09] [DBH13]. Der Fokus einiger Ansätze liegt nur auf der physikalischen Architektur (Schamai, Simulink, PREEvision/Adler), wodurch einige Ergebnisse der System-Analyse, die Software-Komponenten der System-Architektur und deren Zuordnung zur Hardware außen vor gelassen werden. Für die Durchführung werden meist detaillierte Informationen über die verwendeten Algorithmen und die System-Architektur benötigt. Gerade dieses Detailwissen steht aber in den frühen Phasen der Entwicklung, bei der Erstellung der ersten Architekturentwürfe, noch nicht zur Verfügung. Durch die Spezialisierung auf eine bestimmte Domäne lässt sich dieses Informationsproblem lösen (Holtmann für den Automotive-Bereich, Hall/Iqbal für Realzeit-Systeme), allerdings ist die Übertragbarkeit auf andere Domänen damit nicht mehr gegeben. Eine Alternative zu den formalen Prüftechniken ist die Generierung von Testfällen aus einem autarken Test-Modell zur Prüfung des Designs. Leider werden beim Ansatz von Aydal die für das Design wichtigen nicht-funktionalen Anforderungen nicht weiter berücksichtigt. Verfahren wie SAAM/ATAM dienen der Bewertung verschiedener Architekturentwürfe, der in der Arbeit vorgestellte Ansatz hat aber das Ziel, die Konformität eines Architekturentwurfs gegenüber den Anforderungen zu prüfen.

5.3.1 Schamai

Wladimir Schamai et al. stellen einen Ansatz zum Erstellen einer virtuellen Verifizierungsumgebung zur Modellierung von physikalischen Systemen, Anforderungen und Testfällen vor [SHFP10] [SFP13]. Mit dieser lässt sich das physikalische Design eines Systems bereits in frühen Projektphasen simulieren und gegen Anforderungen verifizieren. Entsprechend trägt der Ansatz den Namen: **v**irtual **V**erification of system **D**esign against system **R**equirements (vVDR). Der Ansatz kann in die folgenden Schritte aufgeteilt werden:

1. Anforderungen zur Verifizierung auswählen
2. Textuelle Anforderungen formalisieren
3. Design-Modell auswählen
4. Test-Modell erstellen und mit Anforderungen verknüpfen
5. Test-Suite durchführen
6. Testergebnisse auswerten

Abbildung 5.15 zeigt die Beziehungen zwischen den im Ansatz verwendeten Artefakten. Das Design-Modell muss die textuellen Anforderungen erfüllen, was mit Hilfe von Testfällen überprüft wird. Die Testergebnisse werden mit den Anforderungen verglichen. Um dies automatisiert durchführen zu können, werden formalisierte Anforderungen, sogenannte Requirements Violation Monitors (RVMs), erstellt, die für die Testfälle als Repräsentation einer Teilmenge der natürlichsprachlichen Anforderungen verwendet werden. Ein RVM wird über die Auswahl einer Anforderungsteilmenge und die Identifizierung von messbaren Eigenschaften in diesen Anforderungen erstellt. Die RVMs werden als ausführbare Modelle zur Überwachung der durch die zugeordneten Anforderungen definierten Bedingungen beschrieben. Zum

Testen eines Designs wird eine explizite Verbindung zwischen Design und den RVMs erstellt. Desweiteren wird eine Test-Suite mit einem Kontext und einer Menge von Testfällen manuell erstellt, wobei die formalisierten Anforderungen als Test-Orakel dienen, d. h.: wird eine Anforderung verletzt oder nicht evaluiert, ist der Testfall fehlgeschlagen.

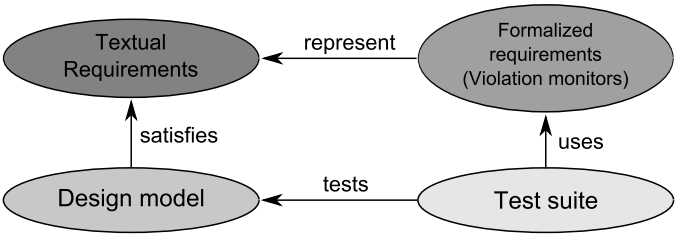


Abbildung 5.15: Beziehungen zwischen den Daten im vVDR-Ansatz, nach [SHFP10]

Der Ansatz basiert auf Modelica[68], einer objektorientierten Sprache zur textuellen Beschreibung großer, komplexer und heterogener physikalischer Systeme [Fri11]. Sie erlaubt die Beschreibung des physikalischen Systemverhaltens durch eine Kombination von gewöhnlichen Differentialgleichungen. Das so beschriebene Verhalten wird zusammen mit einer Wertebelegung zur Lösung der Gleichungen den Modelica-Klassen zugeordnet, welche die grundlegende Struktureinheit repräsentieren. Aus einem solchen Modelica-Modell lässt sich Quellcode generieren, mit dem das Systemverhalten simuliert werden kann. Zur Modellierung mit der UML existiert das Modelica Modeling Language (ModelicaML)-Profil [Sch09], aus dem direkt der ausführbare Modelica-Quellcode erzeugt werden kann.

[68] https://www.modelica.org/

Der Ablauf ähnelt dem des Validierungsprozesses, was nicht weiter verwunderlich ist, da dies einem typischen Testablauf - Vorbereitung, Durchführung, Auswertung - entspricht. Durch den Einsatz von Modelica liegt der Fokus auf der Beschreibung der physikalischen Komponenten und deren Verhalten. Der in der Arbeit vorgestellte Validierungsprozess schließt auch die funktionalen Komponenten, die Software des Systems, und deren Zuordnung zur Hardware mit ein. Das Verhalten, also die dahinter liegenden Algorithmen, wird nicht im Modell beschrieben. Es wird, falls die Entwicklungsphase dies bereits zulässt, in den Validierungszielverfahren definiert. Darüber hinaus gibt es Unterschiede bei den formalisierten Anforderungen, den Datenverknüpfungen und den im Test durchgeführten Simulationen. Der Validierungsprozess verwendet keine formalisierten Anforderungen auf Basis von Modelica. Die in den Anforderungen für die Überprüfung ermittelten quantitativen Werte werden mit den Validierungszielen festgehalten, welche selbst unabhängig von spezifischen Werten definiert werden. Die Validierungsziele werden wiederum mit Validierungszielverfahren verknüpft, die keine Abhängigkeit zum Design-Modell besitzen. Eine Verbindung existiert nur aus dem vom Design-Modell abgeleiteten Validierungsmodell, in dem die für die Validierungszielverfahren benötigten Daten enthalten sind. Diese Daten beeinflussen natürlich die validierungsspezifische Notation. Die Notation basiert aber nicht auf einem bestimmten UML-Profil, wie z. B. der vVDR-Ansatz auf ModelicaML, sondern wird anhand der betrachteten Validierungsziele mit Hilfe des UML-Profilmechanismus erstellt. Die Erstellung des Profils ist damit Teil des Validierungsprozesses. Dadurch ergibt sich ein größeres Anwendungsgebiet (über die Modellierung physikalischer Systeme hinaus) und es lässt dem Architekten mehr Freiheit bei der Notationswahl, die wiederum von der im Projekt verwendeten Entwicklungsnotation und den gewählten Validierungszielverfahren abhängt.

5.3.2 Holtmann

Holtmann et al. beschreiben einen Ansatz zur Verbesserung des System-Entwicklungsprozesses für den Automotive-Bereich [HMM11] [PHAB12]. Der im Rahmen des Projektes Software Platform Embedded Systems (SPES) 2020 [TU 12] entwickelte Ansatz versucht, die Prozess- und Werkzeuglücke zwischen der System-Architektur und der Software-Architektur zu schließen. Abbildung 5.16 zeigt die beteiligten Artefakte und die vom Ansatz unterstützten Übergänge vom *System-Analyse-Modell* bis zum *Software-Design*. Zusätzlich sind noch die verwendeten Modellierungselemente der System Modeling Language (SysML) [Obj10] dargestellt, auf die an dieser Stelle nicht näher eingegangen wird. Ausgangspunkt ist das *System-Analyse-Modell*, das eine funktionale Sicht auf das System bereitstellt. Der Architekt entwickelt daraus manuell das System-Design inklusive der Architektur. Dieses Architekturmodell bietet eine logische und technische Sicht auf das System und enthält bereits detaillierte Informationen über die verwendeten Betriebssysteme auf den Electronic Control Units (ECUs). Mit Hilfe von Real Time Statecharts (RTSCs)[69] wird das Verhalten modelliert, da die SysML diesbezüglich keine Modellierungsmöglichkeiten anbietet. Aus dem Architekturmodell wird mit Hilfe einer Modell-zu-Text-Transformation das *System Simulation Model* erstellt, welches für das Echtzeit-Simulationswerkzeug chronSIM[70] zur Verifizierung des Architekturmodells benötigt wird. Falls Daten für dieses Modell im Architekturmodell fehlen, werden diese über den UML-Profilmechanismus hinzugefügt. Auf den Abgleich der Simulationsergebnisse mit den Anforderungen wird nicht eingegangen. Das Software-Design kann mit Hilfe von Modelltransformationen direkt aus dem Archi-

[69] Bei den RTSCs handelt es sich um ein formales Verfahren basierend auf Timed Automatas [BK08, Kapitel 9] [Bér01, Kapitel 5].

[70] http://www.inchron.de/ chronsim.html

tekturmodell erstellt werden. Es besteht aus drei Schichten, die sich durch die Verwendung von AUTomotive Open System ARchitecture (AUTOSAR) ergeben[71]. Aus dem Architekturmodell werden sowohl Daten für die *Application Software*-Schicht als auch für die *Runtime Environment*-Schicht transformiert. Die Daten aus der Application Software-Schicht können von AUTOSAR-Simulationen verwendet werden.

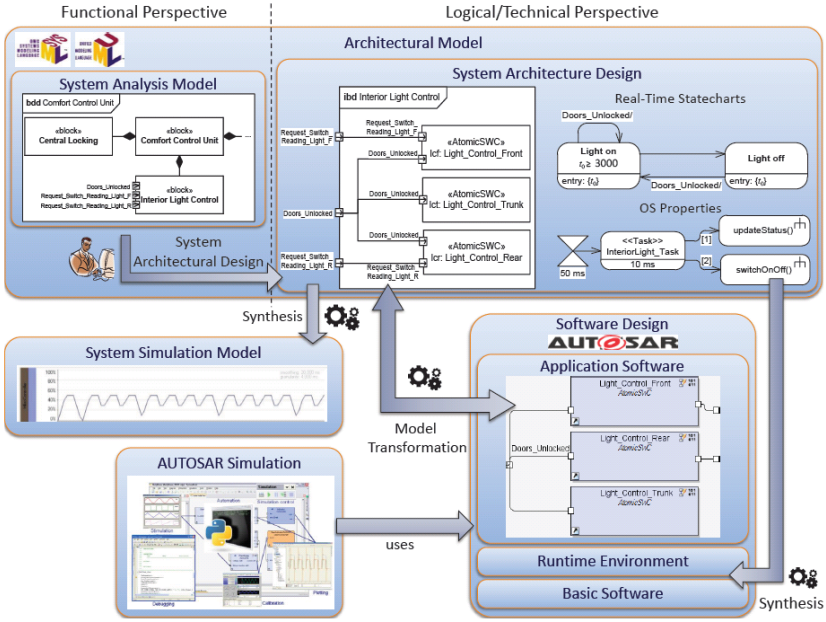


Abbildung 5.16: Das System-Architektur-Design und seine Erweiterungen inkl. der Übergänge zum Software-Design, nach [HMM11]

Anforderungen werden mit Hilfe von *requirements pattern* beschrieben, welche die Aussagekraft der natürlichen Sprachen einschränken, eindeutig ma-

[71] AUTOSAR ermöglicht die Entwicklung von Software unabhängig von der verwendeten ECU.

chen und eine automatisierte Auswertung ermöglichen [Hol10]. Die auf diese Weise formalisierten Anforderungen werden zur Herstellung der Nachverfolgbarkeit mit den Modellelementen verbunden.

Ähnlich wie Schamai et al. (siehe Kapitel 5.3.1) verwendet auch dieser Ansatz formalisierte Anforderungen und eine formale Prüftechnik für die Verifizierung des Modells gegenüber den Anforderungen. Beide Ansätze werden nicht für datenintensive Systeme verwendet, weshalb das State Explosion-Problem (siehe Kapitel 5.1.2, Seite 118) kein oder nur ein geringes Problem für die eingesetzte formale Prüftechnik darstellt. Probleme mit fehlenden Informationen über Eigenschaften des Betriebssystems[72], die verwendete physikalische Architektur bzw. das Auswechseln von Hardware-Komponenten im Laufe der Entwicklung gibt es aufgrund von Sicherheitsbestimmungen sowie des ausgeprägten Produktlinienansatzes mit standardisierten Hardware-Komponenten kaum. Der Fokus auf die Automotive-Domäne verhindert allerdings eine Verwendung oder Übertragung des Ansatzes auf andere technische Domänen, vor allem datenintensiver Systeme. In anderen technischen Domänen kann auch nicht von einer standardisierten physikalischen Infrastruktur ausgegangen werden. Vielmehr müssen heterogene und sich im Verlauf der Entwicklung ändernde Komponenten bei der Verifizierung des Architektur gegenüber den Anforderungen berücksichtigt werden. Holtmann et al. beschreiben auch kein Vorgehen für den Umgang mit den verschiedenen architekturspezifischen Aspekten mit ihren Abhängigkeiten untereinander und zu den architekturrelevanten Anforderungen. Es wird zwar eine Nachverfolgbarkeit zwischen System-Komponenten und Anforderungen hergestellt, dies ist aber nur eine schwache Unterstützung für den Architekten bei der Impact-Analyse von Anforde-

[72] Holtmann geht davon aus, dass es im Automotive-Bereich keine dynamischen Bereiche des Betriebssystems gibt. Alle Eigenschaften sind statisch.

rungsänderungen auf die Architektur und umgekehrt. Fehlende Daten für die Verifizierung werden mit Hilfe des UML-Profilmechanismus direkt im Architekturmodell modelliert. Hierdurch ergibt sich das Problem der Vermischung von Entwicklungs- und Verifizierungsdaten, das durch die Verwendung eines eigenen Modells vermieden werden sollte (siehe Kapitel 8.3).

5.3.3 SAVI

Das vom Aerospace Vehicle Systems Institute initiierte Projekt System Architecture Virtual Integration (SAVI) [FWH10] umfasst drei Schlüsselkonzepte: ein Referenzmodell, ein Modell-Repository mit Modell-Bus sowie einen modellbasierter Entwicklungsprozess. Das Konzept ermöglicht auf das Referenzmodell abgebildete Modelle, z. B. auf Basis von UML, Architecture Analysis & Design Language (AADL) oder SysML, mit in einem Repository hinterlegten Anforderungen zu vergleichen. Damit ließe sich zwar theoretisch eine Validierung der System-Architektur gegenüber Anforderungen durchführen, der Mehraufwand wäre aber sehr groß, da das Projekt eine deutlich andere Zielsetzung hat. Als Kurzfassung lautet diese: "Integrate, then Build" [RWCP10, Seite 57]. Der Ansatz ist zudem noch in der Entwicklung. Es existieren ein Machbarkeitsnachweis [FWH10] und eine Schätzung der Rentabilität [WH11].

5.3.4 SAAM und ATAM

Im Bereich der Software-Architekturen gibt es szenariobasierte Bewertungsverfahren. Die zwei grundlegenden Verfahren sind die SAAM und die ATAM. Sie werden in [CKK02] beschrieben. Basierend auf diesen Verfahren gibt es einige Verfeinerungen, die beispielsweise auf bestimmte Klassen von Systemen spezialisiert sind. Einen guten Überblick gibt hier [IHO02].

Durch die Verfahren lässt sich herausarbeiten, wie gut bestimmte Architekturqualitätskriterien durch die zu testende Architektur erfüllt werden. Die zu betrachteten Qualitätskriterien werden zunächst mit Hilfe qualitativer oder quantitativer Untersuchungsmethoden beurteilt und anschließend anhand von Prioritäten auf die wichtigsten reduziert. Basierend auf diesen Ergebnissen lässt sich dann eine Architektur aus der Menge der untersuchten Architekturen für die Weiterentwicklung auswählen. Die Szenarien werden beispielsweise anhand der Anforderungen oder in Zusammenarbeit mit den Stakeholdern erarbeitet. Die Verfahren beschreiben die einzelnen Bewertungsschritte, geben aber keine Dokumentationsform für die Architektur oder bestimmte Analyse-Verfahren für die Evaluierung vor. Auf die Eignung von UML/SysML als Architecture Description Language (ADL) wird in [CKK02] hingewiesen. Die Grundideen der szenariobasierten Verfahren lassen sich auf die Evaluierung von System-Architekturen übertragen, obwohl sie auf Software-Architekturen ausgerichtet sind. Dies zeigt auch eine ATAM-Variante für System-Architekturen, wobei hierfür noch Organisationen gesucht werden, die das Verfahren in ihrem Entwicklungsprozess einsetzen möchten [Sof13]. Der Validierungsprozess ist allerdings nicht auf die Bewertung von Architekturvarianten hin ausgerichtet, sondern auf die Prüfung, ob die erstellte Architektur den System-Anforderungen gerecht wird und welche Auswirkungen Anforderungsänderungen auf die Architektur haben. Dies könnte als Spezialform der szenariobasierten Verfahren gesehen werden. Allerdings geht der Validierungsprozess ein Stück weiter und zeigt dem Architekten, **wie** Dokumentation und Analyse der Architektur mit Hilfe von UML und Simulationen durchgeführt werden können.

5.3.5 Aydal

In [APUW09] beschreiben die Autoren einen Ansatz zum Testen von Design-Modellen und Quellcode auf Basis eines eigenen Test-Modells. Neben dem aus den Anforderungen entstehenden Design-Modell erstellen sie ein separates Test-Modell. Die Testfälle zur Überprüfung des Design-Modells und des Quellcodes werden nicht aus dem Design-Modell abgeleitet, wodurch darin enthaltene Fehler, struktureller Art oder im Verhalten, sich nicht auf die Testfälle übertragen und damit potenziell unentdeckt bleiben. Abbildung 5.17 zeigt diese Trennung der Zuständigkeiten. Zur Generierung von Testfällen werden zunächst die Anforderungen mit Hilfe der Alloy Modelling Language [Jac12] modelliert. Mit Hilfe von Testauswahlkriterien werden daraus Testfall-Spezifikationen und anschließend abstrakte Testfälle erzeugt, die mit Hilfe von Adaptern in konkrete Testfälle transformiert werden. Auf die beschriebenen Details zum modellbasierten Testen wird an dieser Stelle nicht näher eingegangen, da sie für die Abgrenzung nicht von Bedeutung sind. In der beschriebenen Fallstudie werden Testfälle für ein in der formalen Spezifkationssprache Z modelliertes Design-Modell erstellt.

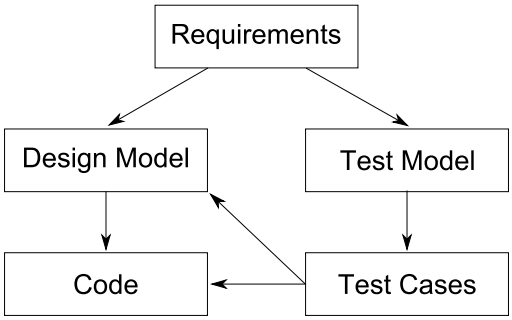


Abbildung 5.17: Beziehungen zwischen Anforderungen, Modell, Quellcode und Testfällen, nach [APUW09]

Ein Adapter für die Generierung von Testfällen für die UML ist noch nicht vorhanden, der Ansatz unterstützt durch die Verwendung von Adaptern aber grundsätzlich die Verwendung anderer Modellierungssprachen. In der im Paper vorgestellten Fallstudie liegt der Fokus auf dem Testen funktionaler Anforderungen; nicht-funktionale Anforderungen werden bei der Erstellung der Testfälle nicht berücksichtigt, wodurch eine Überprüfung der Architektur nur bedingt möglich ist.

Leider gibt es keine explizite Aussage der Autoren zur Unterstützung nicht-funktionaler Anforderungen durch den vorgestellten Ansatz oder deren Berücksichtigung in weiteren Fallstudien. Der Validierungsprozess verwendet auch zwei Modelle zur Datentrennung: das Entwicklungsmodell und das Validierungsmodell. Aus dem zuletzt genannten werden die für die Validierungszielverfahren benötigten Eingabedaten ausgelesen. Das Ziel dieser Trennung ist die Unabhängigkeit der Validierungsinformationen von den Änderungen in der Entwicklung und eine validierungspezifische Sicht auf die Architektur, so dass für die Validierung irrelevante Informationen nicht die Übersicht erschweren. In [APUW09] ist das Ziel der Trennung eine vom Design-Modell unabhängige Generierung von Testfällen. Das Validierungsmodell basiert nicht auf den Anforderungen, sondern auf dem Entwicklungsmodell, um die bei der System-Entwicklung modellierten Informationen nicht redundant modellieren zu müssen und Änderungen am Validierungsmodell auch wieder in das Entwicklungsmodell transformieren zu können.

5.3.6 PREEVision

Das Werkzeug PREEvision von aquintos[73] [BH12] unterstützt eine modellbasierte Entwicklung und Evaluierung von elektrischen und elektronischen Architekturen. Die Modellierungssprache basiert auf einer graphischen Notation, die auf einem eigenen domänenspezifischen Metamodell basiert. Architekturrelevante Informationen werden über ein Datenmodell bereitgestellt. Einen ähnlichen Ansatz beschreiben Adler et al. in [AGMG11]. Dabei handelt es sich um eine modellbasierte Konsistenzüberprüfung der elektrisch und elektronischen Architektur gegenüber den Anforderungen. Für die Modellierung wird die Electric and Electronic Architecture (EEA)-ADL verwendet. Sowohl PREEvision als auch der Ansatz von Adler et al. decken nur einen Teilbereich des von dem in der Arbeit vorgestellten Validierungsprozesses ab. Der Fokus liegt bei beiden Ansätzen auf der physikalischen Architektur. Eine Validierung des gesamten Systems, inklusive der Software, ist nicht möglich. PREEvision setzt zudem auf ein eigenes Metamodell auf, das für eine Verwendung im Validierungsprozess einen Transformationsaufwand erzeugen würde. Der Ansatz von Adler et al. ließe sich als statisches Verfahren zur Überprüfung der Modellkonsistenz verwenden, was vor der Ausführung der Validierungszielverfahren stattfinden würde.

5.3.7 Matlab Simulink

Das Werkzeug Matlab Simulink[74] ermöglicht die grafische Modellierung und Simulation von Schaltungen und Regelungen. Es lassen sich damit auch Signalverarbeitungsketten und Steuergeräte entwickeln, wobei hier vor allem die Simulationsmöglichkeit und die damit verbundene Codegenerie-

[73] http://www.aquintos.com/

[74] http://www.mathworks.de/products/simulink/

rung genutzt wird. Auf diese Weise können Algorithmen vor einer konkreten Implementierung für die Zielhardware überprüft werden. Die Modellierung basiert auf Blockschaltbildern, wobei die Blöcke Stellvertreter für verschiedene mathematische Funktionen sind. Simulink bietet mit verschiedenen Zusatzkomponenten die Möglichkeit der Nachverfolgbarkeit von Anforderungen ins Modell (Verification and Validation), die Modellierung und Simulation von physikalischen Systemen (Simscape) sowie die Verifizierung von Designeigenschaften mit Hilfe formaler Methoden (Design Verifier) an. Um die Vorteile von Simulink, die Simulation des modellierten Systems, nutzen zu können, sind detaillierte Informationen über die fachlichen Algorithmen und die physikalischen Bausteine erforderlich, wodurch ein Einsatz in einer frühen Entwicklungsphase nicht in Frage kommt. Die Ausdruckskraft der grafischen Modellierung in Simulink ist geringer im Vergleich mit der UML. Modellierungselemente wie beispielsweise Use Cases und Aktivitäten zur Beschreibung der Systemfunktionalitäten und abstrakter Abläufe, Sequenzdiagramme zur Beschreibung des Kommunikationsverhaltens zwischen Komponenten oder allgemeine Komponenten für die Beschreibung einer lösungsneutrale Systemstruktur existieren nicht. Die Verifizierung von Designentscheidungen erfolgt durch eine formale Methode mit den in Anhang 5.1.2 genannten Nachteilen, insbesondere für datengetriebene Systeme. Die durch den in der Arbeit vorgestellten Ansatz entstehende System-Architektur kann vielmehr als Grundlage zur Erstellung eines Simulink-Modells dienen, was den Übergang vom Design zur Realisierung ermöglicht.

5.3.8 Debbabi

Debbabi et al. [DHJ+10] beschreiben einen Ansatz, der auf das Zusammenwirken von Validierungs und Verifizierungsmethoden setzt. Abbildung 5.18 [DHJ+10, Seite 98] zeigt den Ansatz im Überblick.

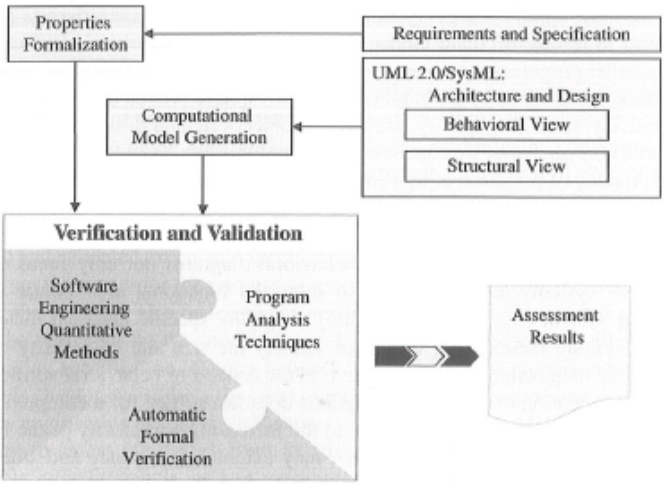


Abbildung 5.18: Ansatz zum Zusammenwirken von Validierung und Verifizierungsmethoden, nach [DHJ+10]

Kernstück des Ansatzes sind die drei als Puzzleteile dargestellten Verfahren: automatische formale Verfahren, Programmanalyse-Techniken und quantitative Metriken für die Softwareentwicklung. Mit der Kombination von formalen, dynamischen und statischen Verfahren (siehe Kapitel 2.8) wird auf die Synergieeffekte gesetzt, so dass für die Auswertungsergebnisse mehr als die Summe der einzelnen (Puzzle-) Teile herauskommt. Aus den Anforderungen und Spezifikationen werden formalisierte Eigenschaften in Form von temporalen Logiken erarbeitet, die als Eingabe für das formale Ver-

fahren, genauer Model-Checking und probabilistisches Model-Checking, benötigt werden. Das Verhalten und die Struktur des Systems werden modellbasiert mit Hilfe der UML und SysML dokumentiert. Um diese Daten für die Validierungs- und Verifizierungsmethoden nutzen zu können, müssen sie in ein geeignetes Modell, dem Configuration Transition Model (CTS), transformiert werden. Diese Transformation wird als Computational Model Generation bezeichnet. Mit Hilfe des Model-Checkings und der Programmanalyse-Techniken wird das Verhalten überprüft, wobei die Programmanalyse-Techniken als Unterstützung für das Model-Checking verwendet werden. Bei großen Systemen droht das State-Explosion-Problem, was durch die geeignete Wahl der zu untersuchenden Systemgröße vermieden werden kann. Diese Abstraktion des zu untersuchenden Systems kann mit Hilfe von graphbasierten Daten- und Kontrollflussanalysen erreicht werden. Die Metriken werden sowohl auf die modellierte Struktur als auch das Verhalten angewendet. Sie dienen zum einen zur Ermittlung der Qualität des modellierten Designs und zum anderen zum Finden von Fehlern in der Semantik oder im Modell. Beispiele sind hier die Länge des kritischen Pfades zur Ermittlung eines minimalen Zeitverzugs bis zur Ausführung einer Funktion und der Komplexitätsvergleich eines Verhaltensdiagramms gegenüber dem CTS-Modell zum Finden redundanter oder bedeutungsloser Elemente (z. B. unerreichbare Zustände). Daraus ergibt sich insgesamt das Ergebnis der Auswertung von Anforderungen, Spezifikation und der modellbasierten Dokumentation.

Debbabi et al. nutzen die Stärken der verschiedenen Verfahren, indem sie diese gemeinsam für die Validierung und Verifizierung einsetzen. Der Fokus bei der Wahl der Verfahren liegt auf den formalen Verfahren, die zwar gut etabliert sind, aber trotzdem nur selten in der Industrie eingesetzt werden [WLBF09]. Model-Checking ist aufgrund des State-Explosion-Problems

gerade für datenintensive Systeme nicht geeignet. Weitere Details hierzu sind in Kapitel 5.1.2 beschrieben. Die Bewertung des modellierten Designs bzw. der Architektur ist nicht Teil des in der vorliegenden Arbeit beschriebenen Ansatzes. Neben Metriken ist hierfür auch der Einsatz von ATAM (siehe Kapitel 5.3.4) denkbar.

5.3.9 Hall und Iqbal

Weitere Ansätze für die Validierung gibt es im Bereich der Realzeit-Systeme. In [Hal08] liegt der Fokus auf Interrupt-getriebenen Realzeit-Systemen, deren Validierung mit Hilfe einer Real-Time Event Queue ermöglicht wird. Dafür wird der zeitliche Verlauf von Ereignissen durch die explizite Modellierung einer der Zeit nachgeordneten Queue modelliert. Die Ereignisse werden dann auf eine reaktive System-Instanz angewendet. Die mathematischen Modelle können der Queue neue Ereignisse hinzufügen, um beispielsweise einen zukünftigen Interrupt zu planen. Der Ansatz ist auf die Validierung reaktiver Real-Zeit-Systeme spezialisiert und basiert auf einem formalen mathematischen Modell. Der Ansatz stellt keinen Validierungsprozess zur Verfügung und es fehlt die Unterstützung für den Architekten bei der Impact-Analyse. Dies gilt auch für den in [IAB10] beschriebenen Ansatz. Er erlaubt die Modellierung der System-Umgebung auf Basis von UML und dem Modeling and Analysis of Real Time and Embedded systems (MARTE)-Profil für eingebettete Realzeit-Systeme, um Black-Box-Tests des Systems durchführen zu können. Bei der Validierung der System-Architektur sollen aber gerade die inneren Komponenten des Systems und deren Zusammenspiel überprüft werden. Die Modellierungsmöglichkeiten für die System-Umgebung können aber für das Validierungsmodell genutzt werden, falls ein Validierungszielverfahren diese Informationen benötigt.

5.3.10 MADES

Das Model-based methods and tools for Avionics and surveillance embeddeD SystEmS (MADES)-Projekt[75] [QBG+12] hat sich mit der Verbesserung des in der Praxis angewendeten Vorgehens bei der Entwicklung eingebetteter Systeme im Bereich der Avionik sowie der Boden-/See-/Luft-Überwachung beschäftigt. Hierzu wurde eine Auswahl des von der OMG definierten MARTE-Profils zur Modellierung aller notwendigen Informationen, die nicht bereits durch die SysML oder UML abgedeckt sind, erarbeitet. Die Überprüfung der System-Architektur erfolgt durch eine formale Prüftechnik, das Model-Checking. Die hierfür benötigten Informationen werden mit Hilfe einer graphischen UML-Modellierungsnotation der System-Dokumentation hinzugefügt. Dadurch soll die Benutzungsfreundlichkeit verbessert werden, die für den Model-Checker erforderliche LTL vor dem Anwender verbergen und den Einarbeitungsaufwand des Modellierers in die temporale Logiken verringern [MAD11] [RMP+12]. Als Informationsquelle werden verschiedene Diagramme verwendet, beispielsweise das Interaktionsdiagramm [BMMR10] und das Zustandsdiagramm [ZBR13]. Zur Verifzierung werden die Informationen auf Basis der in Tempo Reale ImplicitO (TRIO) formulierten formalen Semantik des Interaktionsdiagramms in eine für den Modell-Checker Zot [Pra10] geeignete Form transformiert. Zot ist in Common Lisp geschrieben, unterstützt verschiedene Logik-Sprachen als Eingabe und erlaubt, die Erfüllbarkeit von Aussagen durch Pfadauswertungen zu prüfen. Das MADES-Projekt ist auf eingebettete Systeme spezialisiert und bevorzugt formale Prüftechniken. Der Ansatz, die dafür notwendigen Informationen benutzerfreundlich im Modell angegeben zu können, sieht vielversprechend aus. Er reicht aber noch nicht aus, um die formale Prüftechnik vollständig vor dem Benutzer zu verbergen, um so die Akzep-

[75] http://www.mades-project.org/

tanz zu verbessern. MADES bietet zudem keine Unterstützung für den Architekten bei der Impact-Analyse von Änderungen an der Architektur oder an den Anforderungen.

5.3.11 Vergleich des Ansatzes mit ausgesuchten Arbeiten

Zur besseren Übersicht sind in Tabelle 5.1 ausgesuchte Ansätze zusammen mit dem Validierungsprozess aufgeführt. Dabei sind die verschiedenen Ausprägungen bezüglich der Kriterien verwendete Dokumentationsart, eingesetzte Prüftechnik, Unterstützung bei der Impact-Analyse und Prüffokus zusammengefasst.

5.4 Potenzielle Simulationen

Für die Überprüfung der Validierungsziele werden bei dem in der vorliegenden Arbeit beschriebenen Ansatz Simulationen eingesetzt. Hierfür können selbst implementierte oder auch vorhandene Simulationen verwendet werden. Eigene Implementierungen bieten den Vorteil, dass sie den projekt- und domänenspezifischen Bedürfnissen und den zur Verfügung stehenden Informationen angepasst werden können. Das Systemverhalten lässt sich auf einem hohen Abstraktionsgrad und damit bereits in einer frühen Entwicklungsphase simulieren. Dabei kann das Detaillevel auch innerhalb der Architektur (z. B. für einzelne Software-Komponenten) individuell festgelegt werden. Wie so eine selbst implementierte Simulation ausssehen kann, ist in Kapitel 7.2 beschrieben. Mit fortschreitender Entwicklung und steigendem Wissen über das System lassen sich auch spezialisierte Systemsimulation verwenden. Sie können entweder als eigenständige Validierungszielverfahren oder eingebettet in einem selbst implementierten Verfahren

Tabelle 5.1: Vergleich des Validierungsprozesses mit ausgesuchten Ansätzen

Name	**Dokumentationsart**	**Prüftechnik**	**Unterstützung bei der Impact-Analyse**	**Prüffokus**
Validierungsprozess	UML + Profilmechanismus	dynamisch	X	System-Architekturen
vVDR (Schamai)	ModelicaML	formal	X	physikalische Architekturen
Holtmann	UML + Profilmechanismus + SysML + RTSC	formal	-	AUTOSAR
PREEvision	eigenes Metamodell	formal	-	elektrisch / elektronische Architekturen
Simulink	Blockschaltbilder	formal	-	physikalische Architekturen
Debabbi	UML + SysML	formal	-	System-Architekturen
Hall/Iqbal	UML + MARTE	formal	-	Realzeit-Systeme
MADES	UML + MARTE	formal	-	eingebettete Systeme

verwendet werden. Im Folgenden werden drei potenzielle Simulationen vorgestellt.

5.4.1 Weiss

Weiss et al. beschreiben einen Ansatz [WZEK10] zur simulationsbasierten Validierung von eingebetteten Automotive-Systemen mit Hilfe von Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) [EAS10] und SystemC[76]. EAST-ADL erlaubt die Modellierung der Architektur auf unterschiedlichen Abstraktionsebenen, zu denen auch das Design-Level und damit die System-Architektur gehört. SystemC ist eine standardisierte Modellierungs- und Simulationssprache für das Co-Design von Hardware und Software. Sie ist eine Bibliothek für die Sprache C++ und ermöglicht unter anderem die Simulation nebenläufiger Prozesse, komplexer Kommunikationsstrukturen und von Hardware-Eigenschaften. Letzteres wird über den Register Transfer Level ermöglicht, wodurch SystemC sich als Ersatz für Hardwarebeschreibungssprachen wie beispielsweise VHDL einsetzen lässt. SystemC unterstützt den Transaction Level Modeling (TLM)-Ansatz [CG03], wodurch einerseits die Kommunikation und Berechnungskomponenten voneinander getrennt und andererseits deren Details in frühen Entwicklungsphasen, d. h. auf hohen Abstraktionsleveln, versteckt und später hinzugefügt werden können. Diese Simulation könnte als ein Validierungszielverfahren eingesetzt werden, wobei die EAST-ADL der validierungszielspezifischen Notation entspräche. Hier wäre zu überprüfen, ob wirklich alle Modellierungsmöglichkeiten der EAST-ADL benötigt werden oder eine Submenge in einem eigenen UML-Profil verwendet werden sollte. Die SystemC-Bibliothek kann auch unabhängig von diesem Ansatz für die Erstellung eines Validierungszielverfahrens verwendet werden. Der

[76] http://www.accellera.org

TLM-Ansatz kommt dem iterativen Design-Prozess entgegen und erlaubt somit eine frühe Validierung der Architektur gegenüber den Anforderungen.

5.4.2 Gray

Gray et al. [GA11] [IQG+12] beschreiben einen Ansatz zur Verteilung von Java-Anwendungen auf verschiedene Systeme, ohne dass eine spezielle architekturspezifische Anpassung im Quellcode erforderlich ist. Die Software wird so implementiert, als wenn die Anwendung auf einem System ausgeführt wird. Die Verteilung der Software auf mehrere Systeme und die dafür benötigte Kommunikation werden durch Veränderungen am Java-Bytecode der Anwendung ermöglicht. Dieses Verfahren kann als Erweiterung von Simulationen verwendet werden, um eine verteilte Verarbeitung von Algorithmen (vertikale Schneidung, siehe Kapitel 7.2.3.2) zu ermöglichen.

5.4.3 CoFluent Studio

Das CoFluent Studio[77] [RP10] ist ein Werkzeug zum Testen der System-Architektur in einer virtuellen Umgebung. Für die Dokumentation werden UML-Modelle inklusive der Profile SysML und MARTE eingesetzt. Diese Modelle werden in ausführbare SystemC-Transaction-Level-Modelle transformiert. Das Konzept ist vergleichbar zu Executable UML [MB02], hat aber eine andere Zielsetzung gegenüber dem Validierungsprozess. CoFluent Studio verifiziert die modellierten Informationen durch eine Simulation in einer speziellen virtuellen Umgebung und könnte damit als Validierungszielfahren zum Prüfen eines Validierungsziels verwendet werden, es ermöglicht

[77] http://www.cofluentdesign.com

aber weder die Validierung der System-Architektur als Ganzes, noch unterstützt es den Architekten bei der Impact-Analyse.

5.5 Modellierungssprachen

Für einige Domänen existieren eigene UML-Modellierungssprachen, mit denen beispielsweise eine projekt- und unternehmensübergreifende Modellierung von Systemen möglich ist. Sie setzen sich aus den vorhandenen Elemente der UML oder anderen Modellierungssprachen sowie neu definierten Elementen und Diagrammen zusammen, um den domänenspezifischen Anforderungen an die Modellierung gerecht zu werden. Keine der Sprachspezifikationen stellt eine konkrete Prozessbeschreibung zur Validierung von Architekturen bereit. Der Architekt kann sie aber sowohl zur Modellierung von Informationen im Entwicklungsmodell als auch im Validierungsmodell verwenden. Es besteht kein Zwang ein neues Profil zu erstellen, falls die benötigten Modellierungselemente (Stereotypen und TaggedValues) bereits existieren und zur Verfügung stehen. Der Validierungsprozess macht hier keinerlei Einschränkungen und überlässt dem Architekten die Entscheidungen, mit welchen Elementen er die validierungsspezifischen Informationen modelliert. Im Folgenden werden für ausgewählte UML-Erweiterungen die anvisierte Domäne und einige Besonderheiten beschrieben.

5.5.1 SysML

Die System Modeling Language (SysML) [Obj10] ermöglicht das modellbasierte Spezifizieren, Analysieren, Designen und Testen von komplexen Systemen. Sie besteht aus Elementen und Diagrammen, die zum Teil aus der UML wiederverwendet werden und zum Teil neu definiert wurden. Sie

wird von guten Modellierungswerkzeugen standardmäßig zur Verfügung gestellt. Eine der Neuerung der SysML ist das Anforderungsdiagramm und das Anforderungselement, mit deren Hilfe textuelle Anforderungen modelliert und direkt mit anderen Modellelementen verknüpft werden können.

5.5.2 AADL

Die Architecture Analysis & Design Language (AADL) [Car09] dient zur Analyse und zum Design von Architekturen für eingebettete Echtzeitsysteme. Bei den Architekturen kann es sich sowohl um System- als auch um Software-Architekturen handeln. Im Fokus steht die Analyse der Systemstruktur und des Laufzeitverhaltens an Stelle des funktionalen Verhaltens. Die AADL ist eine textuelle Spezifikation, für die ein UML-Profil zur Verfügung steht.

5.5.3 EAST-ADL

Die Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) [EAS10] ist eine Modellierungssprache für eingebettete elektronische Systeme der Automotive-Domäne. Sie basiert auf Konzepten der UML, SysML und der AADL und ermöglicht die Modellierung von Informationen als Ergänzung für AUTOSAR. Die Verwaltung erfolgt durch die EAST-ADL-Association. Für die Verwendung in UML-Modellen steht ein Profil zur Verfügung.

5.5.4 MARTE

Modeling and Analysis of Real Time and Embedded systems (MARTE) [Obj11b] ist eine von der OMG standardisierte Modellierungssprache zur

Modellierung von Echtzeit- und eingebetteten Systemen auf Basis der UML. Der Fokus der Analyse liegt auf der Systemleistung und den zeitlichen Abläufen. MARTE bietet eine hohe Anzahl von Stereotypen zur Modellierung der unterschiedlichsten Informationen an. Die Verbreitung und Anwendung in der Praxis gestaltet sich allerdings aufgrund des Spezifikationsumfangs als schwierig. Das MADES-Projekt [QBG+12] beschäftigt sich deshalb mit der Aufgabe die Anwendbarkeit des MARTE-Profils für die Industrie zu verbessern, was sich als größere Herausforderung herausgestellt hat (siehe auch Kapitel 5.3.10).

6 Praktische Anwendung des Validierungsprozesses

Dieses Kapitel beschreibt die Anwendung des Validierungsprozesses auf ein Beispiel aus der Domäne Radar-Systeme, das von einem realen Projekt hergeleitet wurde. Kapitel 6.1 stellt hierzu einige Informationen bereit und beschreibt die gesammelten Erfahrungen und Herausforderungen für den Architekten. Kapitel 6.2 stellt die Anforderungen vor und beschreibt Auszüge des Entwicklungsmodells für das verwendete Radar-System, bevor Kapitel 6.3 sich der Durchführung des Validierungsprozesses widmet und zeigt, wie der Validierungsprozess den Architekten bei den Herausforderungen unterstützt.

6.1 Projektbezug und Projekterfahrungen

Das in diesem Kapitel vorgestellte Beispiel basiert auf zwei realen Entwicklungsprojekten. Die Namen der Software-Komponenten sowie der verwendeten Hardware sind verfremdet bzw. entsprechen allgemein verwendeten Fachbegriffen. Die vorgestellten Änderungen an den Anforderungen und der Architektur sowie die auftretenden Probleme basieren auf den Erfahrungen verschiedener Projekte. Zur Einordnung der Projektgrößen kann erwähnt werden, dass die Anforderungsspezifikation etwa 500 bzw. 2000 System-Anforderungen umfasst und die Architektur aus bis zu 80 Software-Komponenten und bis zu 40 Verarbeitungseinheiten besteht. Hinzu kommen noch Komponenten aus dem Bereich der Kommunikation sowie detaillierte Hardware-Beschreibungen. Im nachfolgenden Beispiel werden etwa 15 Software-Komponenten und 11 Verarbeitungseinheiten verwendet. Die-

se Reduzierung ist aufgrund der Übersichtlichkeit notwendig. Der Detaillierungsgrad des Beispiels orientiert sich daran, welcher Grad für ein detailliertes Verständnis des Validierungsprozesses erforderlich ist.

Eine der größten Herausforderungen für den Architekten bei der Validierung der Architektur sind die späten Anforderungsänderungen durch den Kunden. Solche Änderungswünsche finden nicht nur während der Anforderungsanalyse, sondern auch noch während des System-Designs statt. Bei den Änderungen handelt es sich sowohl um funktionale, als auch um nichtfunktionale Aspekte wie beispielsweise die Veränderungen des Energieverbrauchs, des Detaillierungsgrads einzelner Algorithmen, des Projektbudgets oder auch der Aufbewahrungsfrist der verarbeiteten Radardaten. Der Architekt ist dafür verantwortlich, die Auswirkungen solcher Änderungen im Bezug auf die Architektur zu analysieren (engl. *impact analysis*) und im Optimalfall eine valide Architektur zu erstellen. Falls dies nicht für alle Anforderungen möglich ist, ist es für das Gespräch mit dem Kunden hilfreich zu wissen, welche Veränderungen möglich sind und unter welchen Bedingungen die restlichen Änderungswünsche realisiert werden können.

Neben diesen kundenspezifischen Änderungen ergibt sich durch die stetig fortschreitende Entwicklung im Hardware-Bereich die Möglichkeit, neue Hardware-Komponenten einzusetzen. Sie sind nicht unbedingt nur wegen einer verbesserten Leistung interessant, sondern stellen u. a. auch aufgrund der unterstützten Kommunikationsmedien, des Energieverbrauchs, der Haltbarkeit oder des Preises für den Architekten eine Alternative zum vorhandenen Hardware-Katalog dar. Für die Erarbeitung von Alternativen ist im Projektalltag allerdings meist nur wenig Zeit eingeplant und der Aufwand für eine vollständige Validierung einer Architektur ist hoch. Hier wäre bereits eine schnelle, einfache Validierung hilfreich, deren Ergebnisse dem

Architekten eine grobe Einschätzung ermöglichen. Dafür müssen die verwendeten Berechnungen zum Testen der Anforderungen nicht sehr detailliert sein. Vielmehr ist es entscheidend, dass alle architekturspezifischen Aspekte betrachtet werden. Beispielsweise ist ein hoher Geschwindigkeitszuwachs wenig wert, wenn dadurch der Energieverbrauch über die maximal zulässige Grenze steigt. Eine erfolgreiche Schnellvalidierung erleichtert die Argumentation für die Durchführung einer detaillierten Untersuchung.

6.2 Entwicklungsmodell

Das hier vorgestellte Beispiel unterscheidet sich vom Ablauf der Datenverarbeitung her nicht wesentlich von dem in Kapitel 3.2 beschriebenen Radar-System. Die bereits vorgestellten Diagramme für den Ablauf der Signalverarbeitung (siehe Abbildung 4.8, Seite 86), die Software-Sicht der Architektur (siehe Abbildung 4.9, Seite 88) sowie die Zuordnung der Aktivitäten zu den Software-Komponenten (siehe Abbildung 4.10, Seite 88) gelten grundsätzlich auch für das erweiterte Beispiel. Diese Diagramme und deren Elemente sind Teil der modellbasierten Dokumentation der System-Architektur im Entwicklungsmodell.

Grundlage für die Validierung der System-Architektur sind die Anforderungen. Für das im erweiterten Beispiel beschriebene Radar-System sind u. a. die in Tabelle 6.1 und 6.2[78] aufgelisteten funktionalen und nichtfunktionalen Anforderungen erhoben worden.

[78] Alle genannten Daten und Preise sind fiktiv, weshalb auch eine fiktive Währung verwendet wird. Dabei handelt es sich um den Ankh-Morpork-Dollar (AM-$) nach Terry Pratchet.

Tabelle 6.1: Auszug aus der Anforderungsspezifikation (Teil 1)

ID	**Anforderung**
A1	Das System muss digitalisierte Signale in das interne Datenformat Beams konvertieren.
A2	Das System muss aus den Beams Hits ermitteln.
A3	Das System muss aus den Hits See-, Luft- oder Nahziel-Plots ermitteln.
A4	Das System muss aus den See-, Luft- und Nahziel-Plots Tracks ermitteln.
A5	Das System muss die Möglichkeit bieten Tracks, auszugeben.
A6	Das System muss für die Verarbeitung der digitalisierten Radarsignale bis zur Ausgabe der Tracks höchstens 1000ms benötigen.
A7	Das System darf nicht mehr als 440W Energie verbrauchen.
A8	Das System muss die Möglichkeiten bieten, bis zu 3500$\frac{Mibit}{sec}$ digitalisierte Radarsignale zu empfangen.
A9	Das System muss Ziele bis zu einer Entfernung von 200km mit einer Abweichung von 5m ermitteln.
A10	Das System muss Ziele bis zu einer Entfernung von 20km mit einer Abweichung von 3m ermitteln.
A11	Das System muss Seeziele bis zu einer Entfernung von 1km mit einer Abweichung von 1m ermitteln.
A12	Das System muss an der Wasseroberfläche schwimmende Objekte ermitteln, falls diese eine Fläche von mindestens $0,5m^2$ haben, falls das Objekt höchstens 1km entfernt ist und falls der Seegang höchstens Stufe 3 beträgt.
A13	Das System muss mindestens die letzten 100 Tracks verwalten.

Tabelle 6.2: Auszug aus der Anforderungsspezifikation (Teil 2)

ID	**Anforderung**
A14	Das System muss nach höchstens 30 Sekunden fähig sein, digitalisierte Signale zu verarbeiten.
A15	Das System muss fähig sein in einem luftdicht abgeschlossenen System betrieben zu werden.
A16	Das System darf im Normalbetrieb eine Temperatur von $80°C$ nicht überschreiten.
A17	Das System muss fähig sein, sich bei einer Überschreitung von $90°C$ automatisch abzuschalten.
A18	Die Hardware-Kosten müssen maximal 26000AM-$ betragen.
A19	Das System muss fähig sein, die Protokolldaten für die interne Datenverarbeitung zu speichern.
A20	Das System muss fähig sein, bis zu 6 Monate ohne Wartung in Betrieb zu sein.

Als Basis wird ein Board vom fiktiven Typ WHZ verwendet, deren Komponenten unterschiedlich konfiguriert werden können. Als Komponenten stehen FPGAs, PowerPCs, Ethernet-Controller und magnetische Festplatten (engl. *Hard Disk Drive (HDD)*) zur Verfügung. Aus diesen kann sich der Architekt die für seine Bedürfnisse passenden Boards zusammenstellen. Die Komponenten lassen sich allerdings nicht beliebig kombinieren. Im Folgenden sind einige sinnvolle Kombinationen aufgeführt:

- 2 x FPGA
- 2 x PowerPC
- 1 x FPGA, 1 x PowerPC und 1 x Ethernet-Controller

- 1 x FPGA und 1 x HDD
- 1 x PowerPC, 1 x HDD und 1 x Ethernet-Controller

Für die Modellierung mit der UML werden die verwendeten Hardware-Elemente von Komponenten repräsentiert, die als Elemente eines Werkzeugkastens für den Architekten gesehen werden können. Ein konkretes Board wird durch die Instanziierung dieser Elemente modelliert. Abbildung 6.1 zeigt die Hardware-Sicht für die Architektur des erweiterten Beispiels mit sechs Board-Instanzen und den dazugehörigen Komponenten-Instanzen. In der Abbildung sind weitere Details des Boards zu sehen. So verfügt es über vier HiSpeed-Anschlüsse und einem Ethernet-Anschluss, die durch Ports repräsentiert werden. Die Konnektoren zwischen den Ports repräsentieren physikalische Verbindungen, über die sich mögliche Kommunikationspfade ermitteln lassen.

Bei der Verwendung der Kommunikationsmedien ist zu beachten, dass die HiSpeed-Anschlüsse nur in Kombination mit FPGAs, der Ethernet-Anschluss nur in Verbindung mit dem Ethernet-Controller verwendet werden können. Ein PowerPC kann demnach nicht direkt Daten von einem HiSpeed-Anschluss empfangen. Diese müssen zunächst von einen FPGA entgegen genommen und anschließend über den Bus *PCIExpress*, der die beiden Verarbeitungseinheiten verbindet, weitergeleitet werden. Die Instanziierung von Komponenten wird auch für die Software-Komponenten verwendet. Jede Instanz besitzt dasselbe Verhalten wie der dazugehörige Typ. Auf diese Weise lässt sich die redundante Verarbeitung von Daten (horizontale Schneidung[79]) zur Steigerung der Verarbeitungsgeschwindigkeit modellieren.

[79] Mit Schneidung ist die Verfeinerung einer Komponente durch zwei oder mehr Subkomponenten gemeint.

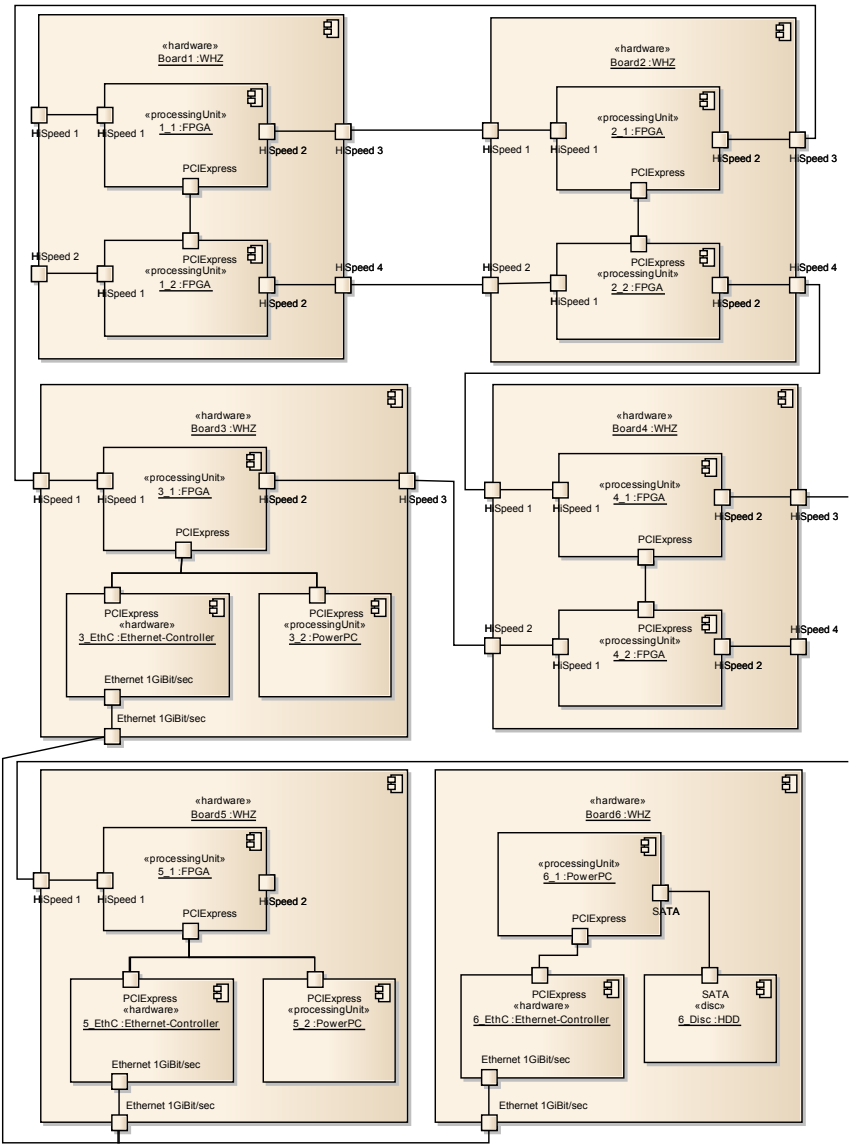


Abbildung 6.1: Hardware-Sicht im Entwicklungsmodell

Für eine einheitliche Modellierung werden deshalb nur Instanzen von Software-Komponenten einer Verarbeitungseinheit (Instanz einer Hardware-Komponente) zugeordnet. Abbildungen 6.2 und 6.3 zeigen die Deployment-Sicht der Architektur im Entwicklungsmodell.

In der Beschreibung der Diagramme und der Semantik der Elemente sind bereits die Elemente der entwicklungsspezifischen Notation enthalten. Der Stereotyp *software* charakterisiert eine UML-Komponente als Software-Komponente, analog der Stereotyp *hardware*[80]. Die Zuordnung von Software- zu Hardware-Komponenten erfolgt über eine *Dependency*-Verbindung mit dem Stereotyp *executedOn*[81].

Ports, Konnektoren, Instanzen oder die *realize*-Verbindung[82] werden nicht zur entwicklungsspezifischen Notation gezählt, da sie aus der Notationsmenge der UML stammen und somit nicht entwicklungs- bzw. projektspezifisch sind.

[80] An dieser Stelle wäre auch die Verwendung von Block-Elementen der SysML denkbar.

[81] In der UML hat der Stereotyp *deploy* eine ähnliche Semantik. Er wird allerdings im Kontext von Artefakten verwendet.

[82] Hierbei handelt es sich um die Kurzschreibweise für eine *Dependency*-Verbindung mit dem Stereotyp *realize*.

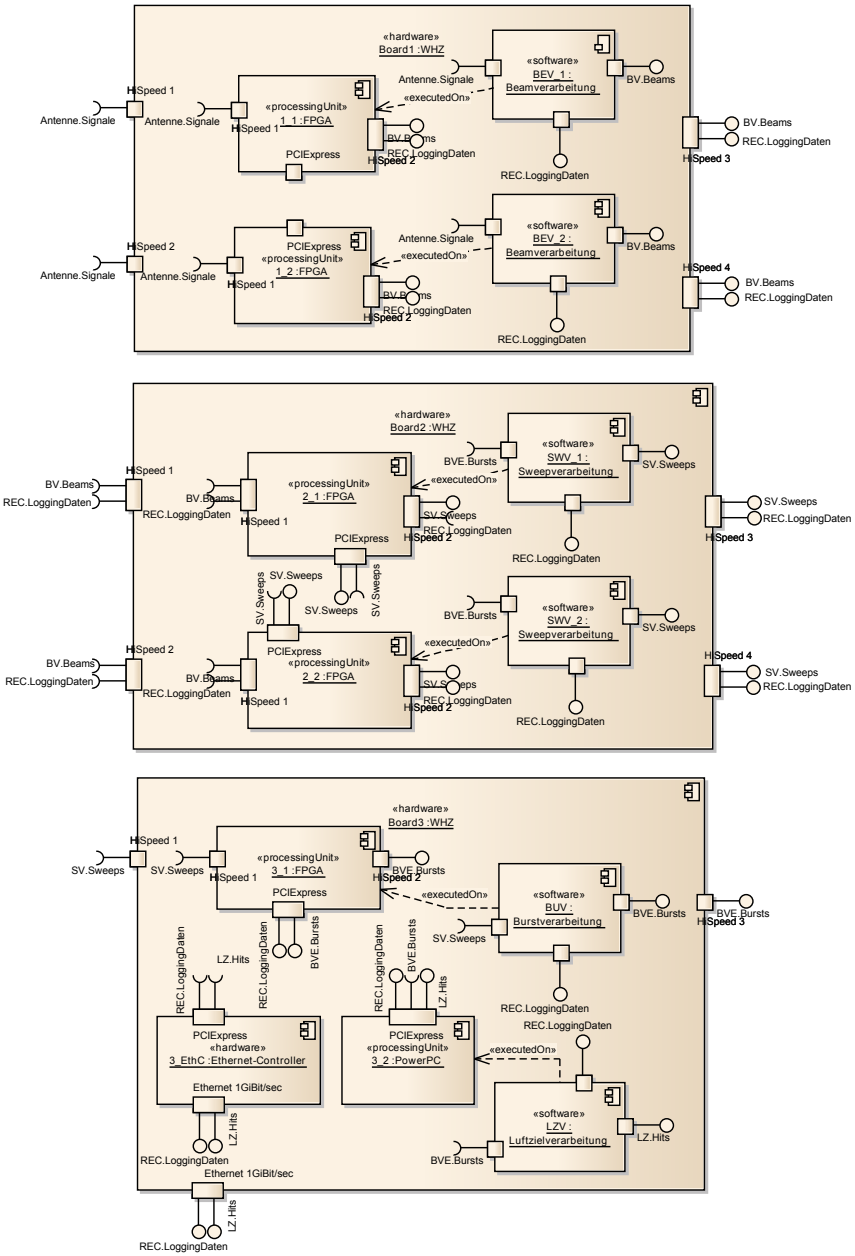


Abbildung 6.2: Deployment-Sicht im Entwicklungsmodell - Teil 1

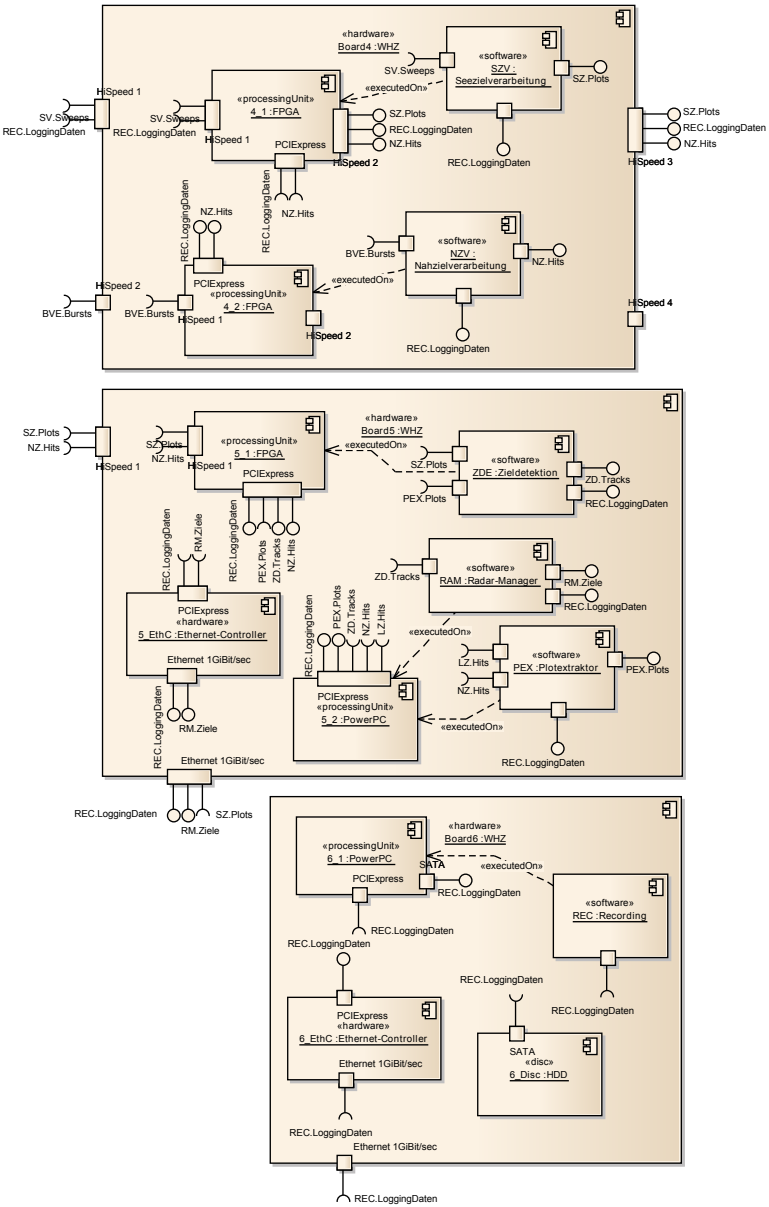


Abbildung 6.3: Deployment-Sicht im Entwicklungsmodell - Teil 2

6.3 Anwendung des Validierungsprozesses

Gemäß dem Ablauf des Validierungsprozesses in Abbildung 4.3 auf Seite 70 muss der Architekt im ersten Schritt aus den architekturrelevanten Anforderungen die Validierungsziele identifizieren.

6.3.1 Validierungsziele identifizieren

Hierfür sind vor allem die NFAs von Bedeutung, zu denen die Anforderungen A6 bis A20 zählen. Der Architekt identifiziert die folgenden Validierungsziele, wobei die dazugehörigen Anforderungen in Klammern aufgelistet sind:

- Energieverbrauch (A7),
- Gesamtverarbeitungszeit (A6),
- Datenkommunikation (A8),
- Genauigkeit[83] (A9, A10, A11, A12 und A13),
- Systeminitialisierung (A14),
- Systemeigenschaft (A15),
- Betriebstemperatur (A16),
- Systemschutz (A17),
- Kosten (A18),
- Speicherkapazität (A19) und
- Wartungsintervall (A20).

Einige der Validierungsziele beeinflussen sich, weshalb der Architekt diese miteinander verbindet. Die Abhängigkeitsbeziehungen sind in Abbildung

[83] Unter Genauigkeit wird hier ein Maß für die Übereinstimmung von berechnetem und realem Wert verstanden.

6.4 grafisch dargestellt. Die Steigerung der Leistung zur Verbesserung der Gesamtverarbeitungszeit wirkt sich auf den Energieverbrauch und die Betriebstemperatur aus. Veränderungen bei der Genauigkeit von Algorithmen wirken sich ebenso auf die Gesamtverarbeitungszeit aus wie die Datenkommunikation. Die Kosten wirken sich auf die Leistungsfähigkeit des Systems und damit auf die Gesamtverarbeitungszeit aus. Das vorgegebene Wartungsintervall bedingt die Größe des Speichers zur Persistierung der Protokolldaten und die Speichergröße wiederum hat Einfluß auf die Kosten.

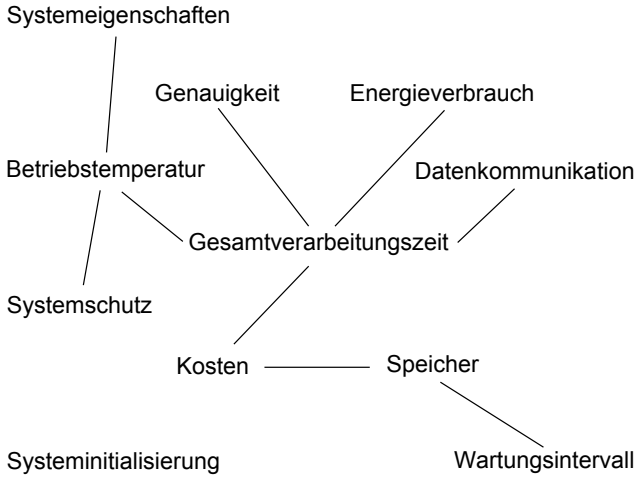


Abbildung 6.4: Abhängigkeiten zwischen den betrachteten Validierungszielen

6.3.2 Validierungszielverfahren festlegen

Als nächstes legt der Architekt die Validierungszielverfahren für die einzelnen Ziele fest. Hierfür ist Wissen über und Erfahrung mit dem System erforderlich. Zur Festlegung schaut der Architekt sich die in den Anforderungen enthaltenen Prüfkriterien, die Abhängigkeiten zwischen den Validie-

rungszielen sowie die zur Verfügung stehenden Daten (z. B. von Hardware-Komponenten, Algorithmen oder Testdaten) an. Auf Basis dieser Informationen entscheidet er, mit welchem Detaillierungsgrad die einzelnen Ziele zu validieren sind. Falls physikalisch noch kein System existiert, lassen sich einige Ziele auch noch nicht überprüfen. In solchen Fällen entscheidet der Architekt sich entweder für ein dem Entwicklungsstand angepasstes Prüfkriterien, oder er berücksichtigt diese Ziele erst zu einem späteren Zeitpunkt in der Systementwicklung.
Die im Folgenden vorgestellten Formeln der Validierungszielverfahren sind Beispiele. Der Architekt kann auch andere Berechnungsverfahren verwenden. Der Validierungsprozess macht hier keine Vorgaben.

Systeminitialisierung, Systemeigenschaft und Systemschutz Da noch kein physikalisches System existiert, entscheidet der Architekt, dass die Validierungsziele *Systeminitialisierung, Systemeigenschaften* und *Systemschutz* mit den zugeordneten Anforderungen A14, A15 und A17 erst zu einem späteren Zeiptunkt im Entwicklungsprozess berücksichtigt werden.

Betriebstemperatur Ein Beispiel für die Festlegung von dem Entwicklungsstand angepassten Prüf-/Abnahmekriterien ist die Betriebstemperatur der Systens. Aus Erfahrungen heraus ist dem Architekten bekannt, dass die Temperatur der Verarbeitungseinheiten (engl. *processing unit*, kurz PU) großen Einfluss auf die Betriebstemperatur hat. Er legt deshalb als angepasstes Abnahmekriterium für die Validierung des Ziels *Betriebstemperatur* fest, dass die Temperatur der Verarbeitungseinheiten $42°C$ nicht überschreiten dürfen. Die Temperatur wird für die Validierung in Abhängigkeit von der mittleren Auslastung der Verarbeitungseinheiten berechnet werden. Zu einem Basiswert wird ein variabler Wert hinzu addiert, der sich über das Produkt aus der mittleren Auslastung der Verarbeitungseinhei-

ten und der Differenz des maximalen Temperatur-/Energieverbrauchswerts und des entsprechenden Basiswertes ergibt (siehe Formeln 6.1 und 6.2). Die Betriebstemperatur anderer Hardware-Komponenten (z. B. Prozessorboard oder Ethernet-Controller oder Festplatten) werden nicht berücksichtigt. Fast analog wird der Energieverbrauch für die Verarbeitungseinheiten berechnet. Allerdings werden für den Energieverbrauch von Hardware-Komponenten (HW)[84] statische Werte verwendet (siehe Formeln 6.3 und 6.4).

$$Energieverbrauch = \sum_{i=1}^{n} Energie_{basis_{PU_i}} + Energie_{variabel_{PU_i}} + \sum_{j=1}^{m} Energie_{HW_i} \tag{6.1}$$

$$Energie_{variabel_{PU}} = \frac{Auslastung_{PU} * (Energie_{max_{PU}} - Energie_{basis_{PU}})}{100} \tag{6.2}$$

$$Betriebstemperatur = \sum_{i=1}^{n} Temp_{basis_{PU_i}} + Temp_{variabel_{PU_i}} \tag{6.3}$$

$$Temp_{variabel_{PU}} = \frac{Auslastung_{PU} * (Temp_{max_{PU}} - Temp_{basis_{PU}})}{100} \tag{6.4}$$

Warungsintervall Anforderung A20 (Validierungsziel *Wartungsintervall*) ist nur mit hohem Aufwand testbar. Aus diesem Grund vereinbart der Architekt mit dem Kunden folgendes Abnahmekriterium: Das System muss mindestens sechs Monate ohne den planmäßigen Wechsel von Hardware-Bauteilen betrieben werden können. Unter einem planmäßigen Wechsel wird der Austausch eines physikalischen Bauteils nach dem vom Hersteller angegebenen Ende der Betriebsdauer verstanden.

Gesamtverarbeitungszeit Im Gegensatz zum Verfahren für das Validierungsziel *Gesamtverarbeitungszeit* (siehe Formel (4.1) auf Seite 82) wählt der

[84] An dieser Stelle sind mit Hardware-Komponenten alle physikalischen Komponenten außer die Verarbeitungseinheiten gemeint.

Architekt in diesem Beispiel ein Verfahren, in dem die benötigte Zeit für die Datenkommunikation und der für die Kommunikation benötigte Verarbeitungsaufwand berücksichtigt werden. Hierfür werden zunächst empirisch ermittelte Verarbeitungszeiten für die einzelnen Software-Komponenten auf den jeweils zugeordneten Verarbeitungseinheiten verwendet. Die Zeiten geben die Verarbeitungsdauer der Software-Komponente bei alleiniger Nutzung der Verarbeitungseinheit wieder. Bei der Berechnung der Gesamtverarbeitungszeit ist deshalb das Ausführen mehrerer Software-Komponenten auf einer Verarbeitungseinheit zu berücksichtigen. In diesem Zusammenhang wird auch auf die Auswirkungen bei der Verarbeitung mehrerer Datenströme hingewiesen (siehe Anhang A.2).
Die für die Datenübertragung erforderliche Zeit ergibt sich durch die zwischen zwei Software-Komponenten auszutauschenden Daten und die durch das entsprechende Kommunikationsmedium bereitgestellte Bandbreite.[85]

Datenkommunikation Die geforderte Datenkommunikationsbandbreite wird durch einen Vergleich der benötigten mit der verfügbaren Bandbreite des Kommunikationsmediums validiert.

Speicherkapazität Die Berechnung der benötigten *Speicherkapazität* erfolgt über die Addition der protokollierten Daten der einzelnen Software-Komponenten und der Berechnung, wie viele Daten innerhalb einer Sekunde maximal geschrieben werden müssen.

Kosten Die *Kosten* werden über eine statische Aufsummierung der Preise der einzelnen Hardware-Komponenten ermittelt.

[85] Die Berechnung der Gesamtverarbeitungszeit wurde mit Hilfe einer Simulation durchgeführt. Hierbei handelt es sich um die Referenzimplementierung des in Kapitel 7.2 beschrieben Konzepts für einen generischen Simulator. Für Details zur Simulation wird auf Poßögel [Poß12, Kapitel 3] verwiesen.

Wartungsintervall Für die Validierung des *Wartungsintervalls* wird die erwartete Lebensdauer der Hardware-Komponenten betrachtet.

Genauigkeit Die Genauigkeit der Algorithmen wird mit Hilfe spezieller Testdaten überprüft und die Ergebnisse werden mit den geforderten Werten verglichen.

Nachdem der Architekt alle Validierungszielverfahren für die zu untersuchenden Ziele festgelegt hat, kann er mit dem Modellieren der von den Verfahren benötigten Daten beginnen.

6.3.3 Validierungsspezifische Notation erstellen

Um die für die Validierungszielverfahren benötigten Daten modellieren zu können, erstellt der Architekt eine VSN. Hierzu verwendet er den UML-Profilmechanismus. Zunächst untersucht er das Entwicklungsmodell, um festzustellen, welche Daten bereits modelliert sind und in das Validierungsmodell übernommen werden können. Neben der System- sowie Software- und Hardware-Komponenten inkl. der Ports, Konnektoren und *executedOn*-Verbindungen, werden auch die den Software-Komponenten zugeordneten Aktivitäten inkl. der *realize*-Verbindugen für die Validierung benötigt. Die für diese Elemente erforderlichen Notationselemente sind Teil der VSN. Die vollständige VSN ist in Abbildung 6.5 dargestellt.

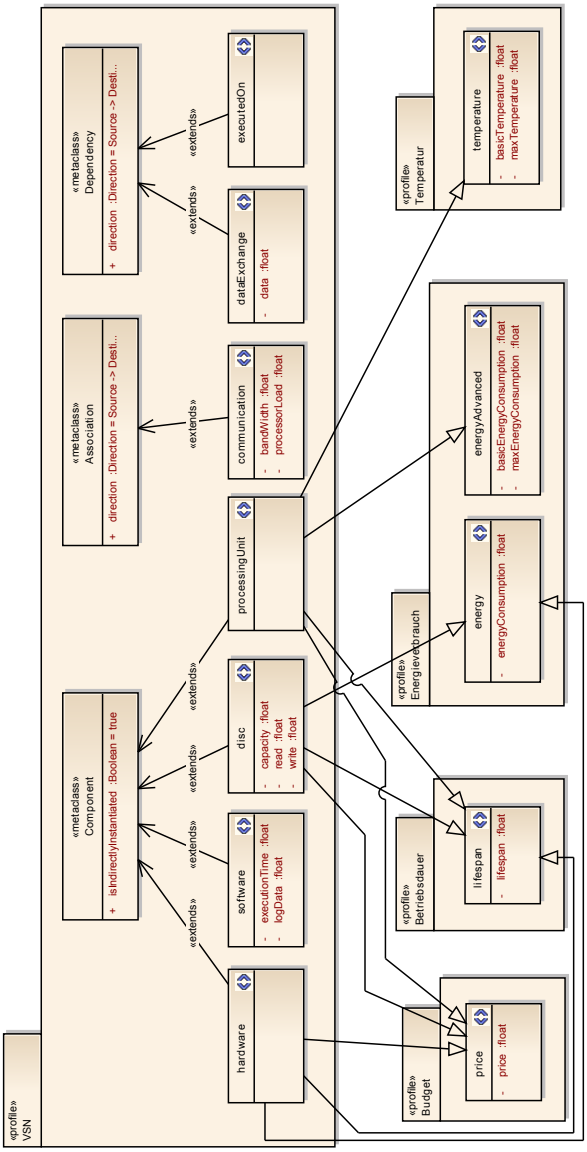


Abbildung 6.5: Validierungsspezifische Notation für das erweiterte Beispiel

Die im Profil *VSN* enthaltenen Stereotypen *hardware, software, disc, processingUnit, communication, dataExchange* und *executedOn* werden für die Modellierung der benötigten Daten im Validierungsmodell verwendet. Sie sind projektspezifisch. Es kann sich dabei

- um aus der entwicklungsspezifischen Notation unverändert übernommene Stereotypen (*extexutedOn* oder *system*),
- um aus der entwicklungsspezifischen Notation übernommene und für die Validierung erweiterte Stereotypen (*software* und *hardware*) oder
- um neue, vollständig validierungsspezifische Stereotypen (*dataExchange, disc, processingUnit* und *communication*)

handeln. Falls allein die Existenz eines Stereotyps als Information für die Verfahren nicht ausreicht, kann der Architekt weitere Daten über TaggedValues hinzufügen. Hier lässt sich zwischen projektspezifischen und projektübergreifenden Daten unterscheiden. Erstere werden direkt an den projektspezifischen Stereotypen modelliert (z. B. *data*), projektübergreifende Daten werden mit Hilfe der Vererbung[86] von anderen Stereotypen geerbt (z. B. *energyConsumption* vom Stereotyp *energy*). Zur Wiederverwendung werden die Stereotypen in separaten Profilen, den sogenannten Basisprofilen[87], abgelegt. Sie werden bei Bedarf in die VSN eingebunden (hier: *Kosten, Betriebsdauer, Energieverbrauch* und *Temperatur*). Falls der Architekt regelmäßig überprüft, ob projektspezifische Stereotypen in ein Basisprofil überführt werden können, füllt sich der Baukasten mit fortschreitender Projekterfahrung (siehe Kapitel 8.1). Wie am Beispiel des Basisprofils *Energieverbrauch* zu erkennen ist, kann ein Profil auch Stereotypen für z. B. Verfahren mit unterschiedlichem Abstraktionsniveau oder für Hardware-Komponenten mit

[86] Bei den Stereotypen ist eine Mehrfachvererbung möglich.

[87] Kapitel 8.1 geht detailliert auf die Bedeutung der Basisprofile im Validierungsprozess ein.

unterschiedlicher Charakteristika enthalten. Der Stereotyp *energy* ist für statische Verbrauchswerte geeignet, *energyAdvanced* für eine dynamische Verbrauchsberechnung.

6.3.4 Validierungsmodell erstellen

Auf Basis der VSN erstellt der Architekt das Validierungsmodell. Hierzu übernimmt[88] er einige modellierte Daten aus dem Entwicklungsmodell (Software-, Hardware-Komponenten, Aktivitäten, Ports, Konnektoren, *realize*- und *executedOn*-Verbindungen) und fügt alle für die Verfahren benötigten Informationen durch das Zuweisen von Stereotypen an die entsprechenden Modellelemente und das Setzen der Werte für die TaggedValues hinzu. Dieses Modell enthält ausschließlich validierungsrelevante Informationen, so dass der Überblick für den Architekten nicht durch rein entwicklungsspezifische Informationen erschwert wird.

Die Abbildungen 6.6 und 6.7 zeigen die Software-Sichten für die Signalverarbeitung und die Datenprotokollierung im Validierungsmodell, die Hardware-Sicht ist in Abbildung 6.8 dargestellt. Die Deployment-Sicht des Validierungsmodells ist in Abbildung 6.9 zu sehen.

[88] Die Übernahme von Modelldaten kann durch ein Werkzeug unterstützt werden. Details sind in Kapitel 7.1.4 beschrieben.

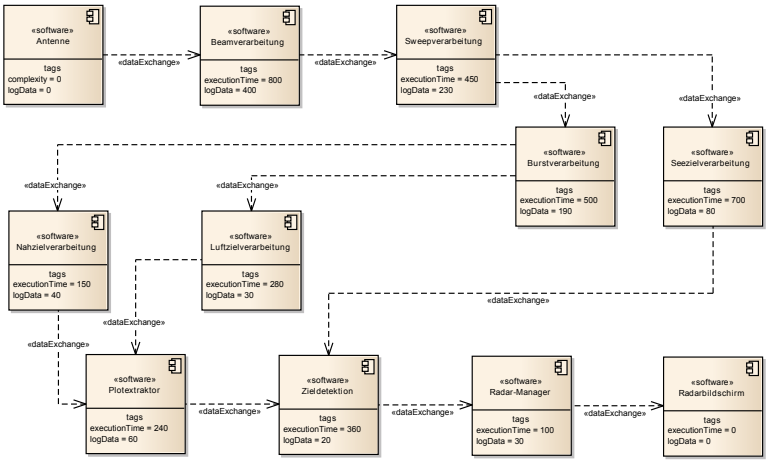


Abbildung 6.6: Software-Sicht im Validierungsmodell für die Signalverarbeitung

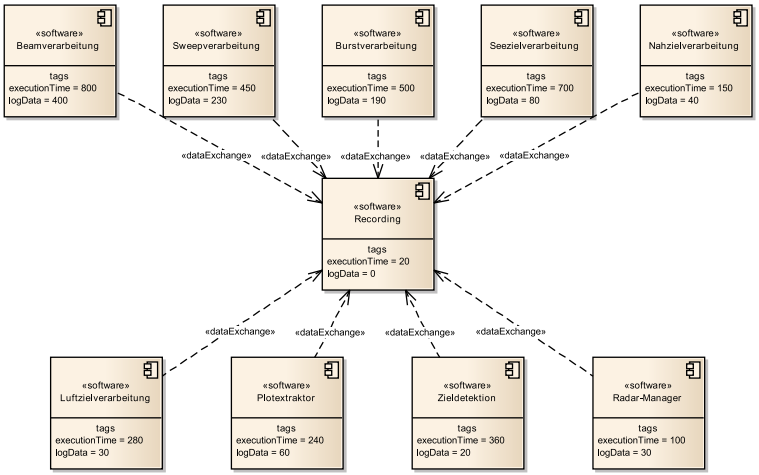


Abbildung 6.7: Software-Sicht im Validierungsmodell für die Protokollierung

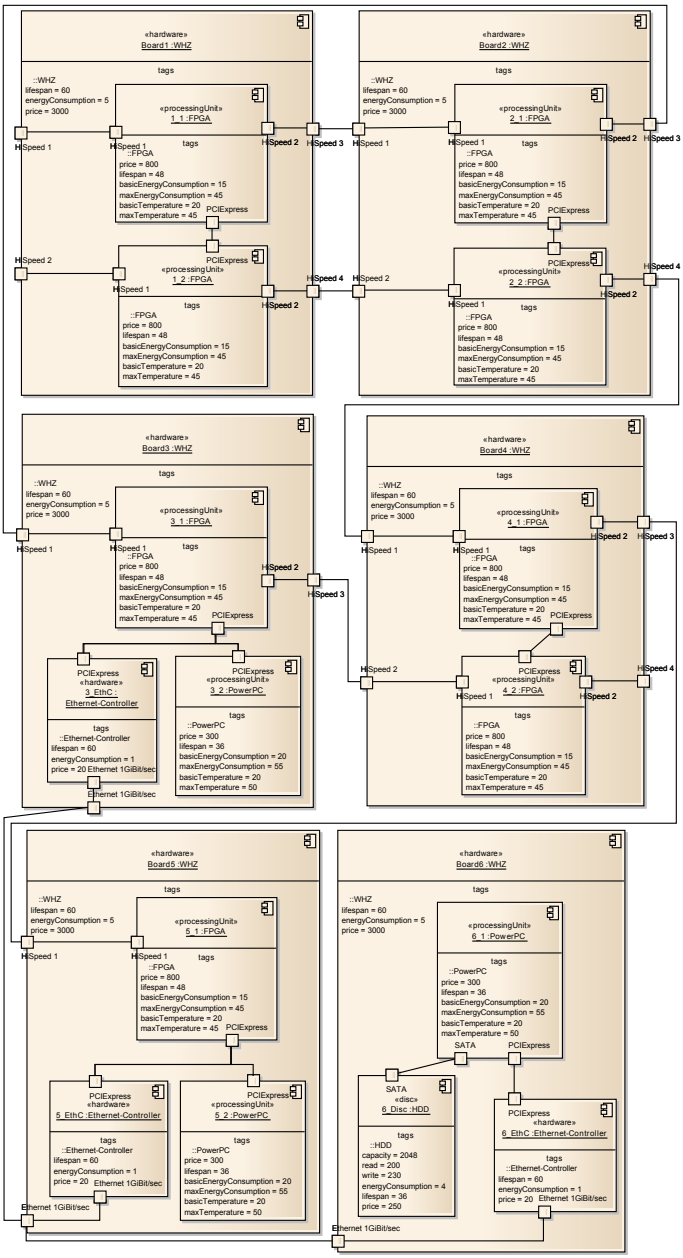


Abbildung 6.8: Hardware-Sicht im Validierungsmodell

Abbildung 6.9: Deployment-Sicht im Validierungsmodell

Das Aktivitätsdiagramm mit dem Ablauf der Signalverarbeitung (siehe Abbildung 4.8, Seite 86) sowie das Diagramm mit dem Mapping der Aktivitäten auf die Software-Komponenten (siehe Abbildung 4.10, Seite 88) sind ebenfalls im Validierungsmodell enthalten. Da an diese Elemente durch die validierungsspezifische Notation keine Zusatzinformationen für die Validierung modelliert wurden, werden sie hier nicht noch einmal dargestellt.

6.3.5 Validierung durchführen

Der Architekt führt die Validierungszielverfahren für seinen ersten Design-Entwurf durch[89]. Die Tabellen 6.3, 6.4 und 6.5 zeigen die Ergebnisse für die Validierungsziele Energieverbrauch, Betriebstemperatur und Kosten. Die grün hinterlegten Tabellenzellen bedeuten, dass die ermittelten Werte das dazugehörige Prüfkriterium erfüllen. Ein roter Hintergrund zeigt an, dass die Werte das entsprechende Prüfkriterium und damit auch das Validierungsziel nicht erfüllen. Da in Tabelle 6.3 die zwei Temperaturwerte vom FPGA *3_1* und *5_2* rot hinterlegt sind, erfüllt der Design-Entwurf die Anforderung zur Betriebstemperatur (maximal $42°C$) nicht. Die anderen drei Validierungsziele werden durch die Architektur mit den verwendeten Verfahren erfüllt. Tabelle 6.6 zeigt die Ergebnisse für die übrigen Validierungsziele (Gesamtverarbeitungszeit, Datenkommunikation, Systeminitialisierung, Speicherkapazität und Genauigkeit). Die Gesamtverarbeitungszeit des Systems ist höher als der in Anforderung A6 geforderte Wert.

[89] Die Ergebnisse sind einer Simulation entnommen. Auf eine genaue Beschreibung wie sich diese ergeben, wird an dieser Stelle verzichtet, da es für die Anwendung des Validierungsprozesses nicht von Bedeutung ist.

Tabelle 6.3: Validierungsergebnisse für Temperatur und Energieverbrauch (1. Design-Entwurf) - Teil 1

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board1	WHZ	-	-	5
1_1	FPGA	83	40,75	39,9
1_2	FPGA	82	40,5	39,6
Board2	WHZ	-	-	5
2_1	FPGA	72	38	36,6
2_2	FPGA	71	37,75	36,3
Board3	WHZ	-	-	5
3_1	FPGA	93	43,25	42,9
3_2	PowerPC	30	29	30,5
3_EthC	Ethernet-Controller	-	-	1
Board4	WHZ	-	-	5
4_1	FPGA	68	37	35,4
4_2	FPGA	23	25,75	21,9
Board5	WHZ	-	-	5
5_1	PowerPC	52	35,6	38,2
5_2	PowerPC	76	42,8	46,6
5_EthC	Ethernet-Controller	-	-	1

Tabelle 6.4: Validierungsergebnisse für Temperatur und Energieverbrauch (1. Design-Entwurf) - Teil 2

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board6	Intel	-	-	5
6_1	PowerPC	19	25,7	26,65
6_Disc	HDD	-	-	4
6_EthC	Ethernet-Controller	-	-	1
Summe				**431,55**

Tabelle 6.5: Validierungsergebnis für das Validierungsziel Kosten (1. Design-Entwurf)

Hardware	**Einzelpreis in [AM-$]**	**Lebens-dauer in [Monat]**	**Anzahl**	**Gesamt-kosten in [AM-$]**
Board WHZ	3000	60	6	18000
FPGA	850	48	7	5950
PowerPC	300	36	4	1200
EC	20	60	3	60
HDD	250	36	1	250
Summe				**25460**

Neben den Validierungsergebnissen stehen dem Architekten durch die Verwendung von Simulationen als Validierungsverfahren nicht nur konkrete Ergebnisse für die Problemanalyse zur Verfügung, er kann auch auf Zwi-

schenwerte aus den Simulationen zugreifen. Anhand der konkreten Ergebnisse erkennt der Architekt, dass sowohl FPGA 3_1 als auch 5_2 die maximal zulässige Temperatur überschreiten. Die sehr hohe Auslastung beim FPGA 3_1 weist auf einen Flaschenhals in der Verarbeitungskette hin, welcher die Ursache für die zu hohe Gesamtverarbeitungszeit (1231ms) sein kann. Zur Überprüfung dieser Vermutung analysiert der Architekt die Simulationsdaten im Detail. Tabelle 6.7 zeigt die durchschnittliche und maximale Anzahl an Datenpaketen in den Eingangspuffern ausgesuchter Software-Komponenten während der Simulation.

Tabelle 6.6: Validierungsergebnisse der restlichen Validierungsziele (1. Design-Entwurf)

Validierungsziel	**Prüfkriterium**	**Validierungs-ergebnis**
Gesamtverarbeitungszeit	$x \leq 1000ms$	$1231ms$
Datenkommunikation	$x \geq 3500MiBit/sec$	$5200MiBit/sec$
Systeminitialisierung	$x \leq 30sec$	$23sec$
Speicherkapazität	$x \leq 2048GiB$	$310GiB$
Schreibrate Daten	$x \geq 120MiB/sec$	$200MiB/sec$
Genauigkeit	$Seeziel \leq 0,5m^2$	$0,37m^2$
Genauigkeit	$Abweichung_{1km} \leq 1m$	$0,7m$
Genauigkeit	$Abweichung_{20km} \leq 3m$	$2,1m$
Genauigkeit	$Abweichung_{200km} \leq 5m$	$4,7m$

Tabelle 6.7: Durchschnittliche und maximale Anzahl an Datenpaketen in den Eingangspuffern ausgesuchter Software-Komponenten

Software-Komponente	**verarbeitete Datenpakete**	**Pakete (Maximum)**	**Pakete (Durchschnitt)**
Sweepverarbeitung 1	2550	31	3,2
Sweepverarbeitung 2	2550	30	3,2
Burstverarbeitung	3400	800	324,6
Seezielverarbeitung	1700	80	21,3
Nahzielverarbeitung	1700	8	0.8
Luftzielverarbeitung	1700	15	1,8

Auch wenn diese Simulationsdaten nicht besonders detailliert sind[90], so wird trotzdem deutlich, dass der Eingangspuffer der Burstverarbeitung im Durchschnitt und im Maximum deutlich mehr Datenpakete enthält als die der anderen Software-Komponenten[91]. Diese Daten bekräftigen die Vermutung, dass die Burstverarbeitung ein Flaschenhals in der Signalverarbeitungskette ist. Während auf 5_2 zwei Software-Komponenten ausgeführt werden, von denen eine zur Reduzierung der Auslastung auf eine andere Verarbeitungseinheit verschoben werden kann, wird auf 3_1 nur eine Software-Komponente ausgeführt. Eine horizontale Teilung der Burstverarbeitung und die Ausführung auf FPGA 3_1 und 4_2 könnte den Flaschenhals beseitigen. Der PowerPC 5_2 wird entlastet, indem die Software-Komponente Radar-Manager auf den PowerPC 6_1 verschoben wird, da die-

[90] Es werden beispielsweise keine Pufferobergrenzen oder Paketgrößen berücksichtigt.

[91] Eine hohe Anzahl an Datenpaketen im Ausgangspuffer würde auf das Kommunikationsmedium als Flaschenhals hinweisen, wobei hier auch die Eingangspuffer der im Verarbeitungsprozess nachfolgenden Software-Komponente und die verwendeten Simulationsalgorithmen betrachtet werden müssten.

ser nur eine geringe Auslastung hat. Nachdem der Architekt diese Änderungen im Validierungsmodell modelliert hat, führt er die Architekturvalidierung erneut durch. Die Ergebnisse sind in Tabelle 6.8, 6.9 und 6.10 zu sehen. Alle Ziele sind valide und somit auch das modellierte Design. Die vorgenommenen Modifikationen der Architektur im Validierungsmodell überträgt der Architekt auf das Entwicklungsmodell, dessen Deployment-Sicht in Abbildung 6.10 zu sehen ist. Die Veränderungen gegenüber der Deployment-Sicht in Abbildung 6.9 sind hervorgehoben.

Tabelle 6.8: Validierungsergebnisse der restlichen Validierungsziele nach der Anpassung

Validierungsziel	**Prüfkriterium**	**Validierungs-ergebnis**
Gesamtverarbeitungszeit	$x \leq 1000ms$	$917ms$
Datenkommunikation	$x \geq 3500MiBit/sec$	$5200MiBit/sec$
Systeminitialisierung	$x \leq 30sec$	$23sec$
Speicherkapazität	$x \leq 2048GiB$	$310GiB$
Schreibrate Daten	$x \geq 120MiB/sec$	$200MiB/sec$
Genauigkeit	$Seeziel \leq 0,5m^2$	$0,37m^2$
Genauigkeit	$Abweichung_{1km} \leq 1m$	$0,7m$
Genauigkeit	$Abweichung_{20km} \leq 3m$	$2,1m$
Genauigkeit	$Abweichung_{200km} \leq 5m$	$4,7m$

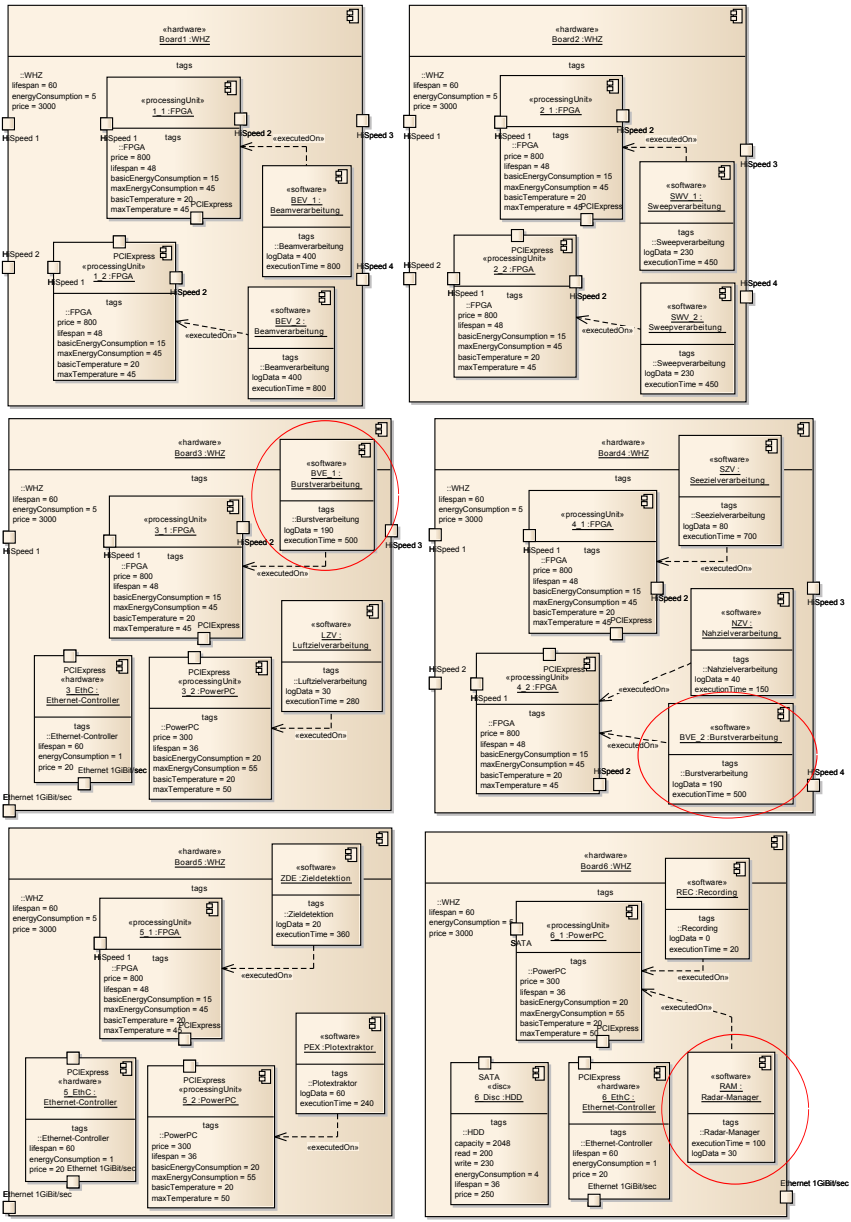


Abbildung 6.10: Deployment-Sicht im Validierungsmodell nach der Anpassung

Tabelle 6.9: Validierungsergebnisse für Temperatur und Energieverbrauch nach der ersten Anpassung - Teil 1

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board1	WHZ	-	-	5
1_1	FPGA	83	40,75	39,9
1_2	FPGA	82	40,5	39,6
Board2	WHZ	-	-	5
2_1	FPGA	72	38	36,6
2_2	FPGA	71	37,75	36,3
Board3	WHZ	-	-	5
3_1	FPGA	59	34,75	32,7
3_2	PowerPC	30	29	30,5
3_EthC	Ethernet-Controller	-	-	1
Board4	WHZ	-	-	5
4_1	FPGA	68	37	35,4
4_2	FPGA	62	35,5	33,6
Board5	WHZ	-	-	5
5_1	PowerPC	52	35,6	38,2
5_2	PowerPC	47	34,1	36,45
5_EthC	Ethernet-Controller	-	-	1

Tabelle 6.10: Validierungsergebnisse für Temperatur und Energieverbrauch nach der ersten Anpassung - Teil 2

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energieverbrauch in [W]**
Board6	Intel	-	-	5
6_1	PowerPC	42	32,6	34,7
6_Disc	HDD	-	-	4
6_EthC	Ethernet-Controller	-	-	1
Summe				**430,95**

6.3.6 Impact-Analyse nach Architekturänderungen

Mit fortschreitender Entwicklung des Systems verbessert sich bei den Entwicklern das Verständnis vom zu entwickelnden System. Dadurch ergeben sich neue Details, die auch Auswirkungen auf das Design haben. Im beschriebenen Beispiel ergaben detaillierte Analysen der verwendeten Signalverarbeitungsalgorithmen, dass mit bestimmten Testdaten für eine kurze Zeitspanne eine deutlich höhere Menge an Protokolldaten entsteht. Dies hat einerseits Einfluss auf die Gesamtmenge der Daten und andererseits auf die erforderliche Schreibrate der Festplatte, um die Daten zu persistieren. Gemäß dem Validierungsprozess in Abbildung 4.3 auf Seite 70 folgt nach einer Architekturänderung die Anpassung des Validierungsmodells. Der Architekt tauscht Board6 vom Typ WHZ gegen ein Prozessorboard vom Typ Intel aus. Dieses enthält eine Solid State Disk (SSD) mit einer höheren Lese- und Schreibrate für Daten und einer geringeren Speicherkapazi-

tät. Zusätzlich ändert er die entsprechenden validierungsspezifischen Werte (TaggedValue *logData* des Stereotyps *software*) für die zu protokollierende Datenmenge. Anschließend führt der Architekt die Validierung durch. Die Ergebnisse sind in Tabelle 6.11, 6.12, 6.13 und 6.14 dargestellt. Die Validierung ist erfolgreich, so dass das vom Architekten ins Validierungsmodell eingefügte Prozessorboard in das Entwicklungsmodell übernommen wird. Die Änderung der validierungsspezifischen Daten (höhere Protokolldatenmengen) wirkt sich nicht auf die Daten im Entwicklungsmodell aus, da es keine validierungsspezifischen Daten enthält.

Tabelle 6.11: Validierungsergebnisse der restlichen Validierungsziele nach der Architekturänderung)

Validierungsziel	**Prüfkriterium**	**Validierungs-ergebnis**
Gesamtverarbeitungszeit	$x \leq 1000ms$	$883ms$
Datenkommunikation	$x \geq 3500MiBit/sec$	$5200MiBit/sec$
Systeminitialisierung	$x \leq 30sec$	$23sec$
Speicherkapazität	$x \leq 512GiB$	$450GiB$
Schreibrate Daten	$x \geq 260MiB/sec$	$500MiB/sec$
Genauigkeit	$Seeziel \leq 0,5m^2$	$0,37m^2$
Genauigkeit	$Abweichung_{1km} \leq 1m$	$0,7m$
Genauigkeit	$Abweichung_{20km} \leq 3m$	$2,1m$
Genauigkeit	$Abweichung_{200km} \leq 5m$	$4,7m$

Tabelle 6.12: Validierungsergebnisse für Temperatur und Energieverbrauch nach der Architekturänderung - Teil 1

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board1	WHZ	-	-	5
1_1	FPGA	83	40,75	39,9
1_2	FPGA	82	40,5	39,6
Board2	WHZ	-	-	5
2_1	FPGA	72	38	36,6
2_2	FPGA	71	37,75	36,3
Board3	WHZ	-	-	5
3_1	FPGA	59	34,75	32,7
3_2	PowerPC	30	29	30,5
3_EthC	Ethernet-Controller	-	-	1
Board4	WHZ	-	-	5
4_1	FPGA	68	37	35,4
4_2	FPGA	62	35,5	33,6
Board5	WHZ	-	-	5
5_1	PowerPC	52	35,6	38,2
5_2	PowerPC	47	34,1	36,45
5_EthC	Ethernet-Controller	-	-	1

Tabelle 6.13: Validierungsergebnisse für Temperatur und Energieverbrauch nach der Architekturänderung - Teil 2

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energieverbrauch in [W]**
Board6	Intel	-	-	5
6_1	iX	33	38,2	36,5
6_Disc	HDD	-	-	4
6_EthC	Ethernet-Controller	-	-	1
Summe				**432,75**

Tabelle 6.14: Validierungsergebnis für das Validierungsziel Kosten nach der Architekturänderung

Hardware	**Einzelpreis in [AM-$]**	**Lebensdauer in [Monat]**	**Anzahl**	**Gesamtkosten in [AM-$]**
Board WHZ	3000	60	5	15000
Board Intel	400	36	1	400
FPGA	850	48	7	5950
PowerPC	300	36	3	900
iX	200	18	1	200
EC	20	60	3	60
SSD	500	12	1	500
Summe				**23010**

Die Architekturänderung bringt den gewünschten Effekt mit sich: die Hardware-Kosten sinken. Dies liegt an den günstigeren Preisen für das Prozessorboard vom Typ *Intel* und der Verarbeitungseinheit *iX*. Daran ändert auch der höhere Preis für die Festplatte vom Typ *SSD* nichts. Die Verarbeitungseinheit iX ist zwar schneller, was sich in einer besseren Gesamtverarbeitungszeit widerspiegelt, allerdings verbraucht diese mehr Energie, so dass auch hier ein Anstieg zu erkennen ist. Die Veränderungen sind allerdings alle im Rahmen der von den Anforderungen vorgegebenen Grenzen.

6.3.7 Impact-Analyse nach Änderungen an den Anforderungen

Gerade bei Projekten mit einer Laufzeit über mehrere Jahre ist es keine Seltenheit, dass der Kunde bis in die Design-Phase hinein Veränderungen an der Anforderungsspezifikation[92] vornimmt. Gründe hierfür sind beispielsweise rechtliche, technische, technologische oder funktionale Anpassungen, so dass das zu entwickelnde System zum Zeitpunkt der Fertigstellung (in einigen Jahren) auch den gesetzlichen Bestimmungen, dem aktuellen Stand der Technik oder den Anforderungen der potenziellen Kunden an das System entspricht. Für das erweiterte Beispiel kommt die folgende Anforderung hinzu:

- Die Kosten für die Wartung des Systems müssen bei einer Laufzeit von 15 Jahren maximal 5000AM-$ pro Jahr betragen. (A21)

Gemäß dem Validierungsprozess in Abbildung 4.3 auf Seite 70 folgt nach dem Hinzufügen oder Löschen von Anforderungen die Identifizierung der

[92] Die Anforderungsspezifikation ist die Menge aller das zu entwickelnde System beschreibenden Anforderungen.

Validierungsziele. Der Architekt ordnet die Anforderung A21 dem Validierungsziel *Wartungskosten* zu und verbindet diese mit dem Validierungsziel Kosten. Dies lässt sich damit begründen, dass sich Preisänderungen der einzelnen Hardware-Bauteile auch auf die Wartungskosten auswirken. Im nächsten Schritt legt der Architekt die Berechnungsformeln 6.5, 6.6 und 6.7 als Validierungszielverfahren fest[93]. Das Prüfkriterium ergibt sich aus der Anforderung A21 und lautet: $Wartungskosten \leq 5000AM - \$$.

$$Wartungskosten = Wartungskosten_{HW} + Wartungskosten_{PU} \quad (6.5)$$

$$Wartungskosten_{HW} = \sum_{i=1}^{n} \left\lceil \frac{Laufzeit_{System}}{Betriebsdauer_{HW_i}} \right\rceil * Kosten_{HW_i} * \frac{1}{Laufzeit_{System}} \quad (6.6)$$

$$Wartungskosten_{PU} = \sum_{j=1}^{m} \left\lceil \frac{Laufzeit_{System}}{Betriebsdauer_{PU_j}} \right\rceil * Kosten_{PU_j} * \frac{1}{Laufzeit_{System}} \quad (6.7)$$

Die für die Berechnung benötigten Werte sind bereits teilweise vorhanden (Betriebsdauer der Bauteile). Die für die Berechnung zu verwendende Laufzeit des Systems wird mit Hilfe eines neuen TaggedValues *lifetime* für den Stereotyp *system* in der validierungsspezifischen Notation ergänzt. Auf Basis dieser veränderten Notation passt der Architekt das Validierungsmodell an. Die veränderte Deployment-Sicht ist in Abbildung 6.11 dargestellt. Die im Validierungsmodell geänderten Daten dienen der Validierung als Datenbasis für die Verfahren. Tabelle 6.15 zeigt die Ergebnisse für das Validierungsziel Wartungskosten.

[93] Da der Stereotyp *disc* vom Stereotyp *hardware* erbt, zählen die verwendeten Festplatten zu den Hardware-Bauteilen. Eine Vererbungsbeziehung zwischen dem Stereotyp *processingUnit* und *hardware* wurde nicht verwendet, da beide eine unterschiedliche Berechnung für den Energieverbrauch verwenden.

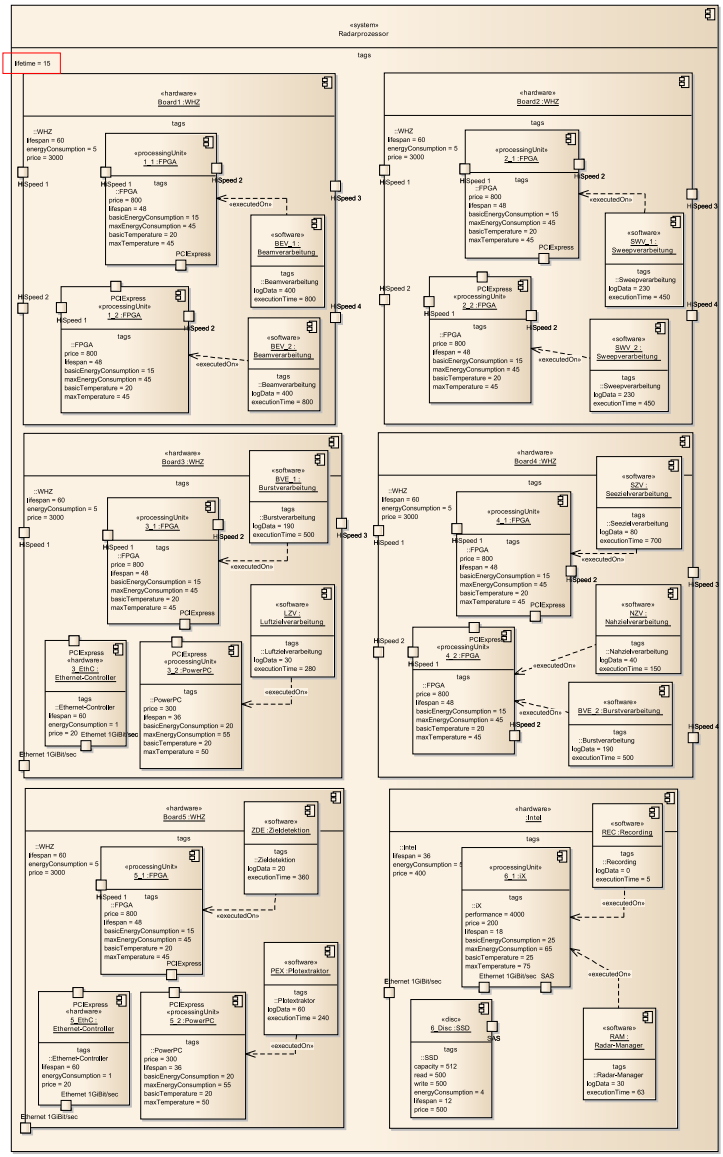


Abbildung 6.11: Deployment-Sicht im Validierungsmodell nach dem Hinzufügen der validierungsspezifischen Daten für die Wartungskosten

Die Ergebnisse der übrigen Ziele haben sich nicht verändert und werden deswegen nicht noch einmal aufgeführt. Die Validierung war erfolgreich, weshalb im nächsten Schritt die Sychronisation von Validierungsmodell und Entwicklungsmodell erfolgt. Da aber keine für das Entwicklungsmodell relevanten Änderungen vorgenommen wurden, werden keine Daten transformiert.

Tabelle 6.15: Validierungsergebnis für die Wartungskosten nach der Anforderungsänderung

Hardware	**Einzelpreis in [AM-$]**	**Lebensdauer in [Monat]**	**Anzahl**	**Wartungskosten in [AM-$]**
Board WHZ	3000	60	5	2000,00
Board Intel	400	36	1	106,67
FPGA	850	48	7	1190,00
PowerPC	300	36	3	240
iX	200	18	1	120
EC	20	60	3	8
SSD	500	12	1	466,67
Summe				**4131,34**

6.3.8 Impact-Analyse nach Änderungen an den Prüfkriterien

Nachdem im letzten Abschnitt eine neue Anforderung hinzugekommen ist, werden in diesem Abschnitt die Prüfkriterien zweier Anforderungen geändert. Dabei handelt es sich um Änderungen, die nicht den (semantischen) Inhalt der Anforderung betreffen. Eine solche semantische Veränderung lässt sich auf das Löschen einer alten und Hinzufügen einer neuen An-

forderung abbilden, was durch den vorhergehenden Abschnitt behandelt wird. Die Änderung der Prüfkriterien erfolgt im Bereich der Kosten und betrifft die Anforderungen A18 und A21. Es müssen Senkungen um jeweils 20% ermöglicht werden. Die Hardware-Kosten dürfen demnach maximal 20800AM-$ betragen, die Wartungskosten nur noch 4000AM-$. Bei Veränderung an den Prüfkriterien ist lediglich eine Revalidierung der Architektur erforderlich. Anpassungen an den Validierungszielen, deren Verfahren, dem Validierungsmodell oder an der validierungsspezifischen Notation sind nicht erforderlich. Die Validierungsergebnisse ergeben, dass das aktuelle Design die Validierungsziele Kosten und Wartungskosten nicht erfüllt (23010AM-$ ist größer als 20800AM-$ und 4131,34AM-$ ist größer als 4000AM-$)[94]. Wie beim WHZ-Prozessorboard, so ist es auch beim Intel-Proztessorboard möglich, anstatt der Festplatte eine zweite Verarbeitungseinheit zu verwenden. Da es preislich günstiger ist und vor allem die SSD-Festplatte hohe Wartungskosten aufweist (Lebensdauer 12 Monate, Kosten 500AM-$), entscheidet der Architekt nach der Analyse der aktuellen und vergangenen Validierungsergebnisse, ein weiteres WHZ-Board gegen eines vom Typ Intel zu tauschen. Er modelliert diese Änderungen in das Validierungsmodell. Dazu zählen der Tausch der Boards und die Veränderung der Ausführungszeit der Software-Komponenten, die nun auf den Verarbeitungseinheiten vom Typ iX ausgeführt werden. Die veränderte Deployment-Sicht des Validierungsmodells ist in Abbildung 6.12 zu sehen. Nach der Validierung zeigt sich, dass der Board-Tausch zur gewünschten Kostensenkung geführt hat. Sowohl das Validierungsziel Kosten als auch Wartungskosten liegen unterhalb der geforderten Werte (siehe Tabelle 6.16). Die höhere Leistung der Verarbeitungseinheit iX, die sich in den geringeren Ausführungs-

[94] Auf eine Darstellung der Ergebnisse in Tabellen wird verzichtet, da die Werte denen in den Tabellen 6.11, 6.12, 6.13 und 6.14 entsprechen. Der Unterschied liegt lediglich in der fehlgeschlagenen Validierung für die Validierungsziele Kosten und Wartungskosten, so dass die entsprechenden Zellen nun farblich rot hinterlegt wären.

zeiten für die darauf ausgeführten Software-Komponenten ausdrückt (siehe Abbildung 6.12), senken die Gesamtverarbeitungszeit des Systems (siehe Tabelle 6.19). Die Validierungsergebnisse in den Tabellen 6.17 und 6.18 zeigen aber auch, dass sich die iX im oberen des erlaubten Temperaturbereichs befinden und einen höheren Stromverbrauch als die PowerPCs haben. Nach der erfolgreichen Validierung übernimmt der Architekt noch die veränderte Architektur (Boardtausch) in das Entwicklungsmodell.

Tabelle 6.16: Validierungsergebnisse für Kosten und Wartungskosten nach der Änderung der Prüfkriterien

Hardware	**Einzel-preis in [AM-$]**	**Lebens-dauer in [Monat]**	**Anzahl**	**Gesamt-kosten in [AM-$]**	**Wartungs-kosten in [AM-$]**
Board WHZ	3000	60	4	12000	1600,00
Board Intel	400	36	2	800	213,33
FPGA	850	48	7	5950	1190,00
PowerPC	300	36	1	300	80
iX	200	18	3	600	360
EC	20	60	3	60	8
SSD	500	12	1	500	466,67
Summe				**20210**	**3918,00**

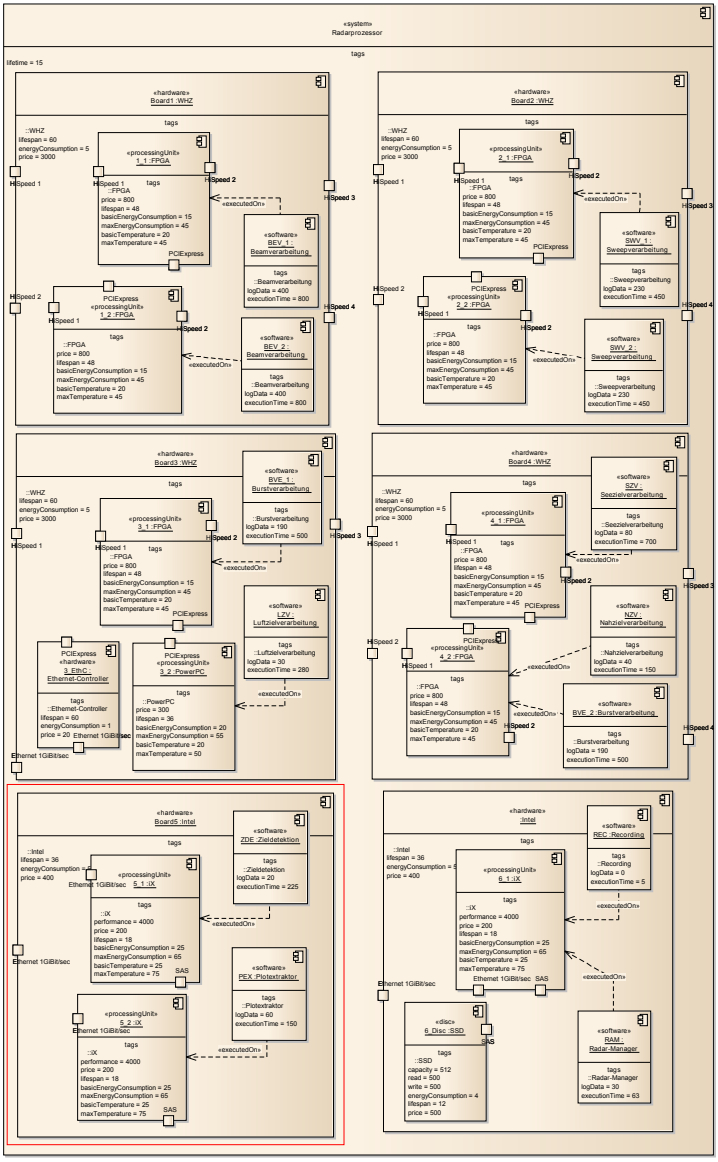


Abbildung 6.12: Deployment-Sicht im Validierungsmodell nach der Änderung der Prüfkriterien

Tabelle 6.17: Validierungsergebnisse für Temperatur und Energieverbrauch nach der Änderung der Prüfkriterien - Teil 1

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board1	WHZ	-	-	5
1_1	FPGA	83	40,75	39,9
1_2	FPGA	82	40,5	39,6
Board2	WHZ	-	-	5
2_1	FPGA	72	38	36,6
2_2	FPGA	71	37,75	36,3
Board3	WHZ	-	-	5
3_1	FPGA	59	34,75	32,7
3_2	PowerPC	30	29	30,5
3_EthC	Ethernet-Controller	-	-	1
Board4	WHZ	-	-	5
4_1	FPGA	68	37	35,4
4_2	FPGA	62	35,5	33,6
Board5	Intel	-	-	5
5_1	iX	41	41,4	40,5
5_2	iX	39	40,6	39,5
5_EthC	Ethernet-Controller	-	-	1

Tabelle 6.18: Validierungsergebnisse für Temperatur und Energieverbrauch nach der Änderung der Prüfkriterien - Teil 2

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energieverbrauch in [W]**
Board6	Intel	-	-	5
6_1	iX	33	38,2	36,5
6_Disc	SSD	-	-	4
6_EthC	Ethernet-Controller	-	-	1
Summe				**438,1**

Tabelle 6.19: Validierungsergebnisse der restlichen Validierungsziele nach der Änderung der Prüfkriterien)

Validierungsziel	**Prüfkriterium**	**Validierungsergebnis**
Gesamtverarbeitungszeit	$x \leq 1000ms$	$865ms$
Datenkommunikation	$x \geq 3500MiBit/sec$	$5200MiBit/sec$
Systeminitialisierung	$x \leq 30sec$	$23sec$
Speicherkapazität	$x \leq 512GiB$	$450GiB$
Schreibrate Daten	$x \geq 260MiB/sec$	$500MiB/sec$
Genauigkeit	$Seeziel \leq 0,5m^2$	$0,37m^2$
Genauigkeit	$Abweichung_{1km} \leq 1m$	$0,7m$
Genauigkeit	$Abweichung_{20km} \leq 3m$	$2,1m$
Genauigkeit	$Abweichung_{200km} \leq 5m$	$4,7m$

6.3.9 Impact-Analyse nach Wechsel eines Validierungszielverfahrens

Mit dem Fortschreiten der Systementwicklung steigt auch das Detailwissen über die verschiedenen Algorithmen zur Signalverarbeitung. Den Entwicklern ist es deshalb möglich, eine Abschätzung für die benötigten Flops abzugeben. Durch die Veränderung des Validierungszielverfahrens zur Ermittlung der Gesamtverarbeitungszeit fließt dieses Detailwissen in die Architekturvalidierung ein. Da das Validierungsziel Gesamtverarbeitungszeit mit den Zielen Betriebstemperatur und Energieverbrauch verbunden ist (siehe Abbildung 6.4 auf Seite 168), lässt sich bereits vor der Revalidierung sagen, dass sich die Verfahrensänderung auch auf deren Validierungsergebnisse auswirken wird.

Gemäß dem Validierungsprozess in Abbildung 4.3 auf Seite 70 folgt nach Änderungen an Validierungszielverfahren, dass die neuen Verfahren festgelegt und ggf. erforderliche Änderungen an anderen Verfahren vorgenommen werden. Anschließend passt der Architekt die validierungsspezifische Notation an. Zum einen muss die Angabe der benötigten Flops bei Software-Komponenten und zum anderen die Angabe der Leistungsfähigkeit bei Hardware-Komponenten ermöglicht werden. Dies wird über je einen neuen TaggedValue am Stereotyp *software* (complexity) und *hardware* (performance) realisiert. Der TaggedValue *executionTime* des Stereotyps *software* fällt weg. Das angepasste UML-Profil der validierungsspezifischen Notation ist in Abbildung 6.13 dargestellt.

Im nächsten Schritt passt der Architekt das Validierungsmodell den Veränderungen an. Dazu wechselt er das verwendete UML-Profil und aktua-

lisiert die validierungsrelevanten Daten. Die aktualisierte Deployment-Sicht des Validierungsmodells ist in Abbildung 6.14 zu sehen. Im nächsten Schritt startet er die Architekturvalidierung, die als Datenbasis das veränderte Validierungsmodell verwendet. Da sich die verwendete Hardware nicht geändert hat, entsprechen die Resultate für die Validierungsziele Wartungskosten und Kosten denen in Tabelle 6.16.

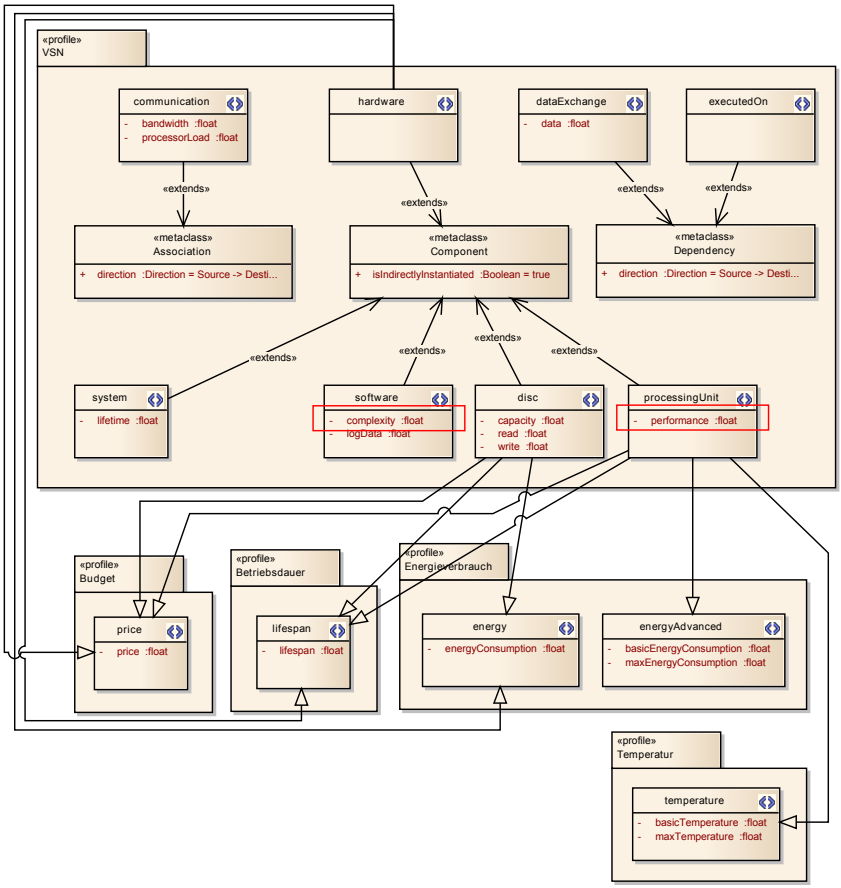


Abbildung 6.13: Validierungsspezifische Notation des erweiterten Beispiels nach dem Wechsel des Validierungszielverfahrens

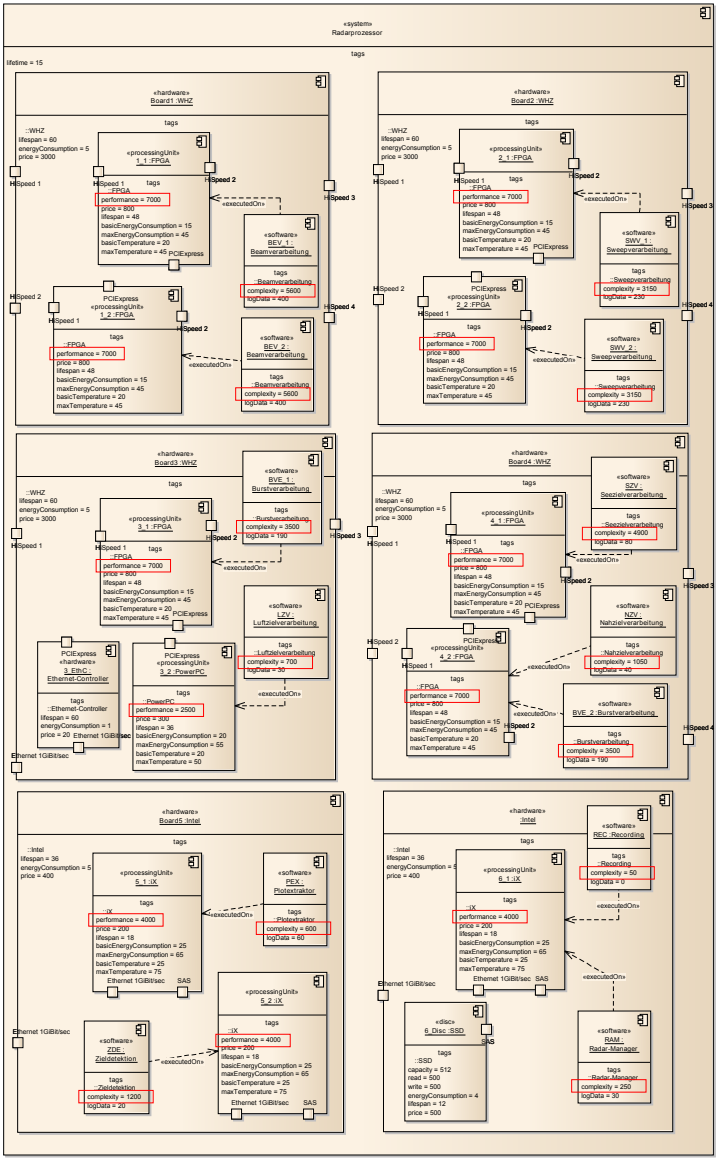


Abbildung 6.14: Deployment-Sicht im Validierungsmodell nach dem Wechsel des Validierungszielverfahrens

Tabelle 6.20: Validierungsergebnisse für Temperatur & Energieverbrauch nach dem Wechsel des Validierungszielverfahrens - Teil 1

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board1	WHZ	-	-	5
1_1	FPGA	90	42,5	42
1_2	FPGA	89	42,25	41,7
Board2	WHZ	-	-	5
2_1	FPGA	78	39,5	38,4
2_2	FPGA	77	39,25	38,1
Board3	WHZ	-	-	5
3_1	FPGA	64	36	34,2
3_2	PowerPC	33	29,9	31,55
3_EthC	Ethernet-Controller	-	-	1
Board4	WHZ	-	-	5
4_1	FPGA	64	36	34,2
4_2	FPGA	59	34,75	32,7
Board5	Intel	-	-	5
5_1	iX	39	40,6	39,5
5_2	iX	42	41,8	41
5_EthC	Ethernet-Controller	-	-	1

Tabelle 6.21: Validierungsergebnisse für Temperatur & Energieverbrauch nach dem Wechsel des Validierungszielverfahrens - Teil 2

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board6	Intel	-	-	5
6_1	iX	36	39,4	38
6_Disc	SSD	-	-	4
6_EthC	Ethernet-Controller	-	-	1
Summe				**448,35**

Tabelle 6.22: Validierungsergebnisse der restlichen Validierungsziele nach dem Wechsel des Validierungszielverfahrens

Validierungsziel	**Prüfkriterium**	**Validierungs-ergebnis**
Gesamtverarbeitungszeit	$x \leq 1000ms$	$1152ms$
Datenkommunikation	$x \geq 3500MiBit/sec$	$5200MiBit/sec$
Systeminitialisierung	$x \leq 30sec$	$23sec$
Speicherkapazität	$x \leq 512GiB$	$450GiB$
Schreibrate Daten	$x \geq 260MiB/sec$	$500MiB/sec$
Genauigkeit	$Seeziel \leq 0,5m^2$	$0,37m^2$
Genauigkeit	$Abweichung_{1km} \leq 1m$	$0,7m$
Genauigkeit	$Abweichung_{20km} \leq 3m$	$2,1m$
Genauigkeit	$Abweichung_{200km} \leq 5m$	$4,7m$

Die Ergebnisse in den Tabellen 6.20, 6.21 und 6.22 zeigen die Auswirkungen des Verfahrenswechsels mit den detaillierten Informationen über die Komplexität der fachlichen Algorithmen. Die Gesamtverarbeitungszeit von 1152ms liegt über den geforderten 1000ms. Die Validierung der Architektur ist damit fehlgeschlagen.

Tabelle 6.23: Validierungsergebnisse für Temperatur & Energieverbrauch nach dem Wechsel der fachlichen Algorithmen - Teil 1

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board1	WHZ	-	-	5
1_1	FPGA	85	41,25	40,5
1_2	FPGA	84	41	40,2
Board2	WHZ	-	-	5
2_1	FPGA	78	39,5	38,4
2_2	FPGA	77	39,25	38,1
Board3	WHZ	-	-	5
3_1	FPGA	62	35,5	33,6
3_2	PowerPC	33	29,9	31,55
3_EthC	Ethernet-Controller	-	-	1
Board4	WHZ	-	-	5
4_1	FPGA	55	33,75	31,5
4_2	FPGA	49	32,25	29,7

Tabelle 6.24: Validierungsergebnisse für Temperatur & Energieverbrauch nach dem Wechsel der fachlichen Algorithmen - Teil 2

Name	**Hardware**	**Auslastung in [%]**	**Temperatur in [°C]**	**Energie-verbrauch in [W]**
Board5	Intel	-	-	5
5_1	iX	37	39,8	38,5
5_2	iX	42	41,8	41
5_EthC	Ethernet-Controller	-	-	1
Board6	Intel	-	-	5
6_1	iX	36	39,4	38
6_Disc	SSD	-	-	4
6_EthC	Ethernet-Controller	-	-	1
Summe				**438,05**

Ein Blick auf die Ergebnisse des Validierungsziels Genauigkeit zeigt dem Architekten, dass diese noch unter den geforderten Werten liegen. Die Verwendung von fachlichen Algorithmen mit geringerem Genauigkeitsgrad könnte die Gesamtverarbeitungszeit und damit auch die Auslastung der Verarbeitungseinheiten reduzieren. Der Architekt sucht zusammen mit den Entwicklern fachliche Algorithmen mit einer geringeren Genauigkeit aus und modelliert deren Komplexitätswerte an die entsprechenden Software-Komponenten im Validierungsmodell. Anschließend führt er die Validierung erneut durch. Die Ergebnisse sind in den Tabellen 6.23, 6.24 und 6.25 aufgelistet. Die Auswertung gegenüber den Prüfkriterien führt zu einer vali-

den System-Architektur. Da die Änderung der fachlichen Algorithmen sich nur auf Änderungen an validierungsspezifischen Daten ausgewirkt haben, sind keine Informationen zwischen dem Validierungs- und Entwicklungsmodell zu synchronisieren.

Tabelle 6.25: Validierungsergebnisse der restlichen Validierungsziele nach dem Wechsel der fachlichen Algorithmen

Validierungsziel	**Prüfkriterium**	**Validierungs-ergebnis**
Gesamtverarbeitungszeit	$x \leq 1000ms$	$967ms$
Datenkommunikation	$x \geq 3500MiBit/sec$	$5200MiBit/sec$
Systeminitialisierung	$x \leq 30sec$	$23sec$
Speicherkapazität	$x \leq 512GiB$	$450GiB$
Schreibrate Daten	$x \geq 260MiB/sec$	$500MiB/sec$
Genauigkeit	$Seeziel \leq 0,5m^2$	$0,42m^2$
Genauigkeit	$Abweichung_{1km} \leq 1m$	$0,85m$
Genauigkeit	$Abweichung_{20km} \leq 3m$	$2,8m$
Genauigkeit	$Abweichung_{200km} \leq 5m$	$4,7m$

In diesem Kapitel wurde die Anwendung des Validierungsprozesses auf ein Radar-System demonstriert. Die möglichen auftretenden Änderungen an den Anforderungen, an der Architektur oder an den Validierungszielverfahren wurden anhand von Beispielen verdeutlicht. Auch die daraus resultierenden Änderungen an den Validierungszielen, am Entwicklungs- und Validierungsmodell wurden vorgestellt. Die Durchführung einiger Prozessschritte wie beispielsweise die Verwaltung der Validierungsziele und der Validierungszielverfahren, die Transformation von Änderungen vom Validierungsmodell in das Entwicklungsmodell sowie die Verwendung der Validierungs-

modelldaten für die Simulation wurden bewusst nur abstrakt beschrieben. Diese können natürlich vom Architekten per Hand erledigt werden, der in der Arbeit vorgestellte Ansatz sieht hier aber eine Werkzeugunterstützung vor. Details hierzu werden im nachfolgenden Kapitel beschrieben.

7 Werkzeugunterstützung

Der in der Arbeit vorgestellte Ansatz unterstützt den Architekten bei seiner Aufgabe, eine zu den System-Anforderungen konforme Architektur zu erstellen. Er bietet ein systematisches Vorgehen an, dessen Prozessschritte zum Teil durch Werkzeuge unterstützt werden können (siehe auch [PGQ12]). Auf diese Weise kann der Arbeitsaufwand des Architekten verringert werden, wodurch die Architekturvalidierung und damit auch die gesamte Systementwicklung effizienter wird. Kapitel 7.1 beschreibt einen Werkzeugprototypen, der die Möglichkeiten zur Automatisierung des Validierungsprozesses und zur Unterstützung des Architekten bei der Architekturvalidierung demonstriert. Kapitel 7.2 beschreibt ein Konzept für einen modellgetriebenen Simulator, der mit den Daten aus einem UML-Modell - in Verbindung mit dem Validierungsprozess handelt es sich um das Validierungsmodell - zu einem domänenspezifischen Simulator wird. Dieser Simulator kann als Validierungszielverfahren für ein oder mehrere Validierungsziele verwendet werden. Kapitel 7.3 gibt einen Ausblick für die Werkzeugunterstützung, indem es weitere Konzepte und Möglichkeiten zur Unterstützung des Architekten bei der Durchführung des Validierungsprozesses skizziert.

7.1 MEASURED

Dieses Kapitel stellt den Werkzeugprototyp Modellgetriebene Validierung von System-Architekturen mit der UML (MEASURED) vor, der die Möglichkeiten zur Automatisierung des Prozesses und zur Unterstützung des Architekten bei der Architekturvalidierung demonstriert. Dazu gibt Kapitel 7.1.1 zunächst eine Übersicht über die Funktionalitäten und die benachbar-

ten Systeme bevor die Kapitel 7.1.2 bis 7.1.6 auf die implementierten Funktionalitäten eingehen.

7.1.1 Übersicht

Der Validierungsprozess bietet an mehreren Stellen die Möglichkeit, den Architekten bei der Durchführung der einzelnen Prozessschritte zu unterstützen. Die Unterstützung konzentriert sich auf wiederholende sowie zeit- und arbeitsintensive Tätigkeiten. Hierunter fallen

- die Verwaltung der Validierungsziele,
- die Verbindung voneinander abhängiger Validierungsziele,
- die Verbindung voneinander abhängiger Anforderungen und Validierungsziele,
- die Auswertung von Abhängigkeitsbeziehungen,
- die Transformation von Elementen zwischen Validierungs- und Entwicklungsmodell,
- die Durchführung von Simulationen auf Basis der im Validierungsmodell enthaltenen Daten sowie
- die Auswertung der Validierungsergebnisse zur Feststellung der Architekturvalidität.

Zum Nachweis der Realisierbarkeit dieser Unterstützungsmöglichkeiten wurde ein Werkzeugprototyp entwickelt.

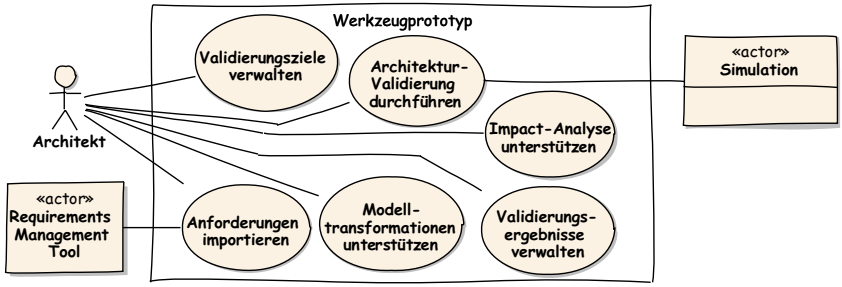


Abbildung 7.1: Use Case-Diagramm des Werkzeugprototypen

Das Use Case-Diagramms in Abbildung 7.1 zeigt die vom Werkzeugprototypen für den Architekten bereitgestellten Funktionalitäten sowie die beteiligten externen Systeme. Es gibt zum einen das Requirements-Management-Tool und zum anderen die Simulation. Bei der Simulation handelt es sich um einen Stellvertreter für alle als Validierungszielverfahren verwendeten Simulationen, die als Datenbasis das Validierungsmodell verwenden und ihre Ergebnisse dem Unterstützungswerkzeug zur Ermittlung der Architekturvalidität zur Verfügung stellen. Details zu solchen UML-basierten Validierungszielverfahren enthält Kapitel 7.2. Der Architekt kann den Werkzeugprototypen zur Verwaltung der Validierungsziele verwenden, worauf Kapitel 7.1.2 näher eingeht. Um diese mit den Anforderungen verbinden zu können, muss der Architekt zunächst die Anforderungen in das Werkzeug importieren (siehe Kapitel 7.1.3). Der Prototyp bietet darüber hinaus Unterstützung bei der Modelltransformation an, die der Architekt zur Erstellung des Validierungsmodells, zur Übernahme von Modellelementen aus dem Entwicklungsmodell oder zur Synchronisation von Entwicklungs- und Validierungsmodell nutzen kann. Details hierzu sind in Kapitel 7.1.4 beschrieben. Kapitel 7.1.5 beschreibt wie der Architekt mit Hilfe des Werkzeugs die Validierungszielverfahren startet und deren Ergebnisse für die Validierung

übernimmt. Kapitel 7.1.6 beschreibt wie das Werkzeug den Architekten bei der Impact Analyse unterstützt. Zu guter Letzt beschreibt Kapitel 7.3 einige weitere Möglichkeiten zur Unterstützung des Architekten, die im Rahmen des Werkzeugprototypen nur partiell oder nicht realisiert wurden.

7.1.2 Validierungsziele verwalten

Die Grundlage für eine grafische Anzeige der architekturspezifischen Sicht sowie der Impact-Analyse ist die Verwaltung der Validierungsziele. Ein Validierungsziel wird in dem Werkzeug mit einem Namen, Prüfkriterien, zugeordneten Anforderungen und einem Validierungszielverfahren angelegt. Die Eingabemaske für das Erstellen von Validierungszielen ist in Abbildung 7.2 zu sehen. Das Prüfkriterium ergibt sich durch einen Vergleichswert sowie einem Vergleichsoperator und wird von dem Werkzeug nicht automatisch aus den Anforderungen ermittelt, sondern muss durch den Architekten eingegeben werden. Eine automatische Ermittlung setzt formalisierte Anforderungen[95] voraus, die in der Praxis nur selten vorhanden sind. Das Validierungszielverfahren wird durch die Angabe einer ausführbaren Datei und durch optionale Startparameter dem Validierungsziel zugeordnet[96]. Der Dialog für die Auswahl der Anforderungen eines Validierungszieles ist in Abbildung 7.3 abgebildet.

[95] Sprachen, mit denen formalisierte Anforderungen erstellt werden können, sind z. B. Alloy [Jac12] oder Z [PST97].

[96] Grundsätzlich können einem Validierungsziel mehrere Verfahren zugeordnet werden. Der realisierte Prototyp unterstützt allerdings nur ein Verfahren.

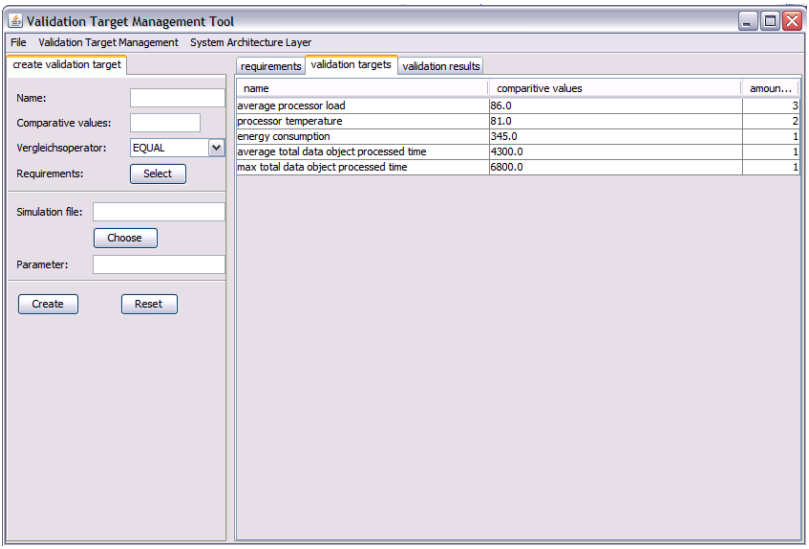


Abbildung 7.2: Maske für das Erstellen eines Validierungszieles

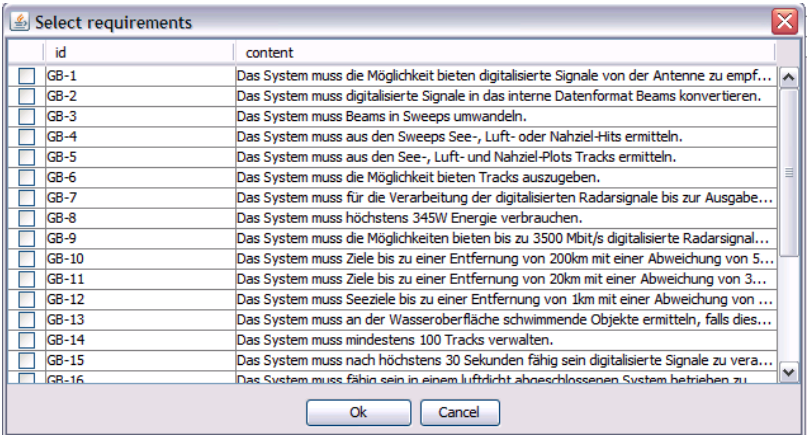


Abbildung 7.3: Maske zur Auswahl der Anforderungen für ein Validierungsziel

In der Liste von Abbildung 7.2 ist für jedes Validierungsziel die Anzahl der zugeordneten Anforderungen (dritte Spalte mit Namen *amount*) zu sehen, so dass der Architekt ein Validierungsziel ohne zugeordnete Anforderungen identifizieren kann, ohne den in Abbildung 7.3 dargestellten Dialog aufrufen zu müssen.

Auf Basis der eingegebenen Validierungsziele ermöglicht das Werkzeug die Visualisierung der architekturspezifischen Sicht, die in Abbildung 7.4 dargestellt ist. Aus der Liste der Validierungsziele kann sich der Architekt Details zu jedem Validierungsziel anzeigen lassen. Dazu zählen die Prüfkriterien und die zugeordneten Anforderungen sowie die letzten Ergebnisse des Validierungszielverfahrens. Anhand der Hintergrundfarbe erkennt der Architekt auf einen Blick den Validierungsstatus, wobei rot für ein nicht valides (average processor load, processor temperature und average total data object processed time) und grün für ein valides Validierungsziel (energy consumption) verwendet wird. Ein gelber Hintergrund weist den Architekten darauf hin, dass für dieses Validierungsziel mit den zugeordneten Anforderungen und Verfahren kein gültiges Validierungsergebnis existiert (max total data object processed time). Ein Validierungsergebnis wird ungültig, wenn sich die zugeordneten Anforderungen selbst ändern, neue hinzukommen, vorhandene entfernt oder das Verfahren ausgetauscht wird. Dies ist für den Architekten ein Hinweis darauf, dass er mindestens dieses Ziel revalidieren muss[97].

[97] Genauer gesagt müssen alle Validierungsziele, die mit diesem in einer direkten oder indirekten Abhängigkeitsbeziehung stehen, revalidiert werden, da das Validierungsergebnis für die Architektur ansonsten auf Basis inkonsistenter Validierungsergebnisse erstellt wird.

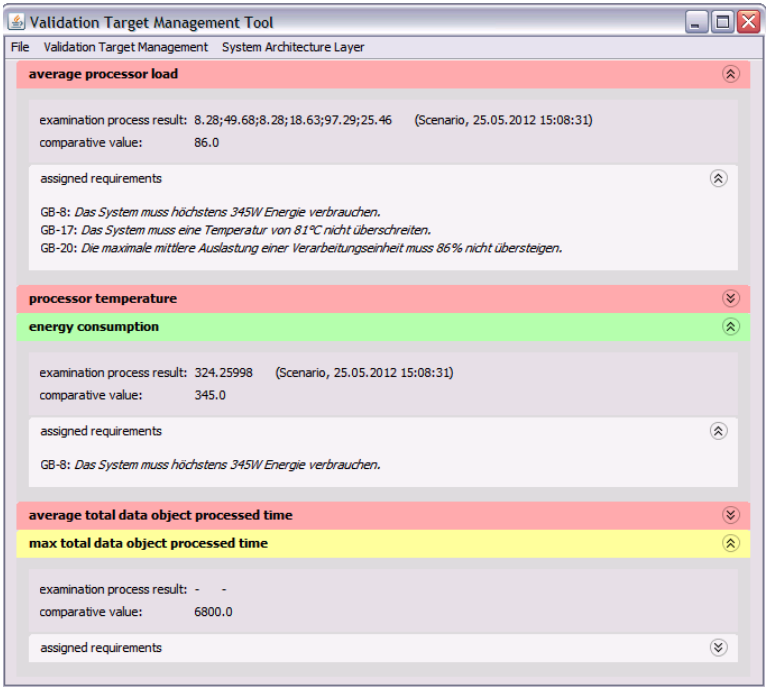


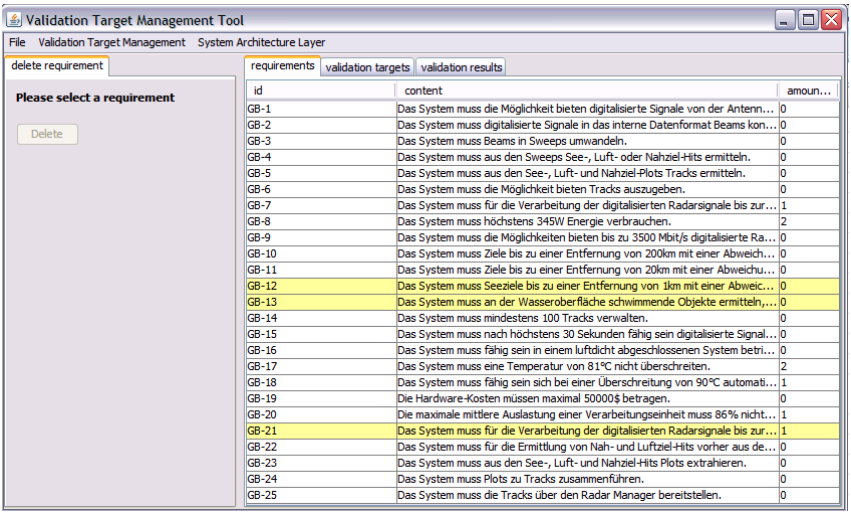
Abbildung 7.4: Grafische Darstellung der architekturspezifischen Sicht

7.1.3 Anforderungen importieren

Die Verwaltung von Anforderungen zählt nicht zu den Aufgaben des Werkzeugprototypen. Hierfür gibt es bereits spezielle Programme[98] mit einem großen Funktionsumfang, beispielsweise Rational DOORS[99], Cradle[100] oder Caliber [101] bzw. weniger spezielle wie Microsoft Excel[102], das in der Praxis

[98] [CGNA+11] enthält einen Vergleich verschiedener Werkzeuge zur Anforderungsverwaltung.
[99] http://www-142.ibm.com/software/ products/de/de/ratidoor
[100] http://www.threesl.com/
[101] http://www.borland.com/products/caliber/
[102] http://office.microsoft.com/de-de/excel/

immer noch häufig verwendet wird. Um die Anforderungen für die Verwaltung der Validierungsziele verwenden zu können, müssen diese importiert werden. Die Veränderung der Anforderungen im Werkzeug ist nicht vorgesehen, es handelt sich also um einen unidirektionalen Datenfluss.



id	content	amoun…
GB-1	Das System muss die Möglichkeit bieten digitalisierte Signale von der Antenn…	0
GB-2	Das System muss digitalisierte Signale in das interne Datenformat Beams kon…	0
GB-3	Das System muss Beams in Sweeps umwandeln.	0
GB-4	Das System muss aus den Sweeps See-, Luft- oder Nahziel-Hits ermitteln.	0
GB-5	Das System muss aus den See-, Luft- und Nahziel-Plots Tracks ermitteln.	0
GB-6	Das System muss die Möglichkeit bieten Tracks auszugeben.	0
GB-7	Das System muss für die Verarbeitung der digitalisierten Radarsignale bis zur…	1
GB-8	Das System muss höchstens 345W Energie verbrauchen.	2
GB-9	Das System muss die Möglichkeiten bieten bis zu 3500 Mbit/s digitalisierte Ra…	0
GB-10	Das System muss Ziele bis zu einer Entfernung von 200km mit einer Abweich…	0
GB-11	Das System muss Ziele bis zu einer Entfernung von 20km mit einer Abweichu…	0
GB-12	Das System muss Seeziele bis zu einer Entfernung von 1km mit einer Abweic…	0
GB-13	Das System muss an der Wasseroberfläche schwimmende Objekte ermitteln,…	0
GB-14	Das System muss mindestens 100 Tracks verwalten.	0
GB-15	Das System muss nach höchstens 30 Sekunden fähig sein digitalisierte Signal…	0
GB-16	Das System muss fähig sein in einem luftdicht abgeschlossenen System betri…	0
GB-17	Das System muss eine Temperatur von 81°C nicht überschreiten.	2
GB-18	Das System muss fähig sein sich bei einer Überschreitung von 90°C automati…	1
GB-19	Die Hardware-Kosten müssen maximal 50000$ betragen.	0
GB-20	Die maximale mittlere Auslastung einer Verarbeitungseinheit muss 86% nicht…	1
GB-21	Das System muss für die Verarbeitung der digitalisierten Radarsignale bis zur…	1
GB-22	Das System muss für die Ermittlung von Nah- und Luftziel-Hits vorher aus de…	0
GB-23	Das System muss aus den See-, Luft- und Nahziel-Hits Plots extrahieren.	0
GB-24	Das System muss Plots zu Tracks zusammenführen.	0
GB-25	Das System muss die Tracks über den Radar Manager bereitstellen.	0

Abbildung 7.5: Importierte Anforderungen mit Markierung zum Erkennen von Veränderungen

Die Schnittstelle muss fähig sein, die Anforderungen aus unterschiedlichen Programmen in das Verwaltungswerkzeug zu importieren. Hierfür eignet sich das Requirements Interchange Format (ReqIF) [Obj11a] von der OMG, das auf einem Extensible Markup Language (XML)-Schema basiert. Falls das Anforderungsmanagement-Werkzeug ReqIF nicht unterstützt, ist der Import über eine Excel-Schnittstelle möglich. Da bei den verschiedenen Programmen bzw. bei der nativen Verwendung von Excel als Anforderungsmanagement-Werkzeug unterschiedliche Datenstrukturen

verwendet werden, ist eine anwendungsspezifische Anpassung der Schnittstelle erforderlich.

Abbildung 7.5 zeigt die Auflistung der Anforderungen im Werkzeug. Neben der eindeutigen Kennung und dem Anforderungstext selbst wird in der dritten Spalte auch die Anzahl der Verbindungen zu Validierungszielen angezeigt. Die Anforderungen werden vom Verwaltungswerkzeug persistiert, wodurch bei einem erneuten Import der Anforderungen Veränderungen festgestellt werden können. Diese Anforderungen werden gelb markiert, so dass sie vom Architekten identifiziert werden können. Welche Validierungsziele von der Veränderung potentiell betroffen sind, lässt sich mit der in Abbildung 7.6 dargestellten Sicht ermitteln. Hier werden für jede Anforderung die zugeordneten Validierungsziele angezeigt.

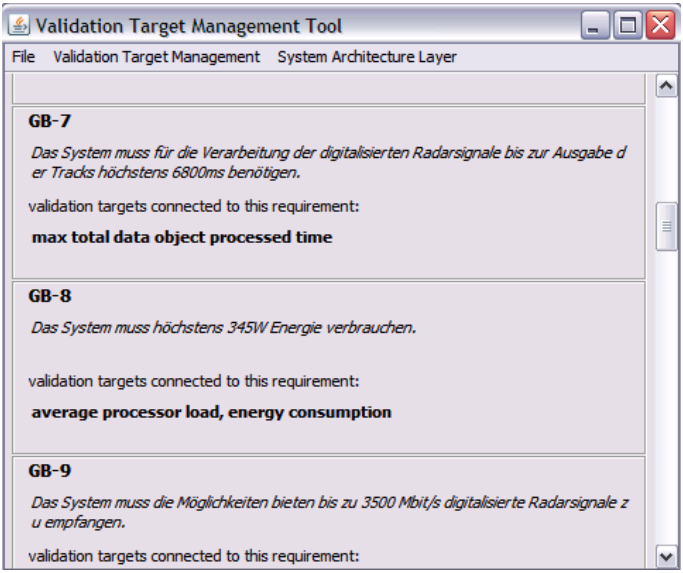


Abbildung 7.6: Anforderungen mit zugeordneten Validierungszielen

Es liegt in der Verantwortung des Architekten zu entscheiden, ob eine Veränderung tatsächlich Auswirkungen auf die zugeordneten Validierungsziele hat. Dies kann gerade bei natürlichsprachlichen Anforderungen nicht von einem Programm entschieden werden.

7.1.4 Modelltransformationen unterstützen

Aufgrund der Trennung von entwicklungs- und validierungsspezifischen Daten in zwei verschiedene UML-Modelle müssen Modelldaten an verschiedenen Stellen im Validierungsprozess synchronisiert oder selektiv vom einen in das andere Modell übernommen werden. Beide Modelle basieren auf demselben Metamodell (siehe Voraussetzungen in Kapitel 4.2). Eine Transformationen ist deshalb mit relativ geringem Aufwand möglich, weil kein Mapping der Elemente aus Metamodell A auf Metamodell B erforderlich ist[103]. Zwar verwendet das Validierungsmodell UML-Profile[104] zur Modellierung validierungsspezifischer Daten[105], da diese Daten aber weder aus dem Entwicklungsmodell übernommen, noch in dieses zurück transformiert werden müssen, stellt dieser semantische Unterschied kein Problem für die Modelltransformation dar.

Die Voraussetzung einer UML-basierten Dokumentation der Entwicklungsinformationen ist für den Validierungsprozess nicht zwingend erforderlich. Sie erleichtert allerdings die Modelltransformationen. Der Validierungsprozess setzt im Validierungsmodell auf die UML als Modellierungssprache.

[103] Detaillierte Informationen zur Transformation von Modellinhalten (auch auf Basis verschiedener Metamodelle) und ein Vergleich vorhandener Werkzeuge haben Rose et al. [RHW+10] zusammengestellt.

[104] Noyrit et al. beschreiben in [NGTS10], wie mit mehreren Profilen eine konsistente Modellierung möglich ist.

[105] Alternative Notationsmöglichhkeiten für die validierungsspezifischen Daten sind in Kapitel 8.2 beschrieben.

Falls die entwicklungsspezifischen Daten auf einem anderen Metamodell basieren, muss zunächst eine Abbildung der Modellelemente aus diesem Metamodell auf die Elemente des UML-Metamodells durchgeführt werden. Der umgekehrte Weg ist bei der Übertragung von entwicklungsrelevanten Daten aus dem Validierungsmodell notwendig, falls beispielsweise der Architekt Veränderungen an der System-Architektur vorgenommen hat.

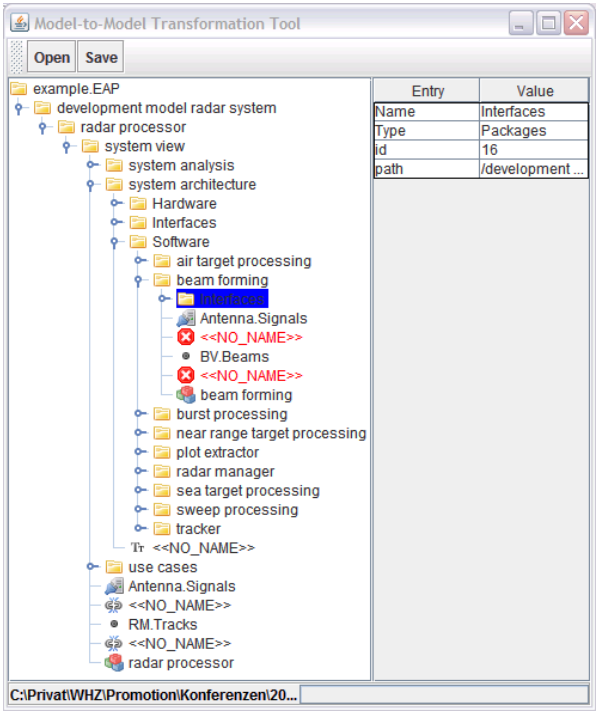


Abbildung 7.7: Oberfläche für die Auswahl der zu transformierenden Modellelemente

Ausgehend von der Transformation von Modellelementen auf Basis desselben Metamodells unterstützt der Werkzeugprototyp den Architekten bei der

initialen Erstellung des Validierungsmodells. Dazu liest er den Modellbaum des Entwicklungsmodells ein und erlaubt dem Architekten die zu transformierenden Elemente auszuwählen. Die dazugehörige grafische Oberfläche im Werkzeug ist in Abbildung 7.7 abgebildet. Als Modellierungswerkzeug wurde der Enterprise Architect (EA) ausgewählt, da dieser in den Entwicklungsprojekten eingesetzt wird. Er verwendet ein proprietäres Dateiformat und erlaubt beispielsweise über eine Java-Schnittstelle den Zugriff auf die Modellelemente. Die Schnittstelle ist schlecht bis gar nicht dokumentiert, nicht gerade performant und auch die interne Modellstruktur ist gewöhnungsbedürftig. Dies erschwerte die Entwicklung des Prototypen, so dass die Entwicklung nicht über die Demonstration, dass eine initiale Erstellung des Validierungsmodells mit ausgewählten Elementen des Entwicklungsmodells für den EA möglich ist, hinaus gegangen ist.

7.1.5 Architekturvalidierung durchführen

Der Architekt muss jedem Validierungsziel bei der Erstellung mindestens ein Validierungszielverfahren zuordnen, mit dem es validiert werden kann. Dazu muss es Werte berechnen, die mit dem gespeicherten Prüfkriterium verglichen werden. Auf diese Weise kann das Werkzeug die Validität der Validierungsziele und damit der gesamten Architektur ermitteln. Damit das Verwaltungswerkzeug diese Werte von beliebigen Validierungszielverfahren einlesen kann, ist eine einheitliche Schnittstelle erforderlich. Diese Schnittstelle wird vom Werkzeug vorgegeben und muss von allen Verfahren verwendet werden. Für die Realisierung der Schnittstelle wird die programmiersprachenunabhängige Beschreibungsprache XML verwendet.

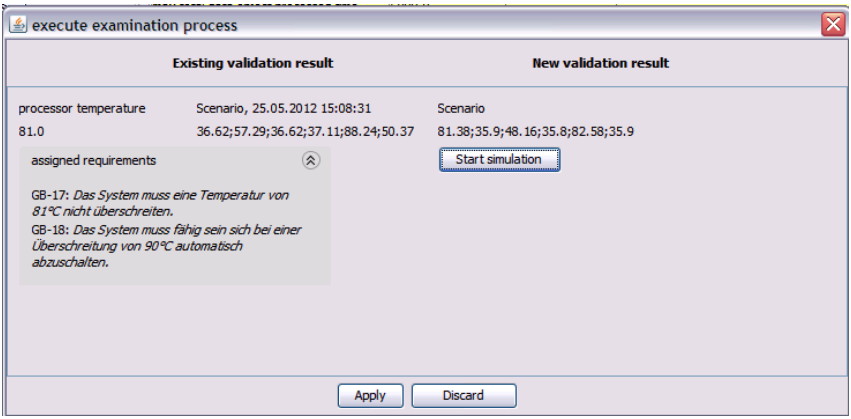


Abbildung 7.8: Maske zur Durchführung eines Validierungszielverfahrens

Für eine bessere Benutzbarkeit des Ansatzes kann der Architekt das Verfahren eines Validierungsziels aus dem Verwaltungswerkzeug heraus starten. Über den Button *Simulation starten* in Abbildung 7.8 wird das zugeordnete Validierungszielverfahren ausgeführt. War die Simulation erfolgreich, werden die neuen Ergebnisse über die XML-Schnittstelle importiert und dem Architekten auf der rechten Seite zum Vergleich mit den vorhanden Ergebnissen und dem Prüfkriterium angezeigt. Der Architekt kann nun entscheiden, ob er die Ergebnisse übernehmen oder verwerfen möchte. Stellt ein Verfahren Ergebnisse für mehr als ein Validierungsziel bereit, werden diese bei Annahme der Ergebnisse automatisch übernommen. Auf diese Weise muss der Architekt das Verfahren nicht für jedes Ziel einzeln und damit mehrfach durchführen. Die Übernahme aller Ergebnisse dient aber nicht nur dazu, dem Architekten Zeit zu sparen, sondern vor allem zur Verhinderung von Dateninkonsistenzen. Voneinander abhängige Validierungsziele lassen sich nur sinnvoll als valide kennzeichnen, falls die Ergebnisse auf Basis derselben Simulationswerte ermittelt werden (siehe Kapitel 7.2). Die

Missachtung von Abhängigkeiten und die Verwendung inkonsistenter Validierungsergebnisse haben falsche Schlussfolgerungen auf die Architekturvalidität zur Folge.

7.1.6 Impact-Analyse unterstützen

Die Möglichkeiten zur Unterstützung des Architekten bei der Impact-Analyse sind in den letzten Kapiteln zum Teil bereits implizit erwähnt worden. Veränderungen an den Anforderungen werden durch das Werkzeug in der Anforderungsübersicht (siehe Abbildung 7.5) durch eine gelbe Markierung angezeigt. Eine weitergehende Unterstützung, beispielsweise der Abgleich der Prüfkriterien, ist mit natürlichsprachlichen Anforderungen nicht möglich. Welche Auswirkungen die Änderungen an den Prüfkriterien haben, kann nur durch die Validierungszielverfahren und durch den Menschen analysiert werden. Das Werkzeug zeigt dem Architekten auch die Veränderung von Validierungszielen durch einen Farbwechsel an. Sobald der Architekt ein Validierungsziel verändert, sei es der Name, die zugeordneten Anforderungen oder das Validierungszielverfahren, sind die Validierungsergebnisse ungültig, was durch eine gelbe Markierung signalisiert wird.

Zur architekturspezifischen Analyse der Anforderungen für den Entwurf oder die Verbesserung einer Architektur sowie zur statischen Impact-Analyse (siehe Kapitel 4.3.6) bietet das Werkzeug eine Übersicht der Abhängigkeiten zwischen den Validierungszielen an (siehe Abbildung 7.9). Diese ergibt sich über die zugeordneten Anforderungen. Dabei erkennt das Werkzeug nicht nur direkte Abhängigkeiten zweier Validierungsziele über eine oder mehrere gemeinsame Anforderungen, sondern auch indirekte Abhängigkeiten, die sich durch die Abhängigkeit der beiden Validierungsziele von einem dritten Validierungsziel ergeben.

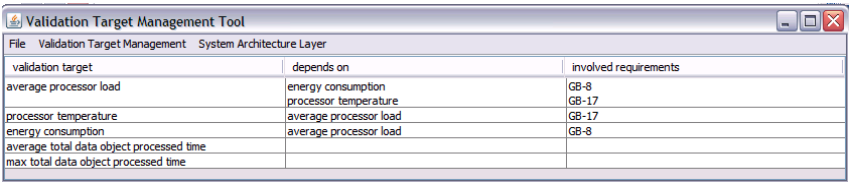


Abbildung 7.9: Anzeige der Abhängigkeiten von Validierungszielen durch die zugeordneten Anforderungen

Die schematische Darstellung einer solchen indirekten Abhängigkeit zeigt die Abbildung 7.10. Dem *Validierungsziel 1* sind *NFA1* und *NFA2* zugeordnet, *Validierungsziel 3* hat eine Verbindung zu *NFA3* und *NFA4*. Da die NFAs 2 und 3 dem *Validierungsziel 2* zugeordnet sind, ergibt sich über dieses eine Verbindung zwischen Validierungsziel 1 und 3. Indirekte Abhängigkeiten sind für die Architekturvalidierung von großer Bedeutung, da sich durch diese die nicht direkt ersichtlichen Auswirkungen von Veränderungen auf die verschiedenen architekturspezifischen Aspekte ermitteln lassen. Durch diese indirekten Abhängigkeiten ist der Architekt in der Lage voneinander abhängige Validierungsziele zu identifizieren, deren Validierungszielverfahren ebenfalls voneinander abhängig sind. Hier empfiehlt es sich, die Verfahren in einer gemeinsamen Simulation durchzuführen. Weitere Details hierzu sind in Kapitel 7.2 beschrieben.

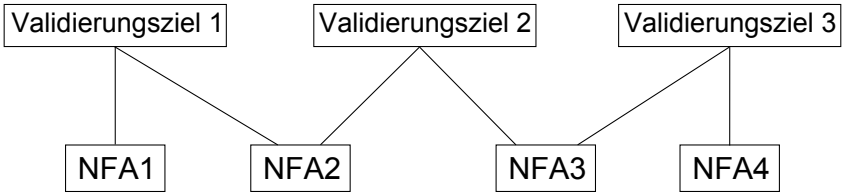


Abbildung 7.10: Schematische Darstellung von indirekten Abhängigkeiten zwischen Validierungszielen

Veränderungen an der Architektur kann das Werkzeug nicht von selbst erkennen. Es wäre zwar ein Abgleich von Dateiattributen (z. B. das Datum der letzten Änderung) möglich, allerdings ist dies sehr ungenau, da sich dabei nicht unbedingt der validierungsrelevante Inhalt des Modells geändert haben muss. Bereits das Bearbeiten von Modelldiagrammen führt zu einer Dateiänderung, was aber keine semantische Auswirkung auf die Validierungsergebnisse hat.

Für die Impact-Analyse von Architekturveränderungen bietet das Werkzeug dem Architekten eine vollständige Revalidierung und eine zweistufige Analyse für ein einzelnes Validierungsziel an. Bei der vollständigen Validierung werden alle Verfahren durchgeführt. Anschließend werden anhand der neuen Ergebnisse und dem Vergleich mit den Prüfkriterien die neuen Status der Validierungsziele ermittelt. Der Status eines jeden Zieles wird dabei durch eine farbige Markierung, z. B. in der architekturspezifischen Sicht (siehe Abbildung 7.4), angezeigt. Bei der zweistufigen Analyse kann der Architekt über die in Abbildung 7.8 dargestellte Maske das Verfahren eines Zieles starten und anschließend die Ergebnisse mit den vorhandenen vergleichen (Stufe 1). Dadurch bleibt der Fokus des Architekten zunächst auf dem Validierungsziel, das er durch die Architekturveränderung beeinflussen möchte. Erst wenn er mit den Ergebnissen zufrieden ist, übernimmt er diese und bekommt durch die farbigen Markierungen der Validierungsziele die Auswirkungen auf andere Validierungsziele angezeigt (Stufe 2).

7.2 Konzept für einen generischen Simulator

Zur Unterstützung des Architekten soll der Validierungsprozess eine modellgetriebene Validierung der System-Architektur[106] ermöglichen. Dabei dient das Validierungsmodell als Datenquelle für die verschiedenen Validierungszielverfahren. Als Modellierungssprache für das Validierungsmodell wurde die UML ausgewählt, bei der Validierungsmethode setzt der in der Arbeit vorgestellte Ansatz auf Simulationen.

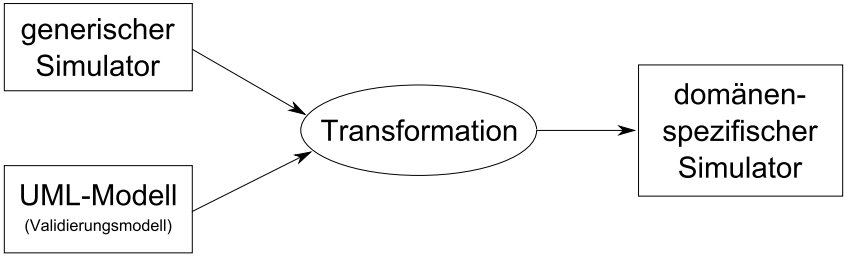


Abbildung 7.11: Transformation des generischen Simulators zu einem domänenspezifischen Simulator

Zur Verdeutlichung der Nutzungsmöglichkeiten modellierter Daten stellt dieses Kapitel das Konzept für einen generischen, UML-basierten Simulator vor. Dieser *Rohsimulator* lässt sich mit Hilfe von Daten aus einem UML-Modell[107] zu einem domänenspezifischen Simulator transformieren (siehe Abbildung 7.11), der zusammen mit dem Werkzeugprototypen MEASURED verwendet werden kann. Der Simulator besteht aus verschiedenen Komponenten (z. B. für die Ein- und Ausgabe von Daten), darunter auch die eigentliche Simulation des Systems.

[106] Für eine detaillierte Beschreibung der Begriffe modellgetriebene Validierung wird auf Kapitel 2.9 verwiesen.

[107] Im Kontext des Validierungsprozesses entspricht dies dem Validierungsmodell.

Für die Konzeptentwicklung des generischen Simulators wurden die folgenden Ziele festgelegt:

- Strukturelle und fachliche Konfiguration des simulierten Systems über ein UML-Modell,
- Abbildung der Semantik von Modellelementen auf Elemente der Simulation,
- Verwendung des Simulators in verschiedenen Entwicklungsphasen,
- Bereitstellung der Simulationsergebnisse für den Werkzeuprototypen MEASURED,
- Bereitstellung von Zwischenergebnissen der Simulation für eine Analyse der Validierungsergebnisse und
- Wiederverwendung der Ein- und Ausgabe-Komponenten für andere Validierungszielverfahren.

Die Bereitstellung der Simulationsergebnisse für den Werkzeugprototypen MEASURED wurde über eine Serialisierung der Daten in eine XML-Datei realisiert. Hierfür wurde die vom Werkzeugprototypen MEASURED definierte XML-Schnittstelle für Validierungsergebnisse verwendet (siehe Kapitel 7.1.5). Die Wiederverwendung der Ein- und Ausgabe-Komponenten für andere Validierungszielverfahren wird über vom Modellierungswerkzeug unabhängige Software-Komponenten gelöst. Details hierzu sind in Kapitel 7.2.1 beschrieben. Das Kapitel geht außerdem auf die eigentliche Simulation des Systems ein. Diese bietet dem Architekten eine domänenunabhängige und erweiterbare Simulationsimplementierung an, die bereits in einer frühen Phase der Systementwicklung verwendet werden kann.

Der Architekt kann mit Hilfe eines UML-Modells sowohl die Architektur des simulierten Systems als auch die benötigten Daten für die fachlichen Algo-

rithmen festlegen. Für diese strukturelle und fachliche Konfiguration muss der Architekt zunächst dem Simulator die Semantik der Modellelemente bekannt machen (siehe Kapitel 7.2.2). Mit diesem Wissen wird das zu simulierende System initialisiert und anschließend die Simulation durchgeführt. Die Abbildung der modellierten Daten auf die Elemente der Simulation sowie deren Verwendung für die Simulation selbst werden in Kapitel 7.2.3 anhand verschiedener Beispiele erläutert.

Simulationen haben gegenüber formalen Methoden u. a. den Vorteil, dass Details während des Simulationsablaufs protokolliert werden können. Formale Verfahren verwenden zur Beschreibung des zu untersuchenden Systems mathematische Ausdrücke, meist temporale Logiken (siehe auch Kapitel 5.1.2). Bei der Durchführung wird die Gültigkeit der miteinander verknüften Ausdrücke überprüft. Simulationen hingegen beschreiben das Verhalten des zu untersuchenden Systems mit Hilfe von Algorithmen. Die Eingangswerte für die Algorithmen werden anhand ausgesuchter Szenarien bereitgestellt. Die berechneten Ergebnisse werden mit Sollwerten verglichen, wodurch die Einhaltung der geforderten Eigenschaften überprüft wird. Neben den Ergebnissen und deren Einhaltung können aber auch Zwischenschritte der Berechnungen protokolliert werden, die dem Architekten bei der Analyse einer nicht validen Architektur helfen können (siehe Anzahl der Datenpakete in Ein- und Ausgangspuffern in ausgesuchten Software-Komponenten in Tabelle 6.7 auf Seite 183). Wie der Simulator interne Daten für den Architekten protokolliert, so dass sie weder zu grob noch zu feingranular sind, beschreibt Kapitel 7.2.4.

Als Referenzimplementierung dieses Konzepts wurde eine Simulation für die Domäne Radar-Systeme, genauer für die Radarsignalverarbeitung (siehe Kapitel 3), entwickelt. Diese Simulation diente als Validierungszielverfahren

für die Validierungsziele Gesamtverarbeitungszeit, Energieverbrauch, Betriebstemperatur sowie Datenkommunikation inkl. deren Abhängigkeiten untereinander (siehe Kapitel 6.3). Für ausführliche Details zu den verwendeten Algorithmen sowie zur Umsetzung des hier vorgestellten Konzepts wird auf [Poß12] verwiesen.

7.2.1 Software-Architektur des Simulators

Abbildung 7.12 zeigt die Software-Architektur des generischen Simulators. Er besteht aus fünf Software-Komponenten, wobei die Komponente *Simulation* die Logik des Rohsimulators enthält. Diese umfasst bei der Referenzimplementierung das Zusammenspiel von Software- und Hardware-Komponenten, die Datenkommunikation zwischen den Software-Komponenten sowie die Berechnung von Energieverbrauch und Betriebstemperatur der Verarbeitungseinheiten.

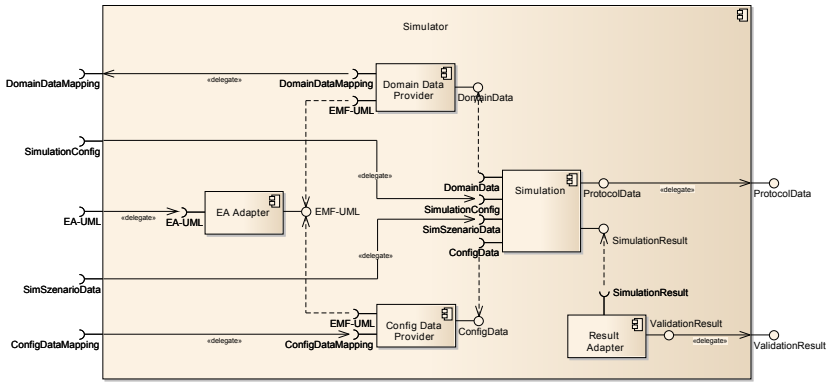


Abbildung 7.12: Software-Architektur des generischen Simulators

Dabei sind die Algorithmen bewusst einfach[108] gehalten, so dass der Simulator auch in frühen Phasen der Systementwicklung ohne große Kenntnis über die fachlichen Algorithmen verwendet werden kann. Diese einfachen Algorithmen können durch den Einsatz des Strategy-Patterns[109] ohne große Eingriffe in die übrige Logik ausgetauscht werden.

Die strukturellen und fachlichen Informationen werden der Komponente Simulation über die Komponenten *Config Data Provider* bzw. *Domain Data Provider* bereitgestellt. Diese wiederum extrahieren die Daten aus dem eingelesenen UML-Modell. Zur Bereitstellung der strukturellen und fachlichen Daten werden Mapping-Dateien benötigt, mit denen die im UML-Modell verwendete Semantik[110] auf die im Simulator verwendeten Elemente übertragen wird. Beispielsweise wird für eine Komponente mit dem Stereotype *software* eine Instanz der Klasse *SoftwareComponent* erzeugt (strukturelle Information) oder der Wert des TaggedValue *executionTime* (des Stereotypes *software*) wird verwendet, um das Attribut einer *SoftwareComponent*-Instanz zu belegen (fachliche Informationen). Weitere Details zum Mapping der Semantik sind in Kapitel 7.2.2 beschrieben. Das Mapping für die Simulation des Radar-Systems wird in Kapitel 7.2.3 vorgestellt.

Das UML-Modell liegt den beiden Providern in dem herstellerunabhängigen Eclipse Modelling Framework (EMF)-Format der Eclipse Foundation vor. Dieses Format wird von vielen Modellierungswerkzeugen unterstützt, so dass diese Komponenten werkzeugunabhängig sind. Für Modellie-

[108] Beispielsweise lässt sich die Verarbeitungszeit einer Software-Komponente für eine bestimmte Verarbeitungseinheit als konkreter Wert angeben.

[109] Das Strategy-Pattern erlaubt die Verwendung verschiedener Implementierungen für einen Algorithmus. Allgemeiner gesagt: Es unterstützt Familien von Algorithmen. Für weitere Informationen wird auf Gamma et al. [GHJV94] verwiesen.

[110] Im Kontext des Validierungsprozesses ist dies die in der VSN festgelegte Semantik der Modellelemente.

rungswerkzeuge, die keinen Export ihrer Modellinformationen in das EMF-Format unterstützen, muss ein entsprechender Adapter (Komponente *EA-Adapter*) erstellt werden. Dies ist beispielsweise für den Enterprise Architect (EA) erforderlich, der in den Projekten, anhand derer der Validierungsprozess entwickelt wurde (siehe Kapitel 6.1), eingesetzt wird. Die Transformation der Modellinformationen aus dem proprietären EA-Format in das EMF-Format erfolgt mit Hilfe des GeneSEZ[111]-Frameworks [HBGH08] [HBG+09] [Gen13]. Eine Beschreibung dieser Transformation ist in [Poß12] enthalten.

Damit die Simulationsergebnisse auch von MEASURED als Validierungsergebnisse genutzt werden können, müssen diese in das vordefinierte Format der XML-Schnittstelle (siehe Kapitel 7.1.5) konvertiert werden. Diese Aufgabe realisiert die Komponente *Result Adapter*.

7.2.2 Abbildung der Semantik von Modellelementen auf Elemente der Simulation

Um die im UML-Modell dokumentierten Daten für die Konfiguration der Simulation verwenden zu können, muss der Architekt eine Abbildung von den für die Simulation relevanten Modellelementen[112] auf die konfigurierbaren Elemente der Simulation festlegen. Diese Abbildung muss surjektiv sein: Jedem konfigurierbarem Simulationselement ist mindestens ein für die Simulation relevantes Modellelement zugeordnet. Beispielsweise instanziiert die Simulation für jede Komponente mit dem Stereotyp *software* eine Objekt der Klasse *SoftwareComponent*. Falls die Verarbeitungszeit ei-

[111] Generative Software Engineering Zwickau (GeneSEZ)

[112] Im Kontext des Validierungsprozesses ist dies eine Teilmenge der VSN, ggf. sind dies auch alle Elemente.

ner SoftwareComponent-Instanz ($executionTime_{SW}$) nach der in Formel 7.1 beschriebenen Berechnung ermittelt wird, benötigt die Simulation hierfür nicht nur die Komplexität der dazugehörigen Software-Komponente[113] ($complexity_{SW}$), sondern auch die Verarbeitungsleistung der ausführenden Verarbeitungseinheit ($performance_{HW_{SW}}$).

$$executionTime_{SW} = \frac{complexity_{SW}}{performance_{HW_{SW}}} \tag{7.1}$$

Für die Festlegung der Abbildung sind die folgenden Punkte zu berücksichtigen:

- Das Werkzeug folgt der Methode: Der Simulator macht keine Vorgaben bzgl. der zu verwendenden Notationselemente und deren Semantik.
- Es wird ermpfohlen, die von der UML definierte Semantik für die Modellelemente zu verwenden. Projektspezifische Anpassungen müssen allerdings möglich sein.
- Die Anpassung der Abbildung muss ohne Änderungen am Quellcode möglich sein.

Der Simulator soll dem Validierungsprozess nicht vorgeben, welche Modellelemente zur Dokumentation der validierungsspezifischen Daten im Validierungsmodell zu verwenden sind. Dasselbe gilt für die Semantik. Auf diese Weise wird sichergestellt, dass auch der Validierungsprozess keine Vorgaben bzgl. der Wahl der Modellelemente und ihrer Semantik an die entwicklungsspezifische Dokumentation des Systems (Entwicklungsmodell) machen muss. Dadurch kann der Validierungsprozess in bestehenden Projekten mit bereits festgelegten Modellierungsvorgaben eingesetzt werden.

[113] Dies ist die Kurzform für eine UML-Komponente mit dem Stereotyp *software*.

Eine Anpassung der entwicklungsspezifischen Notation ist hierfür nicht erforderlich[114].

Die UML wurde entwickelt, um für die Modellierung von Informationen eine gemeinsame Sprache als Kommunikationsbasis für die Projektbeteiligten zur Verfügung zu haben. Die Syntax dieser gemeinsamen Sprache wird durch die Infrastructure [Obj11c] und die Semantik durch die Superstructure [Obj11a] spezifiziert. Falls diese Sprachdefinition eindeutig wäre, wären unterschiedliche Interpretation der modellierten Inhalte nicht möglich. Der Simulator könnte die Semantik der UML verwenden, eine Veränderung der Abbildung wäre nicht erforderlich. Hier gibt es allerdings, selbst bei korrekter Anwendung der UML, zwei Probleme: Zum einen ist die Semantik der UML nicht widerspruchsfrei[115] und zum anderen lässt sich die UML erweitern. Zur Auflösung der Widersprüche und zur Festlegung der Semantik neuer Notationselemente ist eine Anpassung der Abbildung erforderlich. Darüber hinaus wird sie für die häufig nicht spezifikationskonforme Verwendung der UML in der praktischen Anwendung benötigt. Zwar sollte eine konforme Verwendung angestrebt werden, dies sollte aber nicht durch ein Werkzeug erzwungen werden.

Eine im Quellcode beschriebene Konvertierung der Modelldaten in simulationsinterne Daten erzeugt zwar weniger Implementierungsaufwand, der Architekt müsste aber bei jeder Anpassung der Abbildung den Quellco-

[114] Diese Ausage beruht auf der in Kapitel 4.3.3 getroffenen Entscheidung, die validierungsspezifische und entwicklungsspezifische Notation voneinander zu trennen. Die Wahl einer einheitlichen Notation kann sinnvoll sein, falls schon bei Projektbeginn der Einsatz des Validierungsprozesses geplant ist und eine geringe Anzahl validierungsspezifischer Daten zu erwarten ist. Details hier sind in Kapitel 8.2 beschrieben.

[115] Beispiele hierfür sind die Sequenzdiagramme [MW11], die Zustandsautomaten [LLS+12] und die Kompositionsstrukturdiagramme [OD11]. Teile der UML-Semantik werden deshalb als "loosely defined" [OD11, S. 418] beschrieben oder die UML wird als ein semi-formaler Modellierungsansatz [vL01] kategorisiert.

de bearbeiten. Aus diesem Grund wird die Festlegung der Abbildung mit Hilfe von XML-Dateien realisiert (entspricht den Schnittstellen *ConfigData-Mapping* und *DomainDataMaping* in Abbildung 7.12). Diese können vom Architekten mit wenig Aufwand und ohne detailliertes Wissen über den Simulator(-quellcode) verändert werden. Der Simulator liest sie zu Beginn der Simulation ein, konvertiert die Daten aus dem Modell anhand der eingelesenen Informationen in die simulationsinternen Daten und verwendet sie anschließend für die Konfiguration des zu simulierenden Systems.

7.2.3 Abbildung und Verwendung der Modelldaten in der Simulation

Die Transformation erzeugt aus dem generischen Simulator und den Daten aus einem UML-Modell einen domänenspezifischen Simulator (siehe Abbildung 7.11). Hierzu ist es zunächst notwendig die Abbildung der Semantik der Modellelemente auf die Elemente der Simulation festzulegen (siehe Kapitel 7.2.2). Die so von den Komponenten *Config Data Provider* und *Domain Data Provider* bereitgestellten Konfigurations- und Fachdaten werden von der Komponente *Simulation* zur Initialisierung des zu simulierenden Systems sowie zur Bereitstellung der Eingangsdaten für die fachlichen Algorithmen verwendet. Diese Nutzung der Modelldaten wird in den Kapiteln 7.2.3.2 und 7.2.3.3 anhand verschiedener Beispiele beschrieben. Dabei wird zum einen die Abbildung der Semantik und zum anderen die Verwendung der Daten für die Simulation beschrieben. Als Grundlage für die Beispiele dient die Simulation des Radar-Systems, deren Feindesign in Kapitel 7.2.3.1 ausschnittsweise vorgestellt wird.

7.2.3.1 Ausschnitt aus dem Feindesign der Radarsimulation

Abbildung 7.13 zeigt einen Ausschnitt aus dem Feindesign des Radarsimulators. Das vollständige Feindesign ist in [Poß12] beschrieben.

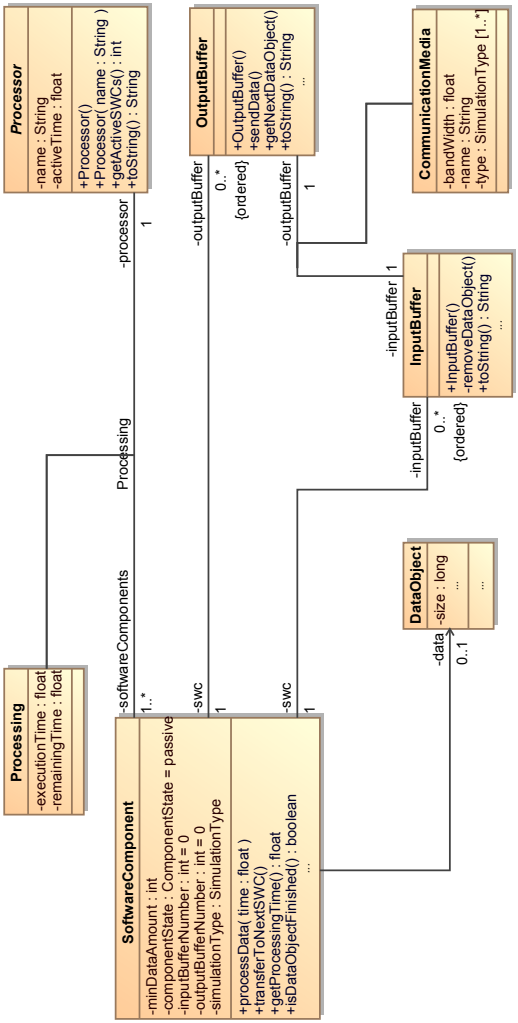


Abbildung 7.13: Ausschnitt aus dem Feindesign des Radarsimulators

Eine zentrale Rolle spielen die beiden Klassen *SoftwareComponent* und *Processor*. Sie stehen über die Assoziation *Processing* miteinander in Beziehung, so dass eine *SoftwareComponent*-Instanz auf genau einer *Processor*-Instanz ausgeführt wird und auf einer *Processor*-Instanz mehrere *SoftwareComponent*-Instanzen ausgeführt werden können. Für die Assoziation *Processing* existiert eine gleichnamige Assoziationsklasse, welche unter anderem die Ausführungszeit (Attribut *executionTime*) der *SoftwareComponent*-Instanz auf der verbundenen *Processor*-Instanz als Attribut enthält. *SoftwareComponent*-Instanzen können über logische Datenverbindungen mit anderen *SoftwareComponent*-Instanzen verbunden sein.

Eine logische Datenverbindung wird durch einen *InputBuffer* und einen *OutputBuffer* im Radarsimulator realisiert. Die Assoziation zwischen einer *InputBuffer*- und einer *OutputBuffer*-Instanz ergibt sich durch die physikalische Verbindung zwischen den beiden *Processor*-Instanzen, auf denen die den Puffern zugeordneten *SoftwareComponent*-Instanzen ausgeführt werden. Die Assoziationsklasse *CommunicationMedia* enthält u. a. das Attribut *bandWidth*, mit dem die Bandbreite der physikalischen Verbindung angegeben werden kann. Zur Berechnung der Datenübertragungszeit zwischen den *SoftwareComponent*-Instanzen ist neben der Bandbreite des Kommunikationsmediums noch die zu übertragende Datenmenge erforderlich. Diese ergibt sich über Instanzen der Klasse *DataObject*, deren Größe über das Attribut *size* angegeben wird[116]. Die übrigen Attribute und Operationen der Klassen sind für die Beispiele nicht weiter von Bedeutung.

[116] Die Klasse *DataObject* ist in diesem Ausschnitt des Feindesigns nur der Klasse *SoftwareComponent* zugeordnet. Dies erleichtert die Übersichtlichkeit der grafischen Darstellung. Im Radarsimulator sind auch die beiden Klassen *InputBuffer* und *OutputBuffer* über eine abstrakte Superklasse *Buffer* mit *DataObject* verbunden. Auf diese Weise reicht der Simulator die zu verarbeitenden Daten von einer *OutputBuffer*-Instanz zu einer *SoftwareComponent*-Instanz und von eben dieser nach der Verarbeitung in die zugeordneten *InputBuffer*-Instanzen.

7.2.3.2 Nutzung der Konfigurationsdaten für die Radarsimulation

Das erste Beispiel erläutert zunächst grundlegend die Nutzung von Konfigurationsdaten aus dem Validierungsmodell für die Initialisierung von Software- und Hardware-Komponenten. Im zweiten Beispiel werden weitere Architekturdetails in der Radardsimulation initialisiert. Dazu gehören die Zuordnung von Software- zu Hardware-Komponenten, die Kommunikationsmedien zwischen den Hardware-Komponenten und der Datenaustausch zwischen den Software-Komponenten. Außerdem wird die Verwendung redundanter Software-Komponenten vorgestellt, mit deren Hilfe der Architekt ausgewählte Verarbeitungsschritte parallelisieren kann.

Software- und Hardware-Komponenten Zunächst legt der Architekt die Abbildung der Semantik der VSN-Notationselemente auf die Feindesign-Klassen der Radarsimulation fest. Dies ist in Abbildung 7.14 dargestellt. Im oberen Bildbereich ist die VSN zu sehen, die der Architekt mit Hilfe eines UML-Profils modelliert hat. Die Notation enthält zwei Stereotypen, *hardware* und *software*, die zur Erweiterung der UML-Metaklasse *Component* eingesetzt werden können[117]. Als Semantik für das Notationselement UML-Komponente mit Stereotyp *software*, kurz Software-Komponente, legt der Architekt fest, dass die Simulation für jede Software-Komponente im Validierungsmodell eine Instanz der Klasse *SoftwareComponent* erstellt. Für jede UML-Komponente mit Stereotyp *hardware*, kurz Hardware-Komponente, soll die Simulation eine Instanz der Klasse *Processor* erstellen[118]. Diese Ab-

[117] Die Darstellung der *extend*-Beziehung zwischen den Stereotypen und der Metaklasse Component ist werkzeugspezifisch (EA) und nicht UML-konform.

[118] Mit dem Begriff Hardware-Komponente können viele physikalische Bausteinarten gemeint sein. Beispiele hierfür sind Speicher, Boards oder Schnittstellen. Programme können natürlich nur auf verarbeitenden Bausteinen wie beispielsweise einem Mikroprozessor oder einem FPGA *ausgeführt* werden. Der Einfachheit halber wird an dieser Stelle die Anzahl der Bausteinarten und damit der Stereotypen gering gehalten und angenommen, dass mit Hardware-Komponente ausschließlich verarbeitende Bausteine gemeint sind.

bildung wird in dem Bild durch die Pfeile zwischen den Elementen der VSN und den Feindesign-Klassen dargestellt.

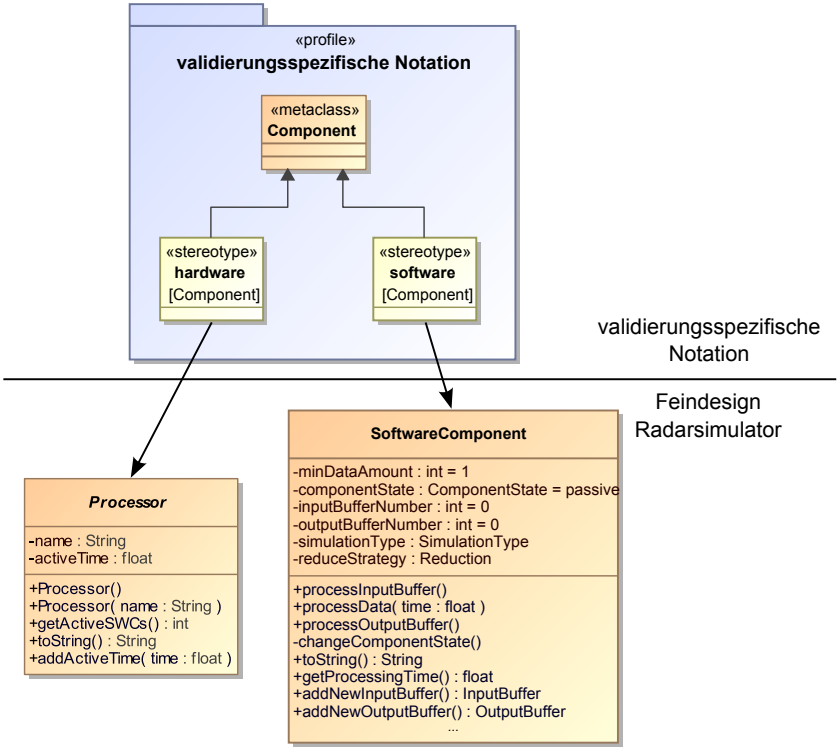


Abbildung 7.14: Abbildung der Semantik von VSN-Elementen auf die Feindesign-Klassen der Radarsimulation zur Festlegung von Software- und Hardware-Komponenten

Ein konkretes Beispiel hierzu ist in Abbildung 7.15 dargestellt. Im oberen Bildbereich sind vier Elemente aus dem Validierungsmodell abgebildet, zwei Software- und zwei Hardware-Komponenten. Entsprechend der oben festgelegten Semantik der Notationselemente legt der Radarsimulator dafür zwei

Instanzen der Klasse *SoftwareComponent* und zwei Instanzen der Klasse *Processor* an (unterer Bildbereich). Die Namen der Klasseninstanzen ergeben sich durch die Namen der dazugehörigen UML-Komponenten. Die Pfeile verdeutlichen, durch welches Element des Validierungsmodells die Klasseninstanzen erstellt wurden.

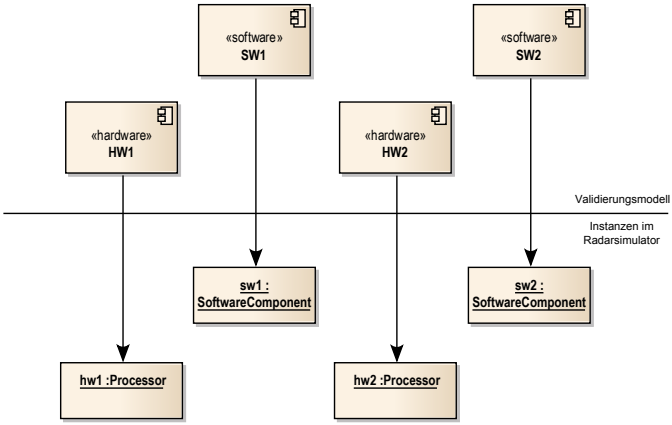


Abbildung 7.15: Beispiel zur Erzeugung von *Processor*- und *SoftwareComponent*-Instanzen des Radarsimulators durch Elemente des Validierungsmodells

Datenkommunikation und redundante Software-Komponenten In performenzkritischen Systemen werden Funktionalitäten, falls erforderlich, auf verschiedene Verarbeitungseinheiten aufgeteilt. Die Aufteilung kann zum einen vertikal erfolgen, wobei eine Funktionalität durch zwei neue substituiert wird; zum anderen ist eine horizontale Schneidung denkbar, bei der die Funktionalität redundant durchgeführt wird, beispielsweise um die Verarbeitung einer großen Datenmenge zu ermöglichen. Die erste Möglichkeit lässt sich durch das Modellieren der beiden Software-Komponenten und der Abbildung auf Instanzen der Klasse *SoftwareComponent* realisieren. Für die

redundante Durchführung der Funktionalität ist neben der Abbildung auf die *SoftwareComponent*-Instanzen noch ein Verwaltungsaufwand erforderlich, der die zu verarbeitenden Daten verteilt. Hier ergibt sich für die Simulation das Problem: Welche Software-Komponenten sind redundant und welche redundanten Komponenten gehören zueinander? Im nachfolgenden Beispiel wird dies über die Namen der Software-Komponenten beantwortet[119]. Ein Unterstrich gefolgt von einer Nummer weist auf eine horizontale Schneidung hin; beispielsweise *SW2_1* und *SW2_2*. Wird eine gleichverteilte Aufteilung der Daten auf die Komponenten in der Simulation verwendet (Standardfall im Radarsimulator), reichen diese Informationen bereits aus, um die Datenkommunikation zwischen den redundanten Software-Komponenten und deren Vorgänger zu simulieren[120].

Abbildung 7.16 zeigt die dafür erforderliche Abbildung von VSN-Elementen auf die Elemente des Feindesigns. Es handelt sich um eine erweiterte VSN des Beispiels aus dem letzten Abschnitt. Sie enthält neben den zwei bereits vorgestellten Stereotypen noch die Stereotypen *executedOn, dataExchange* und *communication*. Die ersten beiden dienen zur Charakterisierung einer Abhängigkeitsbeziehung (Erweiterung der UML-Metaklasse *Dependency*), der dritte erweitert die UML-Metaklasse *Connector*[121]. Die Abbildung zwischen den VSN-Elementen im oberen Bildbereich und den Feindesign-Elementen im unteren Bildbereich wird durch Pfeile dargestellt. Mit Hilfe der *executedOn*-Beziehung legt der Architekt fest, welche Software-Komponente auf welcher Hardware-Komponente ausgeführt wird.

[119] Eine weitere Möglichkeit ist der Einsatz von Instanzen der entsprechenden Komponente im Validierungsmodell.

[120] Für andere Verfahren zur Verteilung der Daten sind ggf. zusätzliche Daten erforderlich.

[121] Die Darstellung der *extend*-Beziehung zwischen den Stereotypen und der Metaklasse Component ist werkzeugspezifisch (EA) und nicht UML-konform.

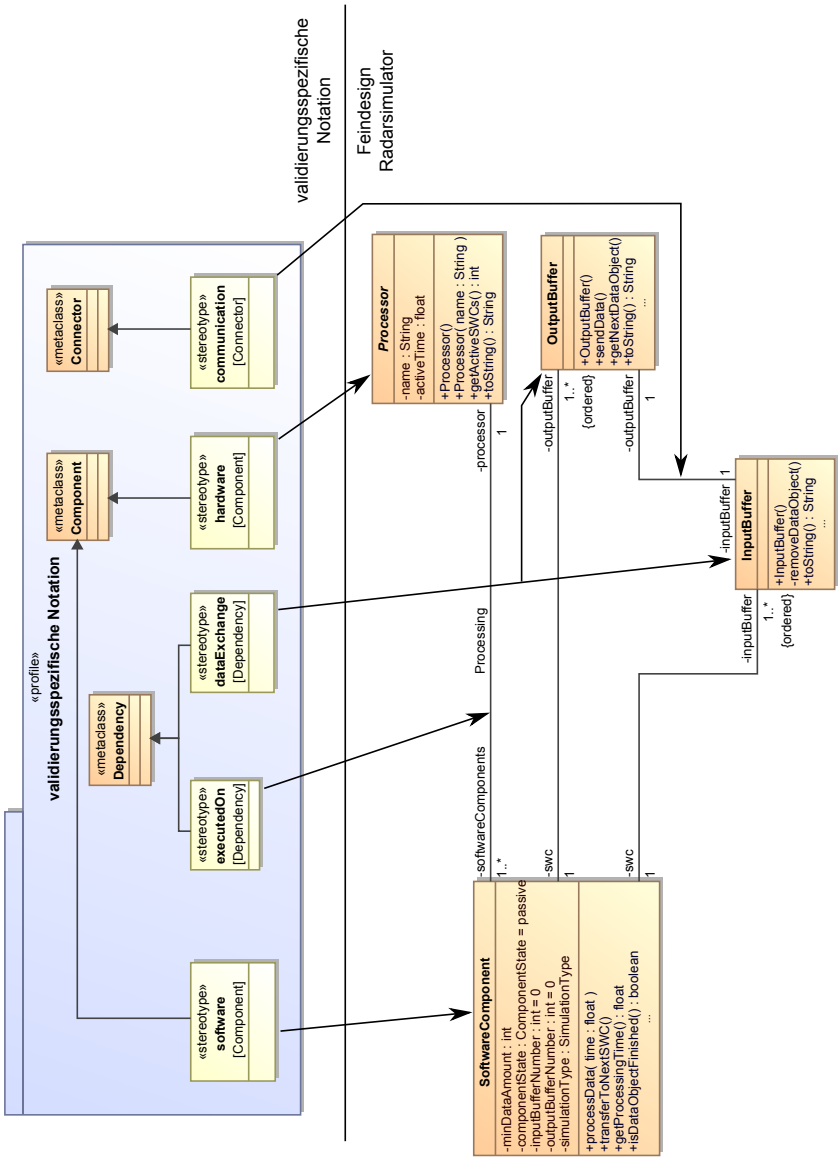


Abbildung 7.16: Erweitertes Beispiel zur Abbildung von Architekturinformationen auf die Simulationskonfiguration

Im Radarsimulator wird dies durch die Assoziation *Processing* abgebildet. Die *dataExchange*-Verbindung zwischen zwei Software-Komponenten *SW1* und *SW2* erzeugt einen *OutputBuffer* für *SW1* und einen *InputBuffer* für *SW2*. Die Verbindung dieser beiden Puffer erfolgt über den *communication*-Konnektor, der eine physikalische Kommunikationsleitung zwischen zwei *Hardware*-Komponenten repräsentiert. Werden zwei Software-Komponenten auf derselben Hardware-Komponente ausgeführt, erzeugt der Radarsimulator automatisch eine Verbindung zwischen den entsprechenden Puffern mit einer Bandbreite von ∞, so dass eine Datenübertragungszeit von 0ms für die Simulation verwendet wird.

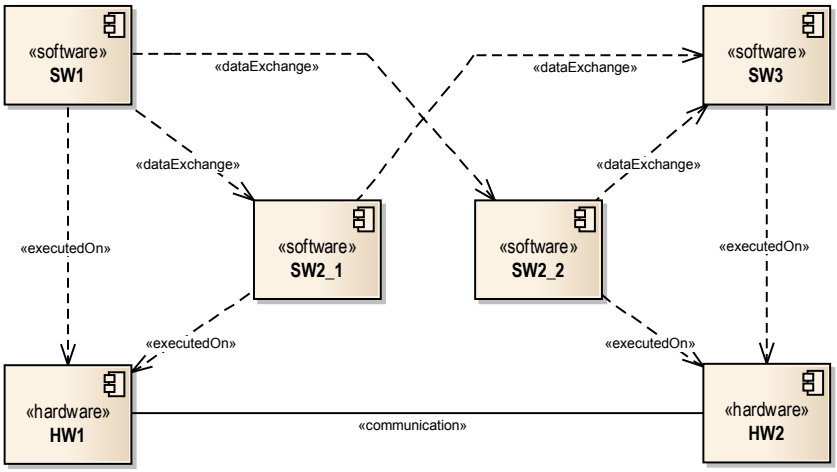


Abbildung 7.17: Validierungsmodell des Beispiels zur Konfiguration eines Validierungszielverfahrens mit redundanten Software-Komponenten

Ein konkretes Beispiel mit vier Software-Komponenten und zwei Hardware-Komponenten im Validierungsmodell ist in Abbildung 7.17 dargestellt[122].

[122] Aufgrund der Beispielgröße wird an dieser Stelle auf eine Verteilung der Architekturinformationen auf die Sichten Hardware, Software und Deployment [Kru95] verzichtet.

Die Software-Komponente *SW2* ist redundant ausgelegt. Die Software-Komponente *SW1* sendet Daten an *SW2_1* und *SW2_2*, welche wiederum beide Daten an *SW3* schicken. Während *SW1* und *SW2_1* auf der Hardware-Komponente *HW1* ausgeführt werden, ist *HW2* zur Abarbeitung von *SW2_2* und *SW3* verantwortlich. Die Datenkommunikation zwischen *SW2_1* und *SW2_2* erfolgt über die physikalische Verbindung zwischen *HW1* und *HW2*. Die durch diese im Validierungsmodell enthaltenen Informationen erzeugten Instanzen und Links des Radarsimulators zeigt Abbildung 7.18. Auf eine Verbindung der Elemente im Validierungsmodell und den Radarsimulatorelementen wie in Abbildung 7.15 wird aus Gründen der Übersichtlichkeit verzichtet.

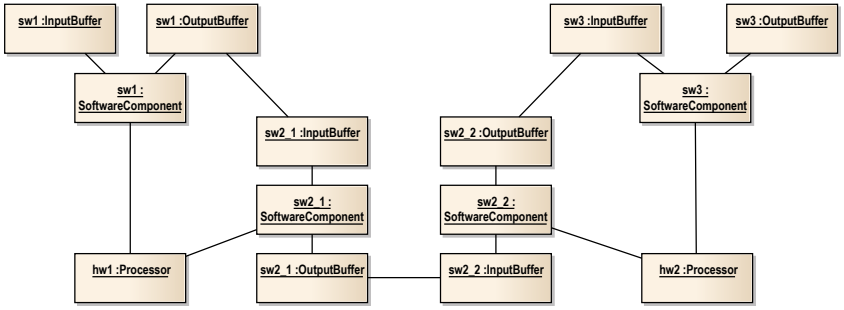


Abbildung 7.18: Radarsimulator-Instanzen des Beispiels zur Konfiguration eines Validierungszielverfahrens mit redundanten Software-Komponenten

7.2.3.3 Nutzung von Fachdaten für die Radarsimulation

Zur Simulation der Gesamtverarbeitungszeit, des Energieverbrauchs und der Betriebstemperatur werden neben den Berechnungsformeln auch deren Eingangsdaten benötigt. In dem in der Arbeit vorgestellten Ansatz werden sie aus dem Validierungsmodell ausgelesen, in das sie vom Architekten mit Hilfe der VSN hinzugefügt wurden. Bei den Fachdaten kann es

sich sowohl um quantitative Werte als auch um strukturelle Informationen handeln. Einige Konfigurationsdaten dienen deshalb nicht nur zur Architekturkonfiguration des simulierten Systems, sondern auch zur Durchführung der Simulation selbst. Beispielsweise wird zur Berechnung der Ausführungszeit einer Software-Komponente auf einer Verarbeitungseinheit die strukturelle Information benötigt, auf welcher Verarbeitungseinheit die Software-Komponente ausgeführt wird. Für die Berechnung der Gesamtverarbeitungszeit kommen zwei verschiedene Algorithmen zum Einsatz. Der Erste verwendet konkrete Zeitwerte als Ausführungszeiten für die Software-Komponenten. Der Zweite berücksichtigt die Datenkommunikation zwischen den Software-Komponenten und ermittelt die Ausführungszeiten anhand der benötigten MFlops der Software-Komponenten und der Leistung der verarbeitenden Hardware-Komponenten. Für die Berechnung der Gesamtverarbeitungszeit wird die Verarbeitung eines einzelnen Datenpakets durch die Software-Komponenten angenommen. Dadurch wird die Komplexität der Algorithmen reduziert und der Fokus der Beschreibung bleibt auf der Nutzung der Fachdaten.

Berechnung der Gesamtverarbeitungszeit über Ausführungszeiten Im ersten Beispiel wird die Gesamtverarbeitungszeit über die Summe der Verarbeitungszeiten der einzelnen Software-Komponenten entsprechend der Ausführungsreihenfolge ermittelt (siehe Formel (7.2)). Die Verarbeitungszeit der jeweiligen Software-Komponente auf der Verarbeitungseinheit legt der Architekt anhand empirisch ermittelter Werte fest. Die Ausführungsreihenfolge ergibt sich durch die *dataExchange*-Beziehungen zwischen den Software-Komponenten (siehe Kapitel 7.2.3.2).

$$\sum_{i=1}^{n} executionTime_{SW_i} \tag{7.2}$$

Der Radarsimulator erkennt die erste Komponente in der Verarbeitungskette daran, dass diese keine eingehende *dataExchange*-Beziehung besitzt, die letzte besitzt keine ausgehende. Die Komponenten dazwischen ergeben sich durch die Navigation über die *dataExchange*-Beziehung von der i-ten zur $i+1$-ten[123] Software-Komponente.

Abbildung 7.19 zeigt die Abbildung der VSN-Elemente auf die Feindesign-Elemente. Die genaue Zuordnung ist durch Pfeile vom oberen in den unteren Bildbereich kenntlich gemacht. Die Bildbereiche werden durch die horizontale, gestrichelte Linie voneinander getrennt. Der Stereotyp *software* wird auf die Klasse *SoftwareComponent* abgebildet[124]. Der TaggedValue *executionTime* wird auf das gleichnamige Attribut in der Assoziationsklasse *Processing* abgebildet. Die Verbindung der Software-Komponenten über *dataExchange*-Abhängigkeitsbeziehungen wird auf die Klassen *InputBuffer* und *OutputBuffer* transformiert. Die übrigen Attribute und Methoden im Simulatorfeindesign sind für dieses Beispiel nicht weiter von Bedeutung.

Ein konkretes Validierungsmodell für dieses Beispiel ist in Abbildung 7.20 dargestellt. Es enthält drei Software-Komponenten mit zwei *dataExchange*-Verbindungen und jeweils einem konkreten Wert für den TaggedValue *executionTime*. Die Gesamtverarbeitungszeit für ein Datenpaket ist demnach:

$$\sum_{i=1}^{3} executionTime_{SW_i} = 200ms + 600ms + 175ms = 975ms$$

[123] Da die erste und letzte Komponente ausgenommen sind, gilt $i \leq (n-1)$ und $i \geq 2$.

[124] Da für die Software-Komponente in diesem Beispiel keine verarbeitende Hardware-Komponente angegeben wird, erstellt der Radarsimulator für jede Software-Komponente eine eigene Instanz der Klasse *Processor*.

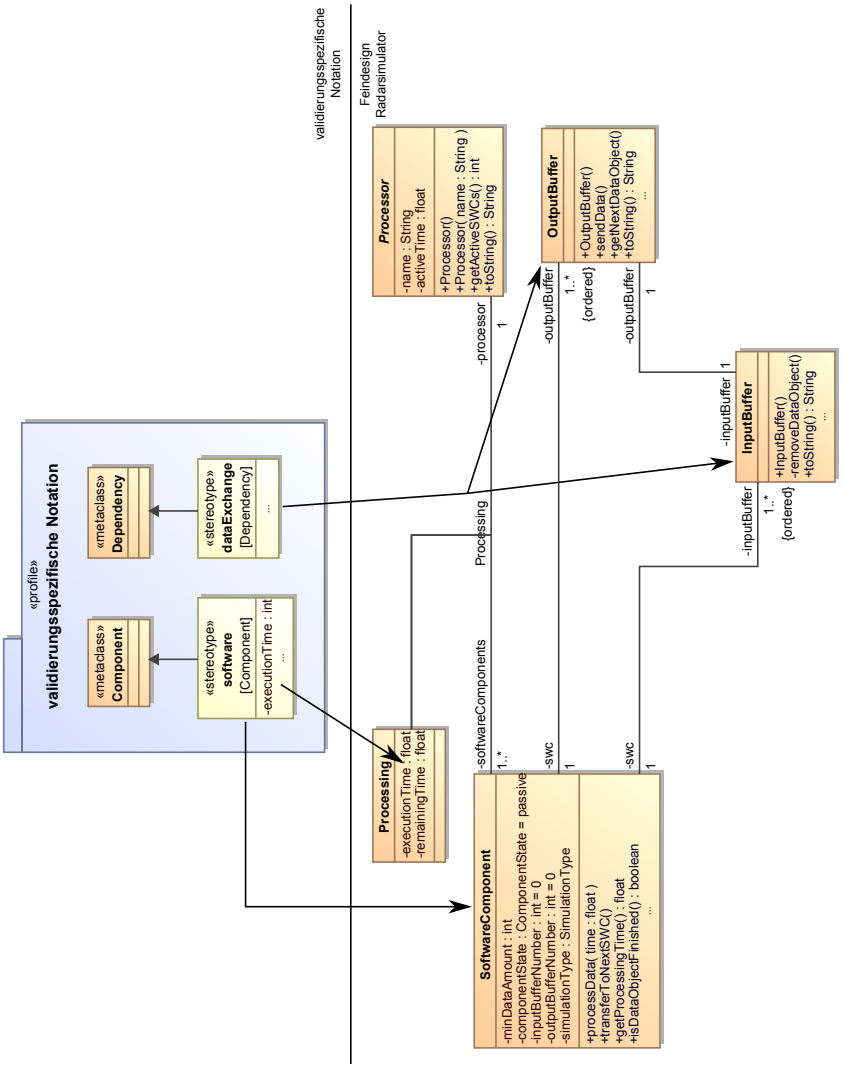


Abbildung 7.19: Abbildung der Semantik von VSN-Elementen auf die Feindesign-Elemente der Radarsimulation zur Berechnung der Gesamtverarbeitungszeit ohne Einflussfaktoren

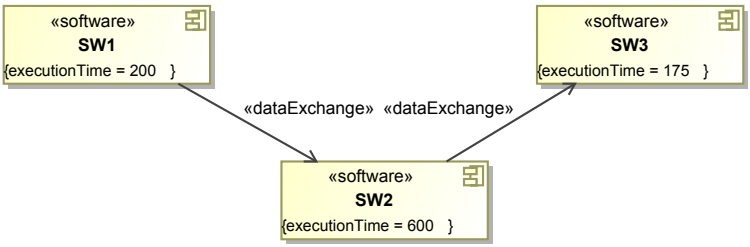


Abbildung 7.20: Software- und Hardware-Komponenten mit validierungsspezifischen Informationen zur Berechnung der Gesamtverarbeitungszeit ohne Einflussfaktoren

Berechnung der Gesamtverarbeitungszeit über MFlops und unter Berücksichtigung der Datenkommunikation Im zweiten Beispiel gibt der Architekt die Ausführungszeit nicht mehr anhand eines konkreten Zeitwertes an, sondern sie wird über die Komplexität der Software-Komponenten und die Leistung der verarbeitenden Hardware-Komponenten berechnet. Zusätzlich wird in diesem Beispiel auch die benötigte Zeit für die Datenkommunikation zwischen zwei Software-Komponenten bei der Berechnung der Gesamtverarbeitungszeit berücksichtigt.

Für die Berechnung der Gesamtverarbeitungszeit werden die erforderlichen MFlops einer Software-Komponente ($complexity_{SW_i}$) und die Leistung der verarbeitenden Hardware-Komponente in MFlops/sec ($performance_{HW_{SW_i}}$) benötigt.

$$\sum_{i=1}^{n} \frac{complexity_{SW_i}}{performance_{HW_{SW_i}}} + \sum_{j=1}^{n-1} \frac{data_{SW_j SW_{j+1}}}{bandWidth_{CD_{SW_j SW_{j+1}}}} \tag{7.3}$$

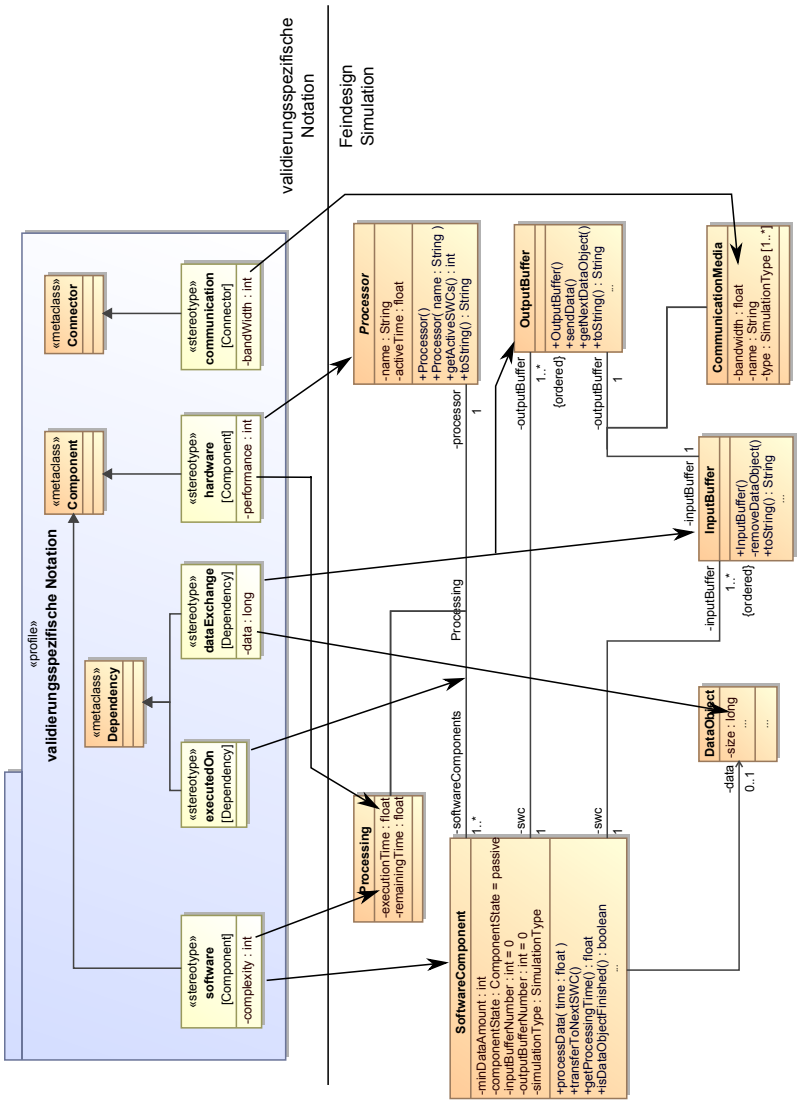


Abbildung 7.21: Abbildung der Semantik von VSN-Elementen auf die Feindesign-Elemente der Radarsimulation zur Berechnung der Gesamtverarbeitungszeit mit Berücksichtigung der Datenkommunikation

Die für die Datenübertragung erforderliche Zeit ergibt sich durch die zwischen zwei Software-Komponenten auszutauschenden Daten ($data_{SW_j SW_{j+1}}$) und die durch das entsprechende Kommunikationsmedium[125] bereitgestellte Bandbreite ($bandWidth_{CD_{SW_j SW_{j+1}}}$) (siehe Formel (7.3)).

Die Abbildung der VSN-Elemente auf die Elemente des Simulatorfeindesigns ist in Abbildung 7.21 zu sehen. Die Stereotypen *software* und *dataExchange* inkl. Abbildung auf die Elemente im Feindesign sind bereits aus dem ersten Beispiel bekannt. Hinzu kommen die Stereotypen *hardware*, *executedOn* und *communication* als Erweiterung der UML-Metaklassen *Component*, *Dependency* und *Connector*. Für Komponenten mit dem Stereotyp *hardware* erstellt der Radarsimulator eine Instanz der Klasse *Processor*. Abhängigkeitsbeziehungen mit Stereotyp *executedOn* zwischen Software-Komponenten und Hardware-Komponenten werden auf die Assoziation *Processing* transformiert.

Über den TaggedValue *performance* (Stereotyp *hardware*) kann der Architekt die Verarbeitungsleistung der Hardware-Komponente angeben. Daraus ermittelt der Radarsimulator zusammen mit dem Wert des TaggedValues *complexity* des Stereotypen *software* den Wert für das Attribut *executionTime* in der Assoziationsklasse *Processing*. Der TaggedValue *data* (Stereotyp *dataExchange*) wird auf das Attribut *size* der Klasse *DataObject* abgebildet und der TaggedValue *bandWidth* (Stereotyp *communication*) ist dem gleichnamigen Attribut in der Assoziationsklasse *CommunicationMedia* zugeordnet.

[125] Die Modellierung der physikalischen Kommunikationsleitungen erfolgt durch Konnektoren. Für die Datenübertragung zwischen zwei Software-Komponenten auf derselben Verarbeitungseinheit wird kein physikalisches Kommunikationsmedium benötigt. Für die Bandbreite wird hier der Wert ∞ angenommen, wodurch sich eine Übertragungszeit von 0ms ergibt.

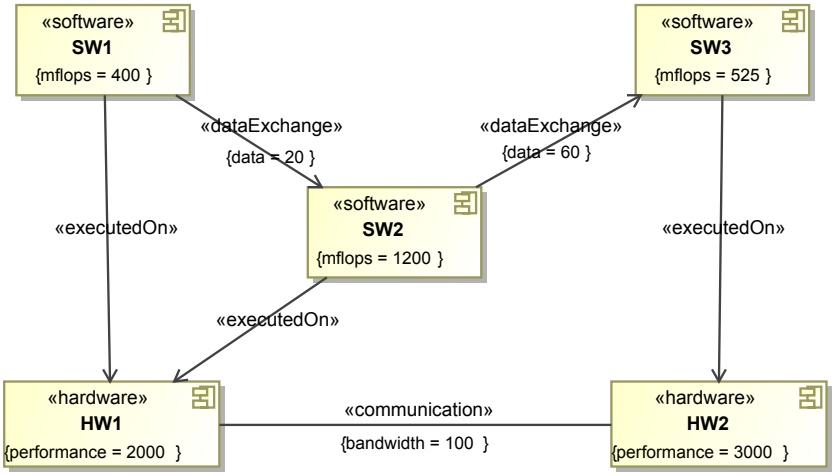


Abbildung 7.22: Software- und Hardware-Komponenten mit validierungsspezifischen Informationen für eine alternative Berechnung der Verarbeitungszeit mit Berücksichtigung der Kommunikationszeiten für die ausgetauschten Daten zwischen den Software-Komponenten

Die Elemente des Validierungsmodells für das zweite Beispiel sind in Abbildung 7.22 zu sehen. Das Beispiel enthält die drei Software-Komponenten aus dem ersten Beispiel sowie zwei Hardware-Komponenten. Die Software-Komponenten sind über die *executedOn*-Abhängigkeitsbeziehung genau einer verarbeitenden Hardware-Komponente zugeordnet. Die Hardware-Komponenten sind über einen *communication*-Konnektor miteinander verbunden, für den der Architekt über den TaggedValue *bandWidth* die Bandbreite 100 MiB/sec festgelegt hat. Den Software- und Hardware-Komponenten sind über die TaggedValues *complexity* und *performance* die Algorithmenkomplexität und die Verarbeitungsleistung zugeordnet. Für die Gesamtverarbeitungszeit ergibt sich demnach:

$$\sum_{i=1}^{3} \frac{complexity_{SW_i}}{performance_{HW_{SW_i}}} + \sum_{j=1}^{2} \frac{data_{SW_j SW_{j+1}}}{bandWidth_{CD_{SW_j SW_{j+1}}}}$$

$$\Rightarrow \frac{400MFlops}{2000\frac{MFlops}{sec}} + \frac{1200MFlops}{2000\frac{MFlops}{sec}} + \frac{525MFlops}{3000\frac{MFlops}{sec}} + \frac{20MiB}{100\frac{MiB}{sec}} + \frac{60MiB}{100\frac{MiB}{sec}}$$

$$= 0,2sec + 0,6sec + 0,175sec + 0,2sec + 0,6sec = 1,775sec$$

Durch das zweite Beispiel wird deutlich, dass zu den fachlichen Daten nicht nur quantitative Werte (MFlops, MFlops/sec, MiB und MiB/sec), sondern auch Informationen über die Architektur des Systems (Ausführungsort der Software-Komponenten, Kommunikationsleitungen) gehören. Die Konfigurationsdaten und die fachlichen Daten haben eine Schnittmenge. Abhängig vom gewählten Detaillierungsgrad müssen die Konfigurationsdaten vom Verfahren mehr oder weniger genau ausgewertet werden. Im ersten Beispiel, Abbildung 7.19, muss durch die Angabe der Verarbeitungszeit an den Software-Komponenten die Zuordnung der Software-Komponenten zu den Hardware-Komponenten durch die Simulation nicht berücksichtigt werden. Im zweiten Beispiel ist die Auswertung dieser Information allerdings erforderlich, da nur so die Verarbeitungszeit auf Basis der benötigten MFlops berechnet werden kann.

7.2.4 Protokollierung von Zwischenergebnissen

Das Erstellen der System-Architektur ist ein iterativer Prozess, bei dem durch Analysieren der Architektur und der Anforderungen sowie durch Simulieren des Systems und Vergleich mit den architekturrelevanten Anfor-

derungen eine valide Architektur entsteht. Die Simulation des Systems auf Basis einer bestimmten Architektur lässt sich durch vom Architekten festgelegte Arbeitsschritte, die Validierungszielverfahren, realisieren. Im Optimalfall lassen sich diese automatisieren. Anders sieht es bei der Analyse aus. Hier ist ein Mensch erforderlich, der die Ergebnisse der Simulation in Verbindung mit der Architektur untersucht, Schlussfolgerungen daraus zieht und Änderungen an der Architektur vornimmt. Als Unterstützung bei der Architekturanalyse können Simulationsdetails dienen, wodurch beispielsweise Flaschenhälse bei der Datenverarbeitung oder Kommunikation (siehe Auswertung der Datenpakete in den Eingangspuffern der Software-Komponenten in Kapitel 6.3, Seite 183) erkannt werden können.

Für eine detaillierte Analyse der Simulationsergebnisse können Protokolldaten ausgewertet werden, die den Status des Systems für einen bestimmten Zeitpunkt der simulierten Zeit[126] dokumentieren. Auf diese Weise lässt sich das Verhalten des Systems während der Simulation nachvollziehen. Die Größe der Protokolldaten ergibt sich durch den Detaillierungsgrad des erfassten Systemstatus und durch die Anzahl der für die Protokollierung verwendeten Zeitpunkte. Eine sehr grobe Betrachtung ergibt sich durch die Wahl des Anfangs- und Endzeitpunkts der Simulation. Genauer wird es durch die Festlegung auf ein statisches Zeitintervall, beispielsweise eine Nanosekunde. Hierbei ist mit einem hohen Datenaufkommen zu rechnen, insbesondere bei der Wahl eines kleinen Zeitintervalls. Die Auswertung die-

[126] An dieser Stelle wird zwischen der Simulationszeit und der simulierten Zeit unterschieden. Die Simulationszeit entspricht der Zeitdauer vom Start bis zum Ende der Simulation (siehe Kapitel 2.8). Bei der simulierten Zeit handelt es sich um die Zeitdauer, die für das zu simulierende System vom Simulationsstart bis zum Simulationsende vergeht. Bei einer Echtzeitsimulation entspricht die Simulationszeit genau der simulierten Zeit. Bei sehr detaillierten Simulationen, beispielsweise dem Verhalten von Molekülen, ist die simulierte Zeit kleiner als die Simulationszeit, bei einem geringeren Detaillierungsgrad vergeht die Zeit innerhalb der Simulation schneller als in der Realität.

ser Informationen kann für den Architekten sehr aufwändig sein. Filterwerkzeuge ermöglichen die Reduktion der Datenmenge durch Selektion bestimmter Datensätze bzw. durch die Verdichtung von Informationen (zur Erzeugung neuer Daten, z. B. prozentuale Anteile). Dabei können aber auch wichtige Informationen herausgefiltert werden. Da dem Architekten zu Beginn der Analyse nicht unbedingt bekannt ist welche Daten wichtig sind, ist zur Konfiguration der Filter eine gewisse Erfahrung von Vorteil. An dieser Stelle erleichtert der vorgestellte Radarsimulator die Datenanalyse, indem er neben der Festlegung statischer Intervallwerte auch dynamische Intervalle für die Datenprotokollierung ermöglicht.

Die dynamischen Zeitintervalle ergeben sich über die Datenverarbeitung in der Simulation selbst. Einfach ausgedrückt: Die Simulation protokolliert Simulationsdaten nur, wenn wirklich etwas passiert. Natürlich verändert sich während einer Simulation ständig irgendetwas, allerdings haben einige Änderungen für den Architekten eine geringere und andere eine höhere Relevanz. Bei der Berechnung der Gesamtverarbeitungszeit unter Berücksichtigung der Datenkommunikation zwischen den Software-Komponenten (vergleiche zweites Beispiel in Kapitel 7.2.3.3, Abbildung 7.22) sind die Zeitpunkte von Interesse, in denen ein Datenpaket die verarbeitende Software-Komponente wechselt bzw. in denen ein Datenpaket vollständig übertragen wurde. Alle Zeitpunkte dazwischen zeigen dem Architekten nur den Verarbeitungs- bzw. Übertragungsfortschritt des Datenpakets an. Wird für diese Zeitpunkte die simulierte Zeit protokolliert, können beispielsweise Software-Komponenten mit einer langen Verarbeitungszeit identifiziert werden. Ist der Engpass lokalisiert[127], hat der Architekt die

[127] Die Lokalisierung eines Engpasses ist in der Praxis meist aufwändig, da in den Simulationsergebnissen nur die Auswirkungen von Engpässen zu erkennen sind. Der Ort des Auftretens von Symptomen und des eigentlichen Flaschenhalses stimmen häufig nicht überein.

Möglichkeit, die Gesamtzeit beispielsweise durch den Einsatz redundanter Software-Komponenten (siehe Kapitel 7.2.3.2) zu verbessern.

Die Auswahl der zu protokollierenden Zeitpunkte ist abhängig vom Detaillierungsgrad der Simulation. Mit steigender Genauigkeit wächst der zu protokollierende Informationsgehalt zur Erfassung des Systemstatus und gegebenenfalls die Anzahl der relevanten Zeitpunkte. Soll der Speicherverbrauch einer Software-Komponente bei der Simulation berücksichtigt werden, ist für eine Worst-Case-Abschätzung der Anfangs- und Endzeitpunkt der Verarbeitung eines Datenpakets durch eine Software-Komponente ausreichend. In diesem Fall sind zwar keine neuen Zeitpunkte hinzugekommen, allerdings muss neben der simulierten Zeit auch der Speicherbedarf erfasst werden. Wird anstelle der Worst-Case-Abschätzung ein Durchschnittswert benötigt, kommen neue Zeitpunkte hin, die zwischen dem Verarbeitungsstart und dem Verarbeitungsende des Datenpakets durch eine Software-Komponente liegen.

Die Verwendung dynamischer Zeitintervalle erleichtert dem Architekten die Analyse der Simulationsergebnisse. Die Menge der Protokolldaten ist auf die wesentlichen Werte reduziert, was eine zielgerichtete Analyse ermöglicht. Als Nachteil gegenüber den festen Zeitintervallen ist zu nennen, dass einige Details verborgen bleiben können, die vom Architekten bzw. bei der Implementierung des Validierungszielverfahrens bei der Auswahl der Zeitpunkte nicht beachtet wurden. Dies kann allerdings auch bei einer zu groben Wahl des statischen Zeitintervalls auftreten.

Zur Lokalisierung der Engpässe sind Erfahrung und detaillierte Simulationsdaten erforderlich.

7.3 Ausblick

Sowohl für das vorgestellten Werkzeug MEASURED als auch für das Konzept des generischen Simulators sind weitere Funktionalitäten denkbar. Im Folgenden werden einige Ideen vorgestellt, mit denen der Architekt bei der Durchführung des Validierungsprozesses noch besser unterstützt werden könnte.

7.3.1 Erkennen von Architekturänderungen

Veränderungen an der Architektur sind im Validierungsprozess meistens mit Veränderungen am Validierungsmodell verbunden. Wird der Prozess von einer Person durchgeführt, kann davon ausgegangen werden, dass diese über alle Änderungen am Modell informiert ist. Wird diese Rolle aber von mehreren Personen eingenommen und sind diese zudem alle an der Architekturvalidierung beteiligt, ist es hilfreich, die validierungsrelevanten inhaltlichen Änderungen[128] zu identifizieren, so dass der Architekt weiß, dass die Validierungsergebnisse nicht auf dem aktuellen Modell basieren. Die Möglichkeit, Modelländerungen zu erkennen würde auch eine automatische Durchführung der Validierungszielverfahren[129] ermöglichen. Das Werkzeug könnte den Modellstatus in einem periodischen Zeitraum prüfen und bei Unterschieden eine vollständige Architekturvalidierung starten. Über die Ergebnisse könnte das Werkzeug den Architekt beispielsweise mit Hilfe einer Email informieren.

[128] Veränderungen an Diagrammen, beispielsweise das Verschieben von Modellelementen oder das Ändern des Aussehens, ändern zwar das Modell, aber nicht den validierungsrelevanten Inhalt.

[129] Es werden an dieser Stelle nur die automatisiert durchführbaren Verfahren betrachtet.

Falls sich die modellierten Daten in das EMF-UML-Format exportieren lassen, ist eine praktische Umsetzung des Modellvergleichs mit Hilfe von EMF-Compare [Ecl13] möglich. Diese Lösung wäre unabhängig von einem bestimmten Modellierungswerkzeug, weil der Export der Modelldaten in das EMF-Format von vielen Werkzeugen unterstützt wird. Für andere Werkzeuge, wie z. B. dem EA, müsste ein Adapter erstellt werden. Dass das Auslesen der Modelldaten aus einem solchen Werkzeug möglich ist, zeigen die in Kapitel 7.1.4 und 7.2 realisierten Prototypen für die Modelltransformation und für ein UML-basiertes Validierungszielverfahren.

7.3.2 Katalog für wiederverwendbare Elemente

Elemente wie die VSN, z. B. in Form UML-Profilen, oder Validierungszielverfahren lassen sich projektübergreifend einsetzen. Dazu muss der Architekt sie losgelöst von projektspezifischen Inhalte dokumentieren. Diese Schritte sind im detaillierten Prozessablauf in Abbildung 8.1 auf Seite 266 nach der Festlegung, dass die Anforderungen und die System-Architektur sich nicht mehr ändern, vorgesehen. Ein Unterstützungswerkzeug könnte dem Architekten eine Möglichkeit zur Verwaltung dieser Daten anbieten, so dass diese dem Architekten im nächsten Projekt zur Verfügung stehen. Auf diese Weise lassen sich Projekterfahrungen auf neue Projekte übertragen. Grundsätzlich wäre dies auch für die verwendeten Validierungsziele und Prüfkriterien denkbar, wobei diese ggf. auch durch das verwendete Qualitätsmodell dokumentiert sind. Die Verbindung von Validierungszielen, Prüfkriterien, Basisprofilen und Validierungszielverfahren würde dem Werkzeug erlauben, dem Architekten eine Art Portfolio oder Werkzeugkasten für die Architekturvalidierung anzubieten. Der Architekt kann in der Liste gespeicherter Validierungsziele nach einem geeigneten suchen und bekommt vom Werkzeug mögliche Prüfkriterien, Validierungszielverfahren

und Basisprofile für die Erstellung der VSN angezeigt. Dies würde die Benutzbarkeit des Validierungsprozesses noch einmal deutlich verbessern und den Architekten bei den ersten Prozessschritte (Validierungsziele identifzieren, Validierungszielverfahren festlegen und VSN erstellen) unterstützen.

7.3.3 Verwaltung von Validierungsergebnissen

Das Werkzeug erlaubt es dem Architekten, die Validierungsergebnisse aller Ziele zu laden und zu persistieren (siehe Abbildung 7.23). Auf diese Weise können verschiedene Architekturen validiert und mit den abgespeicherten Ergebnissen verglichen werden.

Damit die gespeicherten Ergebnisse reproduzierbar sind bzw. der Architekt zu einer bestimmten Architekturversion wechseln kann, müssen neben den reinen Validierungsergebnissen noch weitere Daten persistiert werden. Dabei handelt es sich um das Entwicklungs- und Validierungsmodell, die VSNs sowie die Validierungszielverfahren.

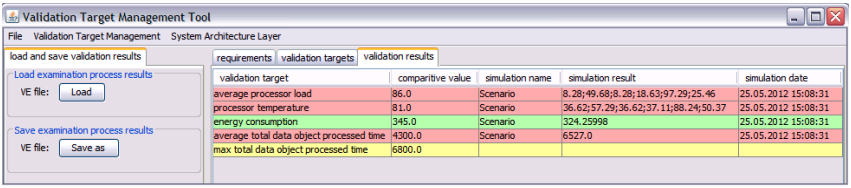


Abbildung 7.23: Maske zum Laden und Speichern von Validierungsergebnissen

7.3.4 Konsistenzprüfung der modellierten Elemente

Eine Voraussetzung für die Auswertung von Modelldaten ist eine semantisch korrekte Modellierung. Da es sich bei der UML um eine semi-formale Modellierungssprache handelt, deren Semantik nicht eindeutig definiert ist, sind hierfür Modellierungsrichtlinien notwendig. Die Einhaltung dieser Regeln muss vor der Verwendung der Daten überprüft werden, da die Ergebnisse der Verfahren, die diese Daten nutzen, ansonsten nicht verwendet werden können. Im Werkzeug könnte dies so aussehen, dass vor der Architekturvalidierung eine Konsistenzprüfung stattfindet und nur nach einem positiven Ergebnis die Validierung durchgeführt wird.

Die Überprüfung von Modellinhalten ist beispielsweise mit Hilfe der UML-eigenen Object Constraint Language (OCL)[130] möglich. Dabei dienen die OCL-Beschreibungen als Prüfregeln für den Konsistenzcheck. Einige Modellierungswerkzeuge wie beispielsweise der EA bieten eine darauf basierende oder ähnliche Überprüfungsfunktionalität an. Die Verwendung von OCL setzt praktischen Erfahrungen nach allerdings einen höheren Einarbeitungsaufwand voraus und ist keineswegs intuitiv. Eine werkzeugunabhängige Lösung ist mit Hilfe des Eclipse Modeling Project (EMP) möglich. Hier existiert beispielsweise im Unterpaket Xpand die Sprache Check[131], mit der sich im Vergleich zur OCL relativ einfach Prüfungsregeln mit einer C-Sprachen ähnlichen Programmierweise definieren lassen.

[130] http://www.omg.org/spec/OCL/
[131] http://www.eclipse.org/modeling/m2t/?project=xpand

7.3.5 Automatische Modellierung von validierungsspezifischen Daten

Bei der Erstellung oder Aktualisierung des Validierungsmodells kann das Werkzeug den Architekten zusätzlich zur Übernahme validierungsrelevanter Daten aus dem Entwicklungsmodell auch bei der Modellierung validierungsspezifischer Daten unterstützen. Einige Unternehmen besitzen für die Modellierung ihrer Systeme einen modellierten Hardware-Katalog, so dass sie die modellierten Bauteile projektübergreifend verwenden können[132]. Neben den Modellelementen könnten die verschiedenen Produkteigenschaften (Energieverbrauch der Bauteile, Zugriffszeiten bei Speicherelementen, Preise usw.) entweder im Modell oder beispielsweise in einer Datenbank abgelegt werden. Ordnet der Architekt nun den TaggedValues in den UML-Profilen eine entsprechende Produkteigenschaft zu, kann das Werkzeug bei der Transformation diese validierungsspezifischen Daten direkt dem Validierungsmodell hinzufügen. Dadurch wird der Modellierungsaufwand für den Architekten zur Ergänzung der validierungsspezifischen Daten reduziert. Erfolgt die Zuordnung TaggedValue - Produkteigenschaft in Verbindung mit den Basisprofilen, können die Zuordnungen wiederverwendet werden. Es kann dann zwischen projektspezifischen und projektübergreifenden Zuordnungen unterschieden werden.

7.3.6 Nutzung von Aktivitäten für den generischen Simulator

Der generische Simulator könnte zur Bestimmung der Datenflüsse zwischen den Software-Komponenten die Informationen aus Aktivitätsdiagrammen verwenden. Hierzu muss zunächst eine Verbindung zwischen den ver-

[132] Die Verwaltung solcher Katalogdaten ist auch Bestandteil des MADES-Projekts [MAD11].

haltensorientierten (Aktivitäten) und den strukturellen Daten (Komponenten) hergestellt werden. Dies lässt sich beispielsweise über eine Abhängigkeitsbeziehung mit dem Stereotyp *realize* erreichen. Diese Konstruktion erlaubt es dem Architekten nachzuvollziehen, welche Software-Komponente welche Aktivitäten beinhaltet bzw. durch welche Software-Komponente eine Aktivität realisiert wird[133]. Auf Basis dieser Zuordnung lässt sich die Ausführungsreihenfolge der Software-Komponenten für ausgewählte Use Cases[134] aus den Modelldaten ableiten. Ein simples Beispiel mit drei Aktivitäten und zwei Software-Komponenten ist im oberen Bereich der Abbildung 7.24 zu sehen. *Aktivität 1* und *Aktivität 2* werden durch die Software-Komponente *SW1* realisiert, *Aktivität 3* durch *SW2*. Über den Kontrollfluss der Aktivitäten lässt sich für die Architektur ableiten, dass eine Datenkommunikation zwischen *SW1* und *SW2* existiert[135]. In Kapitel 7.2.3.2 wurde diese Information über eine *dataExchange*-Abhängigkeitsbeziehung modelliert. Falls die Elemente der System-Analyse (z. B. Use Cases und Ablaufbeschreibungen durch Aktivitäten) und die Verbindung zu den Elementen des System-Designs (*realize*-Verbindung) im Entwicklungsmodell vorhanden sind, können diese in das Validierungsmodell übernommen werden. Damit sind alle Informationen zur Bestimmung der Abarbeitungsreihenfolge bereits vorhanden und der Architekt muss keine zusätzlichen *dataExchange*-Abhängigkeitsbeziehungen modellieren.

[133] Werden diese Verbindungen für alle Aktivitäten und Software-Komponenten durchgeführt, lässt sich überprüfen, ob alle Aktivitäten durch mindestens eine Software-Komponente realisiert werden. Gerade bei mittleren und großen Systemen ist ein solcher Konsistenzcheck eine gute Unterstützung für den Architekten.

[134] Ein Use Case besteht aus mehreren Schritten, die beispielsweise mit Hilfe von Aktivitäten modelliert werden können.

[135] Bei der Verarbeitung von Radarsignalen geben die Software-Komponenten die verarbeiteten Daten immer an die nächste(n) Komponente(n) weiter. Dadurch steckt hinter jedem Kontrollfluss auch immer ein Datenfluss. Natürlich kann es auch Kontrollflüsse zwischen zwei Aktivitäten geben, bei denen die Aktivitäten keine Daten austauschen.

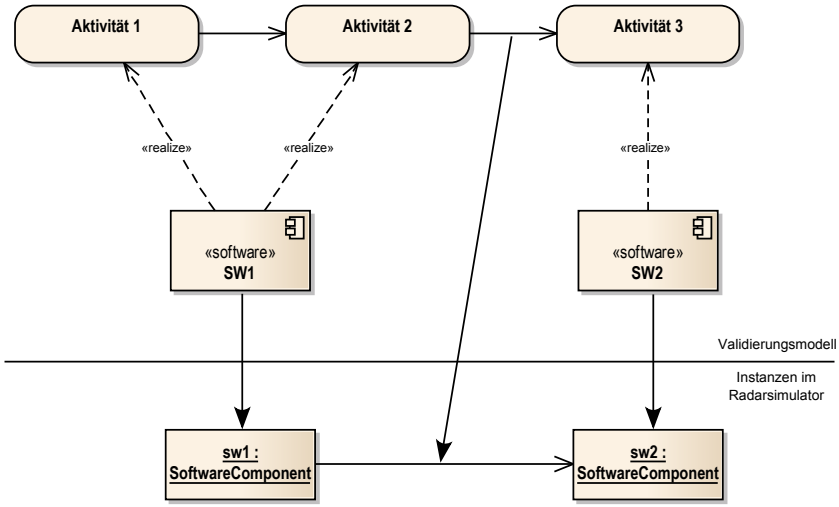


Abbildung 7.24: Abbildung der Verbindung zweier Software-Komponenten über die zugewiesenen Aktivitäten

In Kapitel 6 wird die Gesamtverarbeitungszeit über die Ausführungszeiten der Software-Komponenten berechnet. Diese wurden mit Hilfe von Tagged-Values an die Software-Komponenten modelliert. Eine Alternative hierfür ist die Angabe der Ausführungszeiten auf Aktivitätsebene. Die Verarbeitungszeit der Software-Komponente ergibt sich dann durch die Verarbeitungszeiten der ausgeführten Aktivitäten. Zwar ist der erforderliche Simulationsaufwand höher, die Berücksichtigung des genauen Verarbeitungsablaufs innerhalb einer Software-Komponente, ggf. noch abhängig von den zu verarbeiteten Daten oder anderen Konfigurationsdaten, könnte aber auch eine genauere Berechnung der Gesamtverarbeitungszeit ermöglichen[136].

[136] Natürlich in der Annahme, dass entsprechend detaillierte Daten verfügbar sind.

Durch den Einsatz weiterer Modellelemente lässt sich das beschriebene Beispiel ausbauen. So könnte über die Modellierung von Aktivitätsparametern, Ein- und Ausgangspins von Aktionen oder Objektflüssen modelliert werden, welche Daten zwischen den Aktivitäten bzw. Aktionen und damit zwischen den *SoftwareComponent*-Instanzen ausgetauscht werden. Alternativ ist die Verwendung eines Informationsflusses (engl. *information flow*) zwischen zwei Aktionen oder den Software-Komponenten sowie die Verwendung von Schnittstellen-Elementen an den Software-Komponenten denkbar. Die zu übertragenden Daten ergeben sich über die zugeordneten Informationseinheiten (engl. convey items). Die Modellierung weiterer Hardware-Komponenten wie Speicher oder Controller sowie deren Verbindung zu anderen physikalischen Komponenten ermöglichen eine detailliertere Hardwarebeschreibung, die zusammen mit den fachlichen Daten zu präziseren Simulationsergebnissen[137] führen können.

[137] Die Erhöhung des Detaillierungsgrades ist an dieser Stelle nicht mit qualitativ besseren Ergebnissen gleichzusetzen. Das Simulationsergebnis hängt von den Eingangsdaten und den verwendeten Algorithmen ab. Eingangsdaten und Algorithmen müssen zusammenpassen, damit ein qualitativ hochwertiges Ergebnis simuliert werden kann (siehe Kapitel 10.2).

8 Projektspezifische Anpassungen

Kein Projekt gleicht dem anderen, jedes Projekt hat seine spezifischen Rahmenbedingungen. Aus diesem Grund müssen das zur Entwicklung des Systems verwendete Vorgehen und die eingesetzten Methoden stets den projektspezifischen Bedingungen angepasst werden. Natürlich ist es hilfreich und sinnvoll, Erfahrungen und vorhandenes Wissen aus alten Projekten zu übernehmen, sofern es Vorteile für das neue Projekt bringt. Beim Validierungsprozess lassen sich die in Projekten verwendeten Validierungszielverfahren und validierungsspezifischen Notationen wiederverwenden. Kapitel 8.1 geht hierauf in einer detaillierteren Beschreibung des Validierungsprozesses ein. Darüber hinaus wird der Prozessablauf insgesamt etwas detaillierter als in Kapitel 4.3 beschrieben. Kapitel 8.2 beschreibt die verschiedenen Möglichkeiten der UML zur Modellierung der validierungsspezifische Notation (VSN). Dabei werden die Vor- und Nachteile der verschiedenen Möglichkeiten herausgearbeitet und bzgl. des benötigten Aufwands, der Qualität und Wiederverwendbarkeit in verschiedenen Domänen bewertet. Diese tabellerisch zusammengefasste Bewertung dient dem Architekten damit als Unterstützung bei der Wahl einer für sein Projekt geeigneten Modellierungsart für die VSN. In der vorliegenden Arbeit wurde die Verwendung des UML-Profilmechanismus bevorzugt. Kapitel 8.3 geht auf den Dokumentationsort der validierungsspezifischen Daten ein. Hier ist sowohl die Dokumentation zusammen mit den entwicklungsspezifischen Daten in einem UML-Modell als auch eine separate Dokumentation in einem zweiten Modell denkbar. In der vorliegenden Arbeit wurde sich für die Zwei-Modell-Variante entschieden.

8.1 Detaillierter Prozessablauf

Erfahrung ist ein nicht zu unterschätzender Faktor bei der Erstellung und Validierung von Architekturen. Das Wissen aus anderen Projekten kann nicht nur zu einer qualitativ höherwertigeren Architektur führen, sondern auch den Arbeitsaufwand im aktuellen Projekt verringern. Aus diesem Grund bietet der Validierungsprozess die Möglichkeit an, validierungsspezifische Notationen und Validierungszielverfahren wiederzuverwenden (siehe auch [PGQ13a]). Hierzu sind einige weitere Prozessschritte erforderlich, die im Aktivitätsdiagramm in der Abbildung 8.1 zu sehen sind. Im Vergleich zur Ablaufbeschreibung in Abbildung 4.3 werden einige Prozessschritte noch etwas expliziert. Auf diese Weise sind einige Details der einzelnen Prozessschrittbeschreibungen in Kapitel 4.3 auch in der Ablaufbeschreibung des Validierungsprozesses vertreten.

Der Prozess beginnt wie in Abbildung 4.3 mit der Aktivität *Validierungsziele identifizieren*. Darauf folgt die Verbindung der identifzierten Validierungsziele mit den architekturrelevanten Anforderungen. Um wirklich mit allen relevanten Validierungszielen weiter zu arbeiten, erfolgt eine Vollständigkeits- und Relevanzprüfung. Hierfür ist beispielsweise ein Review[138] unter Beteiligung der Projektleitung und der verantwortlichen Entwickler denkbar, in dem fehlende Ziele ergänzt werden und gemeinsam festgelegt wird, welche Ziele für den aktuellen Projektstand zur Architekturvalidierung beitragen können, sollen und müssen. Eine Methode für die Auswahl wäre die Vergabe von Prioritäten. Fällt die Prüfung negativ aus, müssen die Schritte *Validierungsziele identifizieren* und *Validierungsziele mit architekturrelevanten Anforderungen verbinden* wiederholt und eine erneute Prüfung

[138] Weitere Informationen zu Reviews sind in [Rup09b] und [Wal11] beschrieben.

durchgeführt werden[139]. Im positiven Fall kann der Architekt mit der Festlegung der Validierungszielverfahren fortfahren.

Bei der Verfahrenswahl hat der Architekt die Möglichkeit, auf Verfahren aus anderen Projekten zuzugreifen. Dies ist im Aktivitätsdiagramm durch das *datastore*-Element *Validierungszielverfahren* dargestellt. Analog stehen dem Architekten bei der Erstellung der VSN *Basisprofile* aus früheren Projekten zur Verfügung (siehe gleichnamiges *datastore*-Element). Der Validierungsprozess sieht vor, dass der Architekt zur Förderung der Wiederverwendung und zur Wissensübertragung auf andere Projekte am Ende der Design-Phase, d. h. nach dem Freeze der Anforderungen bzw. der Architektur (siehe gleichnamiges Event)[140], Basisprofile und Validierungszielverfahren in einer projektunabhängigen Form dokumentiert. Für die Basisprofile bietet sich die Erstellung eines separaten UML-Profils an, das bei der Verwendung im nächsten Projekt eingebunden werden kann. Beispiele für Basisprofile sind im Kapitel *Erweitertes Radar-System-Beispiel* auf Seite 173 zu finden.

[139] Wie genau diese Wiederholung aussieht, sollte pragmatisch angegangen werden. Kleine Änderungen können z. B. innerhalb des Reviews durchgeführt werden. Falls die Änderungen überschaubar sind, sollte es ausreichend sein, die ersten zwei Schritte zu wiederholen und ohne eine erneute Prüfung den nächsten Prozessschritt zu beginnen. Eine erneute Prüfung sollte nur bei größeren Problemen, beispielsweise unterschiedlichem Verständnis der Anforderungen, durchgeführt werden. Es ist zu bedenken, dass im Laufe der Systementwicklung die Architekturvalidierung mehrmals durchgeführt wird.

[140] Die zeitliche Einordnung muss hier nicht zu genau genommen werden. Der Architekt kann natürlich auch schon während des Validierungsprozesses Basisprofile und Validierungszielverfahren extrahieren, insbesondere beim Wechsel eines Verfahrens, was sich meist auch auf die VSN auswirkt. Hier sollten natürlich Aufwand und Nutzen miteinander verglichen werden.

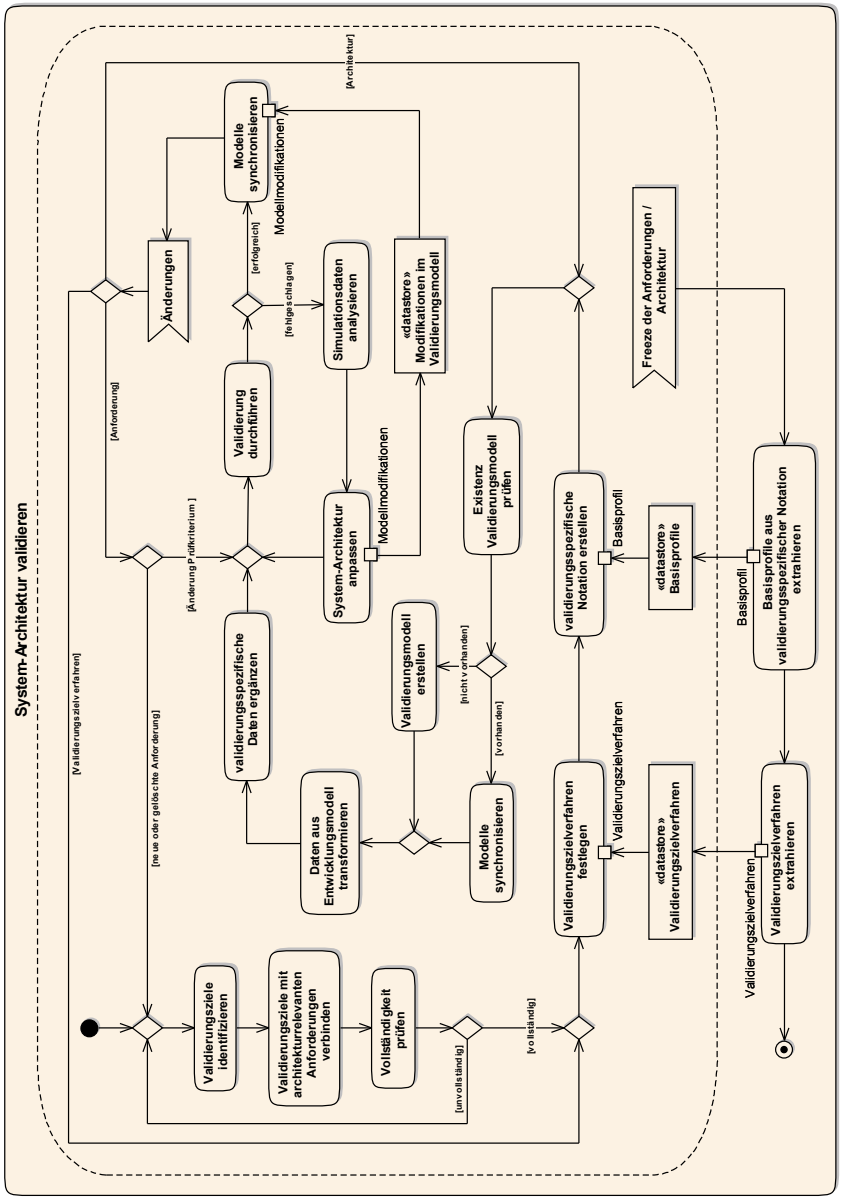


Abbildung 8.1: Detaillierter Ablauf des Validierungsprozesses

Auf Basis der VSN erstellt der Architekt das Validierungsmodell. Falls noch keins existiert, erstellt er ein neues Modell und transformiert alle validierungsrelevanten Elemente (Struktur und Verhalten) aus dem Entwicklungsmodell in das neue Validierungsmodell. Ist bereits ein Validierungsmodell vorhanden, synchronisiert er die beiden Modelle, um mögliche Änderungen an Elementen, die sowohl im Entwicklungsmodell als auch im Validierungsmodell vorhanden sind, zu übernehmen. Falls aufgrund neuer Validierungsziele bzw. neuer oder veränderter Verfahren weitere Elemente aus dem Entwicklungsmodell benötigt werden, transformiert er diese ebenfalls. Im Anschluss ergänzt er alle noch nicht vorhandenen validierungsspezifische Daten.

Nach einer fehlgeschlagenen Validierung der Architektur hat der Architekt die Möglichkeit die Simulationsdaten im Detail zu analysieren. Dies ist ein Vorteil, der sich durch die Verwendung der dynamischen Prüftechnik Simulation gegenüber den formalen[141] ergibt: Neben dem Validierungsergebnis stehen dem Architekten auch Zwischenergebnisse der Simulationen zur Verfügung[142]. Formale Verfahren stellen nur Daten für ein Gegenbeispiel bereit. Ist das Problem identifiziert, passt der Architekt die Architektur im Validierungsmodell an und führt die Validierung auf Basis dieser aktualisierten Daten erneut durch. Falls das Ergebnis positiv ist, werden die entwicklungsrelevanten Daten aus dem Validierungs- ins Entwicklungsmodell übertragen. Die Abläufe nach den verschiedenen Änderungsmöglichkeiten sind ausführlich in Kapitel 6.3 beschrieben.

[141] Die Definition der Begriffe ist in Kapitel 2.8 zu finden. Die Vor- und Nachteile formaler Prüftechniken sind am Verfahren Model-Checking in Kapitel 5.1.2 beschrieben.

[142] Für eine detaillierte Beschreibung siehe Kapitel 7.2, Seite 227.

8.2 Modellierungsalternativen für die validierungsspezifische Notation

Der Validierungsprozess ist ein Hilfsmittel des Architekten innerhalb der Systementwicklung und sollte so wenig Mehraufwand wie möglich erzeugen. Wie die validierungsspezifischen Daten modelliert werden, spielt dabei eine zentrale Rolle, woraus sich die folgenden Anforderungen an die Modellierungsart der VSN ergeben:

1. Die Modellierung der validierungsspezifischen Daten muss einen geringen Arbeitsaufwand für den Architekten erzeugen.
2. Die validierungsspezifischen Daten müssen mit geringem Aufwand den sich ändernden Bedingungen in der Systementwicklung angepasst werden können.
3. Die Modellierungsart muss konform zur UML-Spezifikation sein.

Zusätzlich hat der in der Arbeit vorgestellte Ansatz den Anspruch, für verschiedene Domänen eingesetzt werden zu können.

Anforderung eins und zwei beziehen sich auf den Arbeitsaufwand des Architekten zur initialen Erstellung und zur Anpassung der Notationen an Veränderungen. Die validierungsspezifische Notation wird durch die Eingangsdaten der Validierungszielverfahren und durch die im Entwicklungsmodell eingesetzte Notation zur Dokumentation des Systems beeinflusst. Die Eingangsdaten sind wiederum vom verwendeten Verfahren abhängig, die Notation des Entwicklungsmodells vom aktuellen Projekt. Der Projektfortschritt wirkt sich nicht nur auf die modellierten Inhalte aus, sondern auch auf die eingesetzte Notation zur Repräsentation der Inhalte im Modell. Die Nota-

tion des Entwicklungsmodells ist also keineswegs eine Konstante im Entwicklungsprozess. Da die Verfahren im Zuge des Projektfortschritts ebenfalls Veränderungen unterliegen, wird die validierungsspezifische Notation durch zwei sich ändernde Faktoren beeinflusst. Während der Aufwand zur initialen Erstellung der VSN einmalig ist, muss im Laufe der Systementwicklung mit häufigen Notationsanpassungen gerechnet werden.

Die dritte Anforderung dient der Benutzerfreundlichkeit und erleichtert die Transformation von Inhalten zwischen verschiedenen Modellen. Die UML hat sich in der Industrie als Quasi-Standard etabliert. Die Entwickler verwenden die UML nicht nur als Kommunikationsmittel, sondern auch immer mehr für eine modellbasierte Dokumentation. Das UML-Metamodell ermöglicht es zudem, die Inhalte der Dokumentation weiterzuverwenden, beispielsweise zur Konsistenzprüfung oder zur Generierung von Artefakten wie beispielsweise Quellcode oder textuelle Dokumentation. Zwar werden in den Unternehmen und in deren Projekten häufig unterschiedliche Notationen eingesetzt, ein grundlegendes Verständnis der UML ist aber vorhanden, so dass auch unbekannte Modelle schnell erarbeitet werden können[143]. Die UML-Konformität ist auch für die Auswahl des Modellierungswerkzeugs von Bedeutung. Wird eine konforme Notation eingesetzt, lässt sich diese mit den gängigen Werkzeugen erstellen. Auch die Transformation von Modell-Elementen zwischen zwei konformen Notationen ist mit vorhandenen Werkzeugen (siehe [SWG09] oder [RHW+10] und Kapitel 7.1.4) möglich.

Die OMG ermöglicht mit der Meta Object Facility (MOF) die Erstellung von Metamodellen, ein bekanntes Beispiel ist das Metamodell der UML. Ein auf

[143] Diese Aussage stützt sich auf praktische Erfahrungen.

der MOF basierendes Metamodell ist allerdings nicht zwingend konform zur UML-Spezifikation. Laut der UML-Infrastructure [Obj11c, S. 174] gibt es zwei Möglichkeiten, die UML zu erweitern. Dabei handelt es sich um die Metamodell-Erweiterung und um die Metamodell-Anpassung; letztere ist auch unter dem Namen Profilmechanismus bekannt.

Der erste Ansatz, die Metamodell-Erweiterung, wird in der Literatur häufig als "schwergewichtiger" [MKS00] Ansatz charkaterisiert. Die Größe des Metamodells wächst dabei; nicht benötigte Elemente oder Einschränkungen dürfen aber nicht entfernt werden, da ansonsten die UML-Konformität verletzt wird. Der zweite Ansatz, die Metamodell-Anpassung, fügt Informationen hinzu, wodurch die vorhandenen Elemente des Metamodells semantisch erweitert[144] werden. Das Metamodell selbst wird aber nicht verändert. Die Anpassung erfolgt mit sogenannten Profilen, welche aus *Stereotypen*, *TaggedValues* und *Constraints* bestehen (siehe [Obj11c, Kapitel 12.1] und [Obj11b, Kapitel 18]). Mit Hilfe der Stereotypes und TaggedValues lassen sich Informationen für festgelegte Elemente und Beziehungen hinzufügen. Die Constraints ermöglichen, die Verwendung von Elementen und Beziehungen des UML-Metamodells einzuschränken. Da diese aber immer noch Teil des Metamodells sind, bleibt die Konformität erhalten. Der Profilmechanismus wird vor allem für die Anpassung des UML-Metamodells auf eine bestimmte Plattform oder Domäne verwendet. Weitergehende Informationen sind in den UML-Spezifikationen [Obj11c] [Obj11b] und in [SVEH07] beschrieben.

Zur Erfüllung der geforderten Eigenschaften des Modellierungsmechanismus sind mehrere Lösungsmöglichkeiten denkbar:

[144] Die Erweiterung drückt sich durch eine Vererbungsbeziehung zwischen dem neuen und einem vorhandenen Element aus dem UML-Metamodell aus.

Gemeinsame Notation Der erste Ansatz betrachtet die Notation für die Entwicklung und Validierung als Ganzes. Bereits bei der Erstellung der Notation für die Dokumentation des Systems (Entwicklungsmodell) werden die validierungsspezifischen Aspekte berücksichtigt. Dadurch entsteht eine einheitliche Notation für die Entwicklung und Validierung[145]. Abbildung 8.2 stellt dies schematisch dar. Bei der Bewertung dieses Ansatzes spielt der verwendete Mechanismus eine untergeordnete Rolle. Der Vorteil liegt in der einheitlichen Notation, wodurch die Transformation von Elementen zwischen zwei Modellen[146] vereinfacht wird. Die einheitliche Notation schafft aber auch eine starke Abhängigkeit der entwicklungsspezifischen Daten von den validierungsspezifischen Daten. Da der Validierungsprozess ein Hilfsmittel in der Systementwicklung ist, sollte der Einfluss auf den normalen Entwicklungsprozess und dessen Dokumentation möglichst gering bis gar nicht vorhanden sein. Die hohe Kopplung erschwert zudem die Anwendbarkeit der Modellierungsart auf verschiedene Domänen. Neben der hohen Kopplung ist auch eine niedrige Kohäsion[147] zu beobachten. Die Notation bezieht sich nicht auf eine wohldefinierte Aufgabe, sondern auf die Entwicklungsdokumentation und die Architekturvalidierung.

[145] Dabei ist es nicht weiter von Bedeutung, ob für die beiden Aspekte ein gemeinsamens oder getrennte UML-Modelle verwendet werden.

[146] In diesem Fall verwenden beide Modelle dieselbe Notation.

[147] Die Begriffe Kopplung und Kohäsion stehen häufig im Zusammenhang mit Software- und System-Architekturen. Wie anhand der nachfolgend beschriebenen Lösungen zu sehen ist, wirkt deren Einhaltung sich auch positiv auf die *Architektur* oder Strukturierung der Notationselemente aus.

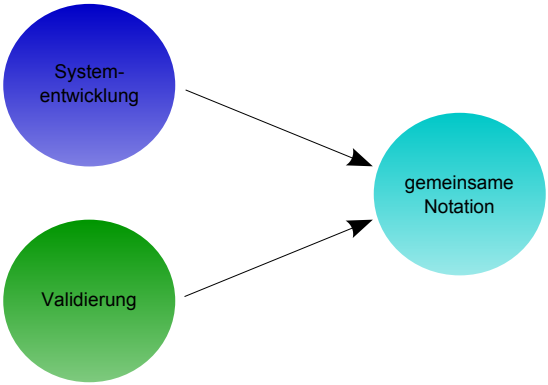


Abbildung 8.2: Schematische Darstellung einer gemeinsamen Notation

Strikte Trennung der Notationen Ein möglicher zweiter Ansatz ist die Trennung von Entwicklungs- und Validierungsnotation. Damit wird die Abhängigkeit der entwicklungsspezifischen und der validierungsspezifischen Notation aufgehoben und beide Notationen haben eine wohldefinierte Aufgabe. Wie die Kopplung zwischen der validierungsspezifischen und der entwicklungsspezifischen Notation aussieht, hängt vom gewählten Mechanismus ab. Die Erstellung eines eigenen Metamodells als Notation für das Validierungsmodell bietet einerseits eine unabhängige Wahl der Notationselemente, so dass die Teile der entwicklungsspezifischen Notation, die für die Architekturvalidierung irrelevant sind, nicht die validierungsspezifische Notation beeinflussen. Diese Trennung wird durch die schematische Darstellung in Abbildung 8.3 verdeutlicht. Die geringe Kopplung wirkt sich positiv auf die domänenunabhängige Modellierung der Eingangsdaten aus. Andererseits entsteht ein erhöhter Aufwand zur Erstellung der validierungsspezifischen Notation und zur Transformation vorhandener Informationen aus dem Ent-

wicklungsmodell in das Validierungsmodell[148]. Aufgrund der freien Wahl der Notationselemente kann es zudem zu unterschiedlichen Repräsentationen von Daten kommen, was die Lesbarkeit für den Architekten erschwert. Veränderungen an der Notation des Entwicklungsmodells können grundlegende Überarbeitung der validierungsspezifischen Notation erfordern, falls eine bewusst unterschiedlich gewählte Repräsentation eines Datums von den Änderungen betroffen ist oder durch die Änderungen notwendig wird. Die geringe Kopplung hat also einen erhöhten Arbeitsaufwand des Architekten zur Folge, der gerade bei häufigen Änderungen ansteigt.

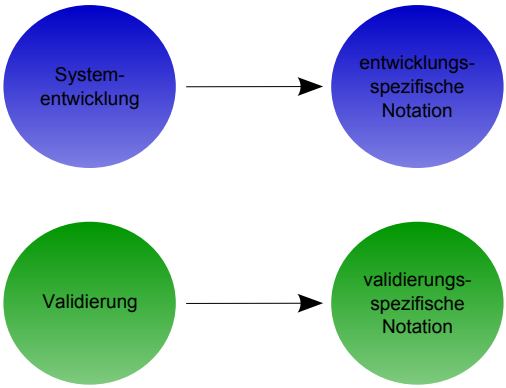


Abbildung 8.3: Schematische Darstellung von getrennten Notationen

Gemischte Notation Die Verwendung des Profilmechanismus reduziert den höheren Arbeitsaufwand für den Architekten bei Notationsänderungen auf Kosten einer höheren Kopplung der validierungsspezifischen gegenüber der entwicklungsspezifischen Notation. Wie in der schematischen Darstel-

[148] Für die Transformation von Elementen aus Modell A in Modell B ist eine Abbildung der Inhalte von Modell A auf die Inhalte von Modell B notwendig. Basieren die Modelle auf derselben Notation, entspricht die Transformation einer eins-zu-eins Abbildung. Unterscheiden sich jedoch die Notationen der Modelle, ist eine spezielle Abbildung, ein Adapter, zu erstellen.

lung in Abbildung 8.4 zu sehen ist, wird aus der entwicklungsspezifischen Notation der für die Validierung relevante Teil in die validierungsspezifische Notation übernommen. Für alle Informationen, die nicht mit der entwicklungsspezifischen Notation modelliert werden können, wird ein Profil mit Stereotypen und TaggedValues für die entsprechenden Elemente und Beziehungen angelegt. Mit diesem Ansatz muss der Architekt sich auf keine vollständig unterschiedliche Repräsentation der Validierungsinformationen umstellen. Bezüglich der Transformation von Modellinhalten ergibt sich der Vorteil, dass der für die Validierung relevante Teil der entwicklungsspezifischen Notation auch eine Teilmenge der validierungsspezifischen Notation ist, wodurch eine einfache Transformation der Inhalte möglich ist.

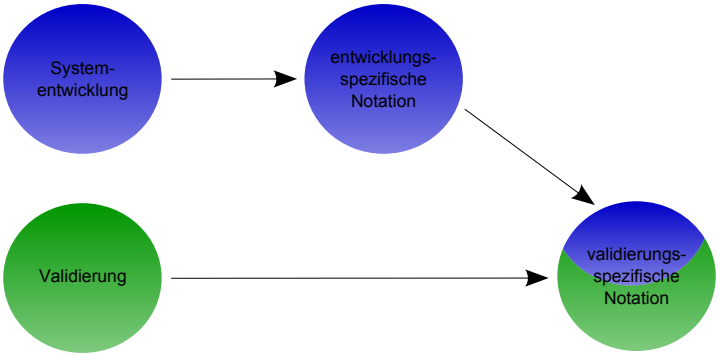


Abbildung 8.4: Schematische Darstellung einer validierungsspezifischen Notation mit einer entwicklungsspezifischen Teilmenge

Die zweite Lösungsmöglichkeit reduziert den notwendigen Mehraufwand zur Modellierung der Validierungsinformationen bereits deutlich. Die Verwendung des Profilmechanismus wirkt sich positiv auf die Anforderungen eins und zwei aus. Dass eine Teilmenge der entwicklungsspezifischen Notation allerdings zur validierungsspezifischen Notation gehört, erschwert die Anwendung der Modellierungsart auf verschiedene Domänen. Möch-

te der Architekt zudem die Eingangsdaten verfahrensspezifisch und nicht vollständig auf eine andere Domäne übertragen, ist eine Modularisierung der validierungsspezifischen Notation notwendig. Dies lässt sich beispielsweise durch die Verwendung eines eigenen Profiles pro Validierungszielverfahren realisieren. Die validierungsspazifische Notation setzt sich dann aus mehreren validierungszielspezifischen Notationen zusammmen. Durch eine domänenunabhängige Dokumentation der aus dem Entwicklungsmodell übertragenen Informationen, d. h. unabhängig von den verwendeten UML-Elementen, wird auch die zweite Anforderung erfüllt. Der dafür notwendige Mehraufwand amortisiert sich durch die Wiederverwendung der Notationen in anderen Projekten. Die vom Entwicklungsmodell unabhängige Dokumentation ist auch für Projekte derselben Domäne von Vorteil, da die Notation des Entwicklungsmodells nicht nur domänen- sondern auch projektspezifisch ist.

Bewertung der Modellierungsalternativen Eine nicht zu vernachlässigende Projektrandbedingung ist das eingesetzte Modellierungswerkzeug. Praktischen Erfahrungen nach wird der Profilmechanismus von vielen Werkzeugen gut unterstützt. Falls die Erweiterung des UML-Metamodells vom Werkzeug unterstützt wird, benötigt dieser im Vergleich zum Profilmechanismus einen höheren Modellierungsaufwand. Es ist die Aufgabe des Architekten, anhand der Randbedingungen des Projekts und unter Berücksichtigung aller Vor- und Nachteile eine geeignete Notation für die validierungsspezifischen Daten zu erarbeiten. Tabelle 8.1 kann dem Architekten dabei als erster Anhaltspunkt dienen. Sie stellt die beschriebenen Lösungsmöglichkeiten den genannten Aspekten gegenüber, die bei der Erstellung einer Notation berücksichtigt werden sollten. Die Bewertung der verschiedenen Aspekte für die jeweilige Lösungsmöglichkeit erfolgt relativ zu den anderen Möglichkeiten.

Tabelle 8.1: Übersicht zu ausgewählten Lösungsmöglichkeiten für die UML-konforme Realisierung einer validierungsspezifischen Notation

	Einheitliche Notation		**Getrennte Notationen**	
	Metamodell	Profil-mechanismus	Metamodell	Profil-mechanismus
Kopplung der Notationselemente	hoch		gering	mittel
Kohäsion zwischen den Notationen	hoch		gering	mittel
Transformationsaufwand	niedrig		hoch	mittel
Synchronisationsaufwand	niedrig		hoch	mittel
Erstellungsaufwand	hoch	niedrig	hoch	niedrig
Modellierungsaufwand	hoch	niedrig	hoch	niedrig
Verwendung der validierungsspezifischen Notation in mehreren Domänen	schlecht		gut	mittel - gut

8.3 Dokumentationsorte für die validierungsspezifischen Daten

Für die Ablage der validierungsspezifischen Daten sind grundsätzlich zwei Möglichkeiten denkbar. Zum einen können sie zusammen mit den entwicklungsspezifischen Daten im Entwicklungsmodell dokumentiert werden. Auf diese Weise sind alle Daten in einem Modell und es muss nur ein Modell gepflegt werden. Abhängig von der gewählten Modellierungsart der VSN (siehe Kapitel 8.2) ist ggf. kein Transformationsaufwand für die Modellelemente von der entwicklungsspezifischen in die validierungsspezifische Notation erforderlich. Die zweite Möglichkeit sieht eine von den entwicklungsspezifischen Daten getrennte Dokumentation in einem separaten Modell, dem Validierungsmodell, vor. Hier entsteht unabhängig von der Modellierungsart der VSN ein Transformations- und Synchronisationsaufwand, allerdings sind die verschiedenen Daten voneinander getrennt, wodurch sich Änderungen an der Architektur nicht unmittelbar auf die Ergebnisse der Validierung bzw. auf die Dokumentation des Systems im Entwicklungsmodell auswirken. Darüber hinaus wird das Entwicklungsmodell nicht mit den für die Architekturvalidierung benötigten Daten angereichert (separation of concern). In der vorliegenden Arbeit wird die Dokumentation der Daten in zwei separaten Modellen bevorzugt.

Ein Modell für entwicklungs- und validierungsspezifische Daten Durch die Verwendung von nur einem UML-Modell enthält dieses die Informationen der Systementwicklung, die sich sinnvoll modellieren lassen, und alle Informationen, die für die Architekturvalidierung benötigt werden. Die UML erlaubt durch die Diagramme eine Sichtenbildung auf die Modellinhalte, so dass nur ausgewählte Informationen angezeigt werden. Die Auswahl der

im Diagramm angezeigten Inhalte reicht vom Weglassen ganzer Elemente oder Beziehungen bis zum Ausblenden von Attributen oder Stereotypes eines Elements. Das zusätzliche Erstellen validierungsspezifischer Diagramme erzeugt allerdings einen Mehraufwand. In Abhängigkeit von der gewählten validierungsspezifischen Notation sind bei der Verwendung von nur einem Modell keine Transformationen notwendig. Für den Fall, dass die validierungs- und entwicklungsspezifische Notation vollständig unabhängig voneinander sind, wird auch bei der Verwendung von nur einem Modell eine Transformation[149] zwischen den Notationen benötigt.

Als Nachteil für die Ein-Modell-Lösung sind, wie bei der Bewertung der Lösungsmöglichkeiten für die validierungsspezifische Notation, die Kopplung und Kohäsion der entwicklungs- und validierungsspezifischen Daten zu nennen. Um die Modellinhalte als Eingangsdaten für die Validierungszielverfahren verwenden zu können, müssen diese konsistent sein. Durch die stetige Weiterentwicklung des Systems kann dies aber nicht zu jedem Zeitpunkt sichergestellt werden. Wird die Rolle des Architekten von mehreren Personen wahrgenommen, kann sich die Architektur verändern ohne dass die für die Validierung zuständige Person dies realisiert. Im schlechtesten Fall werden unterschiedliche Eingangsdaten für die Validierungszielverfahren verwendet, so dass die Ergebnisse nicht vergleichbar sind bzw. es zu nur schwierig nachzuvollziehenden Validierungsergebnissen kommt. Umgekehrt wirken sich Veränderungen an der Architektur durch den Architekten im Zuge der Suche nach einer validen Architektur sofort auf die Entwicklungsdokumentation aus. Der Entwickler ist in dieser Suchphase mit einer sich ständig ändernden Architektur konfrontiert. Bezüglich der Kohäsion ist

[149] Im Sinne der Zeitersparnis wird stets eine automatisierte Transformation angestrebt. Sie kann allerdings auch manuell durchgeführt werden. Die Abbildung der Elemente von der einen auf die andere Notation erfolgt dann im Kopf des Modellierers.

anzumerken, dass der Validierungsprozess ein Hilfsmittel für den Architekten darstellt. Er sollte deshalb keinen Einfluss auf die Entwicklungsdokumentation haben, weshalb die validierungs- und entwicklungsspezifischen Daten voneinander getrennt werden sollten.

Zwei Modelle für entwicklungs- und validierungsspezifische Daten Die Trennung von entwicklungs- und validierungsspezifischen Daten in zwei verschiedene Modelle, das Entwicklungsmodell und das Validierungsmodell, bringt ähnliche Vorteile mit sich wie die Aufteilung in eine entwicklungsspezifische und eine validierungsspezifische Notation. Das Validierungsmodell enthält alle für die Validierungszielverfahren benötigten Informationen, wobei eine Teilmenge dieser Daten bereits im Entwicklungsmodell enthalten ist. Werden die Diagramme aus dem Entwicklungsmodell mit den Elementen zusammen ins Validierungsmodell übertragen, enthalten sie nur noch die im Validierungsmodell vorhandenen Informationen. Der notwendige Mehraufwand zur Erstellung von validierungsspezifischen Diagrammen entfällt damit. Die redundante Datenhaltung ermöglicht die notwendige Entkopplung zur Durchführung der Architekturvalidierung unabhängig vom Fortschreiten der Systementwicklung. Auf der Suche nach einer validen Architektur kann sich die Architektur also nicht mehr für den Architekten unbemerkt verändern. Temporäre Inkonsistenzen, die sich durch die Weiterentwicklung des Systems ergeben, haben nur Auswirkung auf das Entwicklungsmodell; die Daten im Validierungsmodell sind davon nicht betroffen. Auch die Veränderungen an der Architektur während der Suche des Architekten nach einer validen Architektur oder nach möglichen Alternativen sind auf das Validierungsmodell beschränkt.

Die für die niedrige Kopplung benötigte redundante Datenhaltung bringt aber einen Mehraufwand für die Transformation der Daten zwischen den

Modellen mit sich. Hier ist eine Datensynchronisation notwendig. Beim initialen Erstellen des Validierungsmodells besteht die Synchronisation nur aus dem Kopieren der für die Validierung relevanten Daten aus dem Entwicklungsmodell in das neue Validierungsmodell. Bei einem vorhanden Validierungsmodell empfiehlt es sich, eines der Modelle als primäre Datenquelle auszuwählen, so dass die Daten des anderen Modells überschrieben werden. Aus praktischen Erfahrungen heraus eignet sich das Entwicklungsmodell besser als primäre Quelle, da in diesem die verbindliche und bezogen auf die Modellierung[150] vollständige Architekturdokumentation modelliert ist. Die fehlenden Daten müssen dann manuell oder über andere Transformationen[151] hinzugefügt werden. Verändern sich während der Architekturvalidierung allerdings die redundant gehaltenen Daten im Validierungsmodell, müssen diese Änderungen ins Entwicklungsmodell übertragen werden. Diese Datentransformationen lassen sich gut mit Hilfe exitierender Ansätze automatisieren, so dass für den Architekten nur wenig Mehraufwand entsteht. Falls allerdings in beiden Modellen unabhängig voneinander Änderungen an den gleichen Elementen vorgenommen werden, ist ein guter Algorithmus zum Zusammenbringen der unterschiedlichen Informationen oder ein manuelles Eingreifen notwendig. Bei größeren Änderungen an der Architektur im Entwicklungsmodell ist aber meist keine Synchronisation der Daten aus dem Validierungsmodell mehr erforderlich, da die Validierungsergebnisse dadurch veraltet sind. In der praktischen Anwendung kommen ungewollte Änderungen in beiden Modellen eher selten vor, da die Architektur meist nur vom Architekten oder nach Absprache mit diesem verändert wird.

[150] Einige Daten eignen sich nicht zur Modellierung in einem UML-Modell. Die Architektur einer Leiterplatte oder detaillierte, bitgenaue Datenstrukturen lassen sich mit anderen Werkzeugen effizienter dokumentieren.

[151] Hier ist beispielsweise die Übernahme von Daten aus einer Datenbank in das Modell denkbar.

In diesem Kapitel wird auf Anpassungsmöglichkeiten des in der Arbeit vorgestellten Ansatzes näher eingegangen. Da kein Projekt dem anderen gleicht, sollte der Architekt den Validierungsprozess den jeweiligen projektspezifischen Bedingungen anpassen können. So kann der Architekt selber entscheiden, ob er die entwicklungs- und validierungsspezifischen Informationen in einem oder in zwei voneinander getrennten Modellen dokumentieren möchte. Auch bei der Wahl der Modellierungsmöglichkeiten für die VSN kann der Architekt verschiedene Wege beschreiten. Falls der Validierungsprozess in mehreren Projekt eingesetzt wird, kann der Architekt ggf. auf bereits verwendete Validierungszielverfahren zurückgreifen. Mit den vorgestellten Basisprofilen lassen sich auch VSN-Elemente wiederverwenden. Dass eine Wiederverwendung des Ansatzes nicht nur in einer technischen Domäne möglich ist, darauf geht das nachfolgende Kapitel ein.

9 Übertragung auf eine andere Domäne

Das in Kapitel 3 vorgestellte und in Kapitel 4 und 6 verwendete Beispiel Radar-System entstammt einer technischen Domäne. Die Verwendung des Validierungsprozesses ist jedoch keineswegs auf diese Domäne beschränkt, sondern lässt sich grundsätzlich auch auf andere Domänen übertragen. Als Nachweis für diese Aussage wird im Folgenden der Validierungsprozess auf ein Workflow-System angewendet. Workflow-Systeme entstammen einer eher organisatorisch geprägten Domäne, die sich mit Prozessen innerhalb eines Unternehmens und den dafür erforderlichen Ressourcen beschäftigt. Der Architekt eines Workflow-Systems steht hier nicht nur der Herausforderung gegenüber, die abstrakten Prozesse zu modellieren, sondern eine Architektur zu erarbeiten, die eine Bearbeitung aller konkreten Abläufe in einem Unternehmen gemäß den Unternehmensvorgaben ermöglicht. Bei der Vielzahl an Prozessen und den Abhängigkeiten zwischen diesen ist eine Unterstützung erforderlich. Eine Einführung in die Domäne und die verwendeten Begriffe erfolgt in Kapitel 9.1. Wofür genau der Validierungsprozess verwendet werden kann, beschreibt Kapitel 9.2.

Da die Erarbeitung eines zweiten Beispiels, das mit der Detaillierungstiefe des Radar-Beispiels vergleichbar wäre, den Umfang der vorliegenden Arbeit sprengen würde - die Entwicklung des Radar-Beispiels dokumentiert vier Jahre praktischer Arbeit - wird die Anwendung des Validierungsprozesses in Kapitel 9.3 anhand einiger Workflows skizziert und nicht bis ins letzte Detail ausgeführt. Die Grundidee für das Beispiel entstammt einem Kundenprojekt der Firma SOPHIST[152]. Das System und die Problemstellung

[152] http://www.sophist.de

werden hier aber in vereinfachter Form verwendet. Der Fokus liegt auf der Adaptierbarkeit des Ansatzes auf die Domäne Workflow-Systeme.

Zusammenfassend lässt sich sagen, dass der Validierungsprozess auf verschiedene Domänen anwendbar ist. Zwar lassen sich nicht alle architekturrelevanten Aspekte sinnvoll mit Hilfe von Simulationen validieren (z. B. Prüfung der Druckqualität), die Validierungsziele mit ihren Abhängigkeiten untereinander und den Abhängigkeiten zu den Anforderungen sind allerdings für die Impact-Analyse im Allgemeinen verwendbar. Auch der Einsatz verschiedener Modellierungssprachen (für Prozesse häufig die Business Process Model and Notation (BPMN)) stellt grundsätzlich kein Problem dar. Durch die Verwendung der vielseitigen und erweiterbaren UML lassen sich unterschiedlichste Informationen dokumentieren und für die Nutzung in Validierungszielverfahren gezielt auslesen.

9.1 Einführung in die Domäne Workflow-Systeme

Während die Domäne Radar-System technisch geprägt ist, geht die Domäne Workflow-System eher in Richtung Organisation bzw. in Richtung Koordination verschiedener Aufgaben. Durch das Erledigen der Aufgaben wird ein für das Unternehmen wertvolles Ergebnis erzielt. Ein solches Ergebnis kann beispielsweise ein zum Verkauf erstelltes Produkt, aber auch das Einstellen von Personal sein. Welche Prozesse für ein Unternehmen wichtig sind und welche Bedeutung diesen zukommt, wird auf der strategischen Ebene festgelegt. Auf der fachlich-konzeptionellen Ebene werden durch die Definition von Geschäftsprozessen die für das gewünschte Prozessergebnis notwendigen Aufgaben und deren Abarbeitungsreihenfolge abgeleitet. Das Abarbeiten der Aufgaben erfolgt auf der operativen Ebene. Die Bear-

beiter werden durch Anwendungssysteme unterstützt, wobei der Grad der Unterstützung von der Klasse des Anwendungssystems abhängt. Handelt es sich um strukturierte, komplexe Aufgaben mit einem geringen Automatisierunggrad, sprechen Ferstl et al. [FS13, Seite 455] von Workflow-Systemen. Workflow-Systeme koordinieren die zur Erledigung der Aufgaben erforderlichen Arbeitsschritte und stellen dem für die Aufgabe zuständigen Bearbeiter die erforderlichen Ressourcen[153] bereit. Abhängig von der Charakteristik des Workflows bzw. der Arbeitsschritte kann die Bearbeitung auch durch ein Computersystem übernommen werden.

In einem Unternehmen existieren viele verschiedene Workflows, die zum Teil unabhängig voneinander ablaufen. Andere wiederum teilen sich dieselben Ressourcen oder Bearbeiter. Aus Sicht des Architekten einer unternehmensweiten Architektur liegt die Herausforderung zum einen in der optimalen Koordinierung der verschiedenen Workflows und zum anderen in der Bereitstellung ausreichender IT-Ressourcen, damit die verschiedenen und zum Teil einander entgegenstehenden Zielvorgaben eines Unternehmens (z. B. Kostenreduktion, Verringerung von Bearbeitungszeiten, Erhöhung der Ergebnisqualität) erreicht werden können. Die Koordinierung übernehmen die Workflow-Systeme, die durch Funktionalitäten wie Simulation und Analyse sowie Überwachung der konkreten Workflows die zuständigen Personen bei der Optimierung unterstützen [Gad12] [Wor99]. Bei der Festlegung der IT-Ressourcen bieten die Workflow-Systeme hingegen keine Unterstützung an.

Nach diesem Kurzüberblick gehen die nachfolgenden Kapitel etwas detaillierter auf verschiedene Aspekte der Workflow-Domäne ein. Dazu wer-

[153] Dabei kann es sich beispielsweise um materielle Dinge, erforderliche Daten oder auch Software handeln.

den zunächst in Kapitel 9.1.1 die Begriffe Geschäftsprozess, Workflow und Workflow-System definiert. Es werden zudem die Unterschiede zwischen Geschäftsprozessen und den Workflows aufgezeigt und noch etwas detaillierter auf die Workflow-Systeme eingegangen. Kapitel 9.1.3 stellt einige Modellierungsmöglichkeiten für Prozesse vor. Zu den bekanntesten zählen hier sicherlich die BPMN, die erweiterte ereignisgesteuerte Prozesskette (eEPK) [KNS92] und die UML.

9.1.1 Begriffsdefinitionen

Der Begriff Prozess wird für verschiedene Prozessarten und auf ganz unterschiedlichen Abstraktionsstufen verwendet. Dahinter verbirgt sich eine Abfolge von Aktivitäten, die aus bestimmten Eingaben ein Ergebnis erzeugen. Im besten Fall hat dieses Ergebnis direkt oder indirekt (z. B. in Kombination mit anderen Prozessergebnissen) einen Wert für jemanden, z. B. für einen Kunden (vgl. [HCK03]). Bei den Arten lässt sich zwischen technischen und betriebswirtschaftlichen Prozessen unterschieden. Gadatsch nennt hier als Beispiele zum einen die Montage eines Motors und zum anderen das Einstellen eines Mitarbeiters [Gad12, Kapitel 1.3.1].

Geschäftsprozess Ein Prozess auf der fachlich-konzeptionellen Ebene wird auch Geschäftsprozess genannt und in der vorliegenden Arbeit wie folgt definiert:

> "Ein Geschäftsprozess ist eine zielgerichtete, zeitlich-logische Abfolge von Aufgaben, die arbeitsteilig von mehreren Organisationen oder Organisationseinheiten unter Nutzung von Informations- und Kommunikationstechnologien ausgeführt werden können. Er dient der Erstellung von Leistungen entsprechend den vorgegebenen, aus der Unternehmensstrategie abgeleiteten Prozesszielen." [Gad12, S. 36f]

In Abhängigkeit vom Umfang und der Komplexität lassen sich Geschäftsprozesse auf verschiedenen Detaillierungsstufen beschreiben. Bei einer modellbasierten Dokumentation erfolgt dies nicht nur in einem einzigen Diagramm, sondern die verschiedenen Aspekte wie z. B. Abläufe, Daten und Organisationen werden aus unterschiedlichen Sichten beschrieben. Dies reduziert die Komplexität der Modelle und verbessert deren Transparenz und Verständlichkeit (vgl. [Sin96]). Die Aufgaben eines Geschäftsprozesses sind ausreichend detailliert beschrieben, falls sie "je in einem Zug von einem Mitarbeiter ohne Wechsel des Arbeitsplatzes" [Gad12, Seite 37] erledigt werden können.

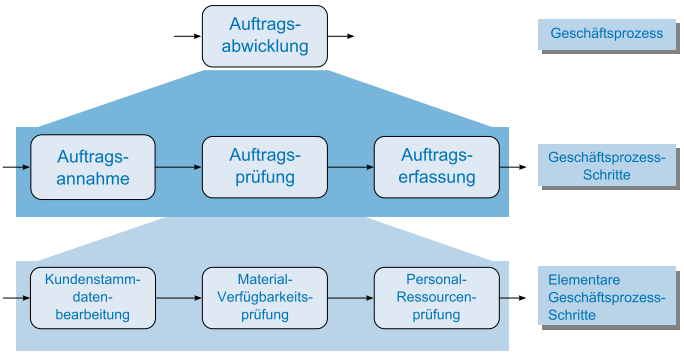


Abbildung 9.1: Zerlegung des Geschäftsprozesses Auftragsprüfung, nach [Gad12]

Es wird dann von einem elementaren Geschäftsprozessschritt gesprochen, der als Grundlage für die Verfeinerung durch Workflows verwendet werden kann. Abbildung 9.1 verdeutlicht dies anhand des Geschäftsprozesses Auftragsabwicklung. Dieser lässt sich zunächst in die Geschäftsprozessschritte Auftragsannahme, -prüfung und -erfassung untergliedern. Der Schritt Auftragsprüfung wiederum lässt sich in die drei elementare Geschätsprozessschritte Kundendatenstammbearbeitung, Materialverfügbarkeitsprüfung und Personalressourcenprüfung unterteilen. Bei solchen elementaren Schritten ist kein Bearbeiter- und Arbeitsplatzwechsel mehr erforderlich [Gad12].

Workflow Ein elementarer Geschäftsprozessschritt kann auf der operativen Ebene mit Hilfe von Workflows weiter verfeinert werden. In der vorliegenden Arbeit wird unter dem Begriff Workflow folgendes verstanden:

> "Ein Workflow ist ein formal beschriebener, ganz oder teilweise automatisierter Geschäftsprozess. Er beinhaltet die zeitlichen, fachlichen und ressourcenbezogenen Spezifikationen, die für eine automatische Steuerung des Arbeitsablaufes auf der operativen Ebene erforderlich sind." [Gad12, S. 41]

Unter dem Begriff *formal* versteht Gadatsch eine Beschreibung mit grafischen Methoden [Gad12, Kapitel 2.4]. Hierunter fällt für ihn beispielsweise auch die UML, die aufgrund von Mehrdeutigkeiten [OD11] eher als semiformal [vL01] bezeichnet werden kann.

Abhängig von der Charakteristik des Workflows werden die verschiedenen Schritte des Arbeitsablaufs von Mitarbeitern oder Software durchge-

führt. Dabei unterscheidet Gadatsch zwischen drei verschiedenen Workflow-Typen [Gad12, Seite 42ff.]:

- Allgemeiner Workflow,
- Fallbezogener Workflow und
- Ad hoc-Workflow.

Der Strukturierungsgrad der Arbeitsabläufe nimmt vom allgemeinen über den fallbezogenen zum Ad hoc-Workflow ab. Damit einher gehen auch die Planbarkeit der Ablaufschritte und der Detaillierungsgrad der Workflow-Spezifikation. Für einen allgemeinen Workflow existiert eine detaillierte Beschreibung der Abläufe, die Bearbeitung kann quasi nach *Schema F* erfolgen. Eine derart detaillierte Beschreibung ist vor allem für häufig stattfindende und sich wiederholende Tätigkeiten sinnvoll. Die Beantragung von Urlaub ist ein typisches Beispiel für einen allgemeinen Workflow. Ein fallbezogener Workflow besteht überwiegend aus vorgegebenen Schritten, abhängig von individuellen Entscheidungen können aber auch Schritte ausgelassen werden. Dies ist beispielsweise beim Einstellen von Mitarbeitern erforderlich. Der Bearbeiter hat eine Mitverantwortung für das Ergebnis. Der Ablauf eines Ad hoc-Workflows lässt sich nicht vorher planen, die Verantwortung für das zu erzeugende Ergebnis liegt bei den Bearbeitern. Hierunter fallen vor allem kreative Aufgaben wie die Erstellung eines Marketingkonzepts.

Eine weitere Unterscheidungsmöglichkeit stellt der Grad der Computerunterstützung für einen Workflow dar. Hier wird zwischen

- freien,
- teilautomatisierten und
- automatisierten Workflows

unterschieden. Bei freien Workflows steht dem Mitarbeiter keine Computerunterstützung zur Verfügung. Hier handelt es sich häufig um inhaltliche Entscheidungen. Bei teilautomatisierten Workflows sind sowohl Anwendungen als auch Mitarbeiter an den Ablaufschritten beteiligt und bei automatisierten Workflows erfolgt die Bearbeitung vollständig durch eine Anwendung [Gad12, Kapitel 1.3.2].

Tabelle 9.1: Gegenüberstellung Geschäftsprozess und Workflow, nach [Gad12, S. 46]

	Geschäftsprozess	**Workflow**
Ziel	Analyse und Gestaltung von Arbeitsabläufen im Sinne gegebener (strategischer) Ziele	Spezifikation der technischen Ausführung von Arbeitsabläufen
Gestaltungsebene	Konzeptionelle Ebene mit Verbindung zur Geschäftsstrategie	Operative Ebene mit Verbindung zu unterstützender Technologie
Detaillierungsgrad	In einem Zug von einem Mitarbeiter an einem Arbeitsplatz ausführbare Arbeitsschritte	Konkretisierung von Arbeitsschritten hinsichtlich Arbeitsverfahren sowie personeller und technologischer Ressourcen

Ein Workflow-Modell oder auch Workflow-Schema stellt eine abstrakte Beschreibung der Ablaufschritte eines elementaren Geschäftsprozesses auf der operativen Ebene dar. Letztendlich ist es eine detaillierte Spezifikation eines Geschäftsprozessschrittes. Eine konkrete Abfolge von Schritten eines Workflow-Schemas wird als Workflow-Instanz bezeichnet. Für das

Workflow-Schema Urlaubsbeantragung ist die Workflow-Instanz die Bearbeitung eines Urlaubsantrags für einen konkreten Mitarbeiter (z. B. für den Autor der vorliegenden Arbeit) [Gad12].

Der Unterschied zwischen den Begriffen Geschäftsprozess und Workflow ist in Tabelle 9.1 zusammengefasst. Dabei wird auf die Ziele, die Gestaltungsebene und den Detaillierungsgrad eingegangen. Kurz zusammengefasst lässt sich sagen: Aus Sicht der Workflows beschreibt ein Geschäftsprozess das *was* und ein Workflow das *wie*.

Für die computergestützte Durchführung von Workflows sind Anwendungssysteme erforderlich. In Abhängigkeit von der Workflow-Charakteristik unterscheiden Ferstl und Sinz [FS13] drei verschiedene Klassen von Anwendungssystemen:

- Integrierte Anwendungssysteme,
- Workflow-Systeme und
- Workgroup-Systeme.

Mit integrierten Anwendungssysteme werden strukturierte und automatisierte Workflows ausgeführt. Workflows mit einem geringeren Automatisierungsgrad werden durch Workflow-Systeme durchgeführt und die Bearbeitung freier Workflows wird durch Workgroup-Systeme unterstützt. Im Bezug auf die Architektur haben sich die verteilten Systeme gegenüber den monolithischen Systemen durchgesetzt [FS13, Kapitel 11].

Da das in Kapitel 9.3 vorgestellte Beispiel teilautomatisierte Workflows verwendet, wird im Folgenden näher auf Workflow-Systeme eingegangen.

9.1.2 Workflow-Systeme

Der Arbeitsablauf eines teilautomatisierten Workflows besteht aus strukturierten und komplexen Aufgaben, deren Automatisierungsgrad niedrig ist. Die Durchführung erfolgt mit Hilfe von sogenannten Workflow-Systemen. Die Bearbeitung der einzelnen Aufgaben erfolgt überwiegend durch Menschen und nur vereinzelt durch unterstützende Anwendungen [FS13, Seite 455]. Der Kern eines Workflow-Systems bildet ein sogenanntes Workflow-Management-System, das für die Koordinierung der verschiedenen Workflow-Instanzen verantwortlich ist. Es ermöglicht eine effiziente Durchführung der Workflows aller Geschätsprozesse eines Unternehmens. In Abhängigkeit von den verwendeten Anwendungen zur Bearbeitung der Workflows und dem Grad der Ablaufkontrolle durch das Workflow-Management-System unterscheidet Gadatsch verschiedene Integrationsstufen. Diese reichen von null (keine Applikationsunterstützung) bis vier (automatisierte Durchführung ohne Beteiligung eines Mitarbeiters). Workflow-Systeme lassen sich in Stufe zwei oder drei einordnen. Workflows werden hier computergestützt ausgeführt und eine für die Aufgabe geeignete Anwendung (z. B. ein Textverarbeitungsprogramm zum Schreiben eines Briefs) wird für den Mitarbeiter gestartet. Bei Stufe drei wird die Arbeit mit der unterstützenden Anwendung soweit wie möglich reduziert, so dass der Mitarbeiter nur noch die erforderlichen Daten angeben muss (es existieren Briefvorlagen, die anhand von Daten vom System vervollständigt werden) [Gad12, Kapitel 4.5].

Workflow-Management-Systeme Der Begriff Workflow-Management-System und die davon bereitzustellenden Funktionalitäten sind in der Literatur nicht einheitlich definiert [Gad12, Seite 227]. Aus diesem Grund

wird in der vorliegende Arbeit eine recht einfach gehaltene Definition aus dem Glossar der Workflow Management Coalition (WfMC) verwendet:

> "A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications." [Wor99, Seite 9]

Die WfMC ist eine globale Organisation von Herstellern, Forschern und Analysten mit dem Fokus Workflows und Business Process Management (BPM) [WfM13]. Sie ist für die Festlegung verschiedener in der Praxis häufig eingesetzter Spezifikationen wie Wf-XML[154] und XPDL[155] verantwortlich. Zudem hat sie ein Referenzmodell (siehe Abbildung 9.2) für Workflow-Systeme erarbeitet und 2005 veröffentlicht.

Nach dem WfMC-Referenzmodell besteht der Kern eines modular aufgebauten Workflow-Systems aus dem *Workflow Enactment Service*, der für die Ausführung von Prozessdefinitionen (den Workflows) mit Hilfe verschiedener Workflow-Engines verantwortlich ist. Mit Hilfe der *Workflow-Application Programming Interface (API) and Interchange Formats* sowie den fünf Schnittstellen ist eine standardisierte Kommunikation zwischen dem Kern und den externen Komponenten möglich. Das erleichtert die Anbindung von Anwendungen verschiedener Hersteller an das Workflow-System.

[154] Workflow-XML ist ein Standard für die Zusammenarbeit verschiedener BPM-Engines. Auf diese Weise können Workflows in verschiedenen Engines zu einem Gesamtprozess verbunden werden.

[155] XML Process Definition Language ist das Standardformat zum Austausch von Prozessdefinition mit Hilfe von XML. Dabei erlaubt es neben dem Austausch der Prozesssemantik auch den Austausch der grafischen Informationen.

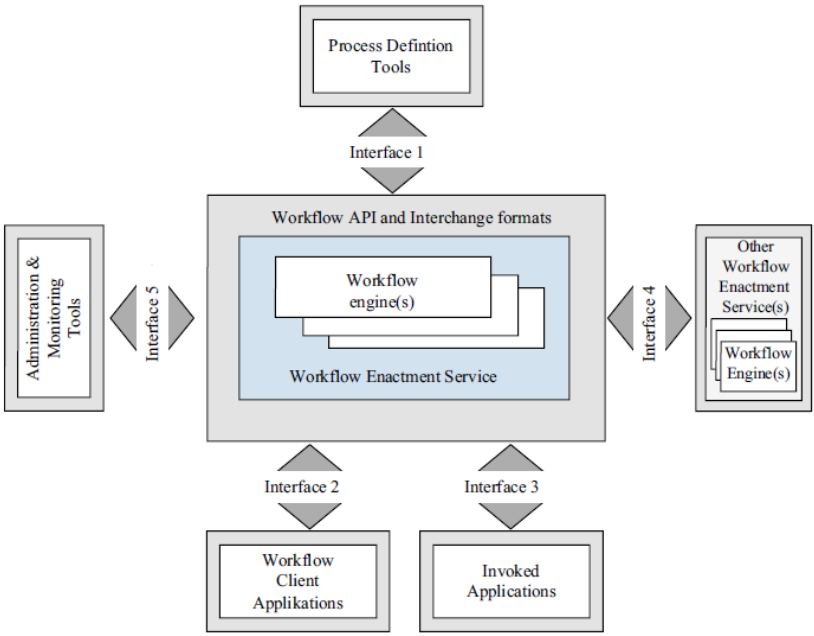


Abbildung 9.2: Referenzmodell der WfMC, nach [Gad12]

Process Definition Tools ermöglichen die Modellierung der Ablaufschritte eines Workflows. Mit Hilfe von XML Process Definition Language (XPDL) können diese dann vom Kern ausgeführt werden. Die *Workflow-Client-Applications* sind für die Benachrichtigung der Workflow-Bearbeiter zuständig. Auf diese Weise wird ihnen beispielsweise mitgeteilt, wann sie einen Workflow durchzuführen haben. Diese einheitliche Schnittstelle ist erforderlich, da über *Other Workflow Enactment Service(s)* auch externe Workflow-Engines zur Ausführung der Workflows eingesetzt werden können. Als Unterstützung des Mitarbeiters bei teilautomatisierten und automatisierten Workflows kann das Workflow-System externe Anwendungen (*Invoked Applications*) aufrufen. Zur Verwaltung und Überwachung des Workflow-

Systems lassen sich *Administration & Monitoring Tools* einbinden. Auf diese Weise können die ausgeführten Workflows analysiert und optimiert werden [Gad12].

Abbildung 9.3 zeigt die verschiedenen Funktionen eines Workflow-Systems. Dabei handelt es sich um die *Modellierung und Simulation von Workflows*, die *Instanziierung und Ausführung von Workflows* sowie das *Monitoring laufender Vorgänge und Analyse ausgeführter Vorgänge*. Bei der Modellierung werden neben den Abläufen und Daten auch die in der Organisation verfügbaren Ressourcen sowie die für die Unterstützung der Bearbeitung verfügbare Software spezifiziert.

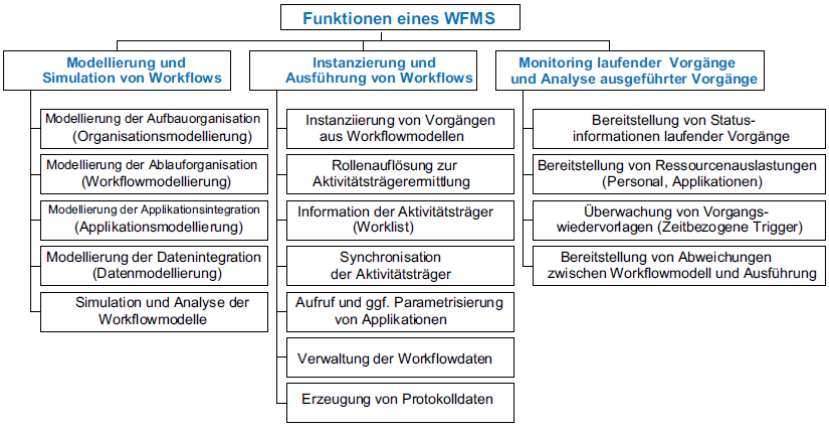


Abbildung 9.3: Funktionen eines WfMC, nach [Gad12]

Diese Modelle werden vor der Ausführung durch die Engine formal und inhaltlich mit Hilfe von Simulationen geprüft. Das Erstellen der konkreten Workflow-Instanzen anhand des geprüften Workflow-Modells sowie die Ausführung in den Workflow-Engines ist die zweite wesentliche Funktiona-

lität eines Workflow-Systems. Für die Workflow-Instanzen müssen neben den Daten auch geeignete und verfügbare Bearbeiter ermittelt, der Instanz zugewiesen und entsprechend informiert werden. Alle Vorgänge während der Workflow-Ausführung sowie die Auslastung der Ressourcen (Mensch und Software) werden protokolliert. Die letzte Kategorie der Funktionalitäten kümmert sich aktiv um die laufenden Vorgänge. So werden beispielsweise gesetzte Termine für einen Vorgang überwacht oder Ausnahmeregelungen für auftretende Probleme bei der Bearbeitung aktiviert. Die Überprüfung der SOLL- und IST-Werte sowie die entsprechenden Reaktionen auf zu hohe Abweichungen gehören nicht zu dieser Kategorie der Funktionalitäten.

Ziele Die Ziele der Workflow-Systeme stimmen mit den Zielen des Workflow-Managements im Allgemeinen (die Koordinierung der Workflows ohne Computerunterstützung) überein. Sie werden auf der strategischen Ebene, also von der Unternehmensleitung, vorgegeben. Dazu gehören unter anderem:

- Verbesserung der Prozessqualität sowie permanente Qualitätssicherung,
- Verkürzung der Durchlaufzeiten,
- Automatisierung der Prozesssteuerung,
- Permanente Anpassung an organisatorische Änderungen und
- Verbesserung der Kundenzufriedenheit [Gad12, Kapitel 1.4].

Zu den bekannten und von den Funktionalitäten sehr umfangreichen Systemen gehören beispielsweise Architektur integrierter Informationssysteme (ARIS) von IDS Scheer sowie Bonapart von Emprise Process Management.

9.1.3 Prozessmodellierung

Prozesse lassen sich mit Hilfe von grafischen Modellierungssprachen dokumentieren. Mit Hilfe von Sichten können die Transparenz einer modellbasierten Dokumentation und der Überblick über die Prozesse verbessert werden. Dadurch wird auch die Verständlichkeit erhöht (vgl. [Sin96]). Mit Hilfe einiger Modellierungssprachen können Workflow-Modelle spezifiziert werden, welche als direkte Eingabe für Workflow-Engines verwendet werden können. Einige Beispiele hierfür sind die eEPKs, BPMN und die UML.

Die eEPKs, die BPMN und die UML-Aktivitätsdiagramme zählen zu den prozessorientierten Sprachen. Im Mittelpunkt der Beschreibung steht hier die dynamische, verhaltensorientierte Sicht. eEPKs werden vor allem in Verbindung mit dem geschäftsprozessorientierten Ansatz ARIS verwendet. Sie ermöglichen eine Ablaufbeschreibungen durch eine Kombination von Funktionen und Ereignissen, wobei eine Funktion oder Aufgabe durch ein Ereignis getriggert wird und am Ende wiederum ein neues Ereignis erzeugt. Durch die Verknüpfung der Funktionen mit anderen Modellbausteinen lässt sich eine Verbindung zu den verschiedenen Sichten von ARIS herstellen. Dazu gehören u. a. die Daten- und Struktursicht[156] [FS13, Kapitel 5.3].

Zwei weitere Vertreter der prozessorientierten Sprachen sind die BPMN und die UML-Aktivitätsdiagramme. Sie gehören zu den vorgangs- und workfloworientierten Ansätzen und unterscheiden sich von den geschäftsprozessorientierten Ansätzen dadurch, dass sie nicht auf die zielbezogene Verfeinerung der Prozesse fokusiert sind, sondern auf die Beschreibung eines Lö-

[156] ARIS verwendet hier den Begriff Organisationssicht. Um bei dem Vergleich der Modellierungssprachen einheitliche Begriffe verwenden zu können, wird hier vereinfachend der Begriff Struktursicht verwendet.

sungsverfahrens für die Workflows [FS13, Kapitel 5.3]. Die BPMN bietet seit der Version 2.0 den Vorteil, dass sie direkt durch Workflow-Engines ausgeführt werden kann. Neben dem Verhalten ermöglicht sie auch die Beschreibung von Daten. Eine Struktursicht lässt sich mit ihr aber nicht erstellen.

Die Diagramme der UML sind nicht direkt auf die Modellierung von Geschäftsprozessen spezialisiert. Bis zu einem gewissen Detaillierungsgrad lassen sich damit aber auch Geschäftsprozesse bzw. Workflows beschreiben (UML-Aktivitätsdiagramme). Sie ermöglicht mit ihren verschiedenen Diagrammen neben der Ablaufbeschreibung unter anderem auch die Spezifikation von Daten und der Systemstruktur.

9.2 Anwendungsgebiet des Validierungsprozesses

Während die Simulation und Analyse der Workflow-Modelle bereits zu den von der WfMC aufgeführten Funktionalitäten eines Workflow-Management-Systems gehören[157], wird der Blick auf das Workflow-System als Ganzes vernachlässigt. Es ist für die Durchführung der Workflow-Modelle verantwortlich, wofür es neben dem Workflow-Management-System zur Koordinierung der Aufgaben auch verschiedene Anwendungen zur Unterstützung der Mitarbeiter und geeignete Mitarbeiter für die Bearbeitung der Aufgaben benötigt. Wird es als verteiltes System realisiert, stellt sich für den Architekten die Frage, ob die von ihm erstellte Architektur wirklich die Durchführung aller anfallenden Workflow-Instanzen im Rahmen der Unternehmensvorgaben (z. B. Kostenreduktion, Verringerung von Bearbeitungszeiten, Erhöhung der Ergebnisqualität) erfüllt. Dies geht über die Simulation und Analy-

[157] Nicht alle Workflow-Management-Systeme unterstüzen die von der WfMC genannten Funktionalitäten im vollen Umfang. Gerade die Simulation und Analyse wird eher von größeren Systemen angeboten [Gad12].

se der einzelnen Workflow-Modelle hinaus und berücksichtigt auch die Abhängigkeiten zwischen den verschiedenen Workflows. Beispielsweise wird ein vom Workflow-System bereitgestelltes Textverarbeitungsprogramm für mehrere Aufgaben verschiedener Workflows benötigt. Stehen hier auch bei Lastspitzen ausreichend Lizenzen zur Verfügung oder müssen einige Mitarbeiter auf freie Lizenzen warten, bevor sie den Workflow weiter abarbeiten können? Weitere Fragen wären beispielsweise:

1. Haben die Verarbeitungseinheiten, auf denen die Software-Komponenten des Workflow-Systems ausgeführt werden, ausreichend Leistung, um alle Workflow-Instanzen innerhalb einer vorgegebenen Durchlaufzeit zu bearbeiten?
2. Lassen sich mit der Anzahl der Mitarbeiter alle anfallenden Workflow-Instanzen innerhalb der Vorgaben abarbeiten?
3. Falls die Vorgaben nicht erfüllt werden können: Wo sind die Engpässe?

Der Validierungsprozess kann den Architekten bei seiner Aufgabe, eine den Unternehmensvorgaben entsprechende Architektur oder Konfiguration für das Workflow-System zu erarbeiten, unterstützen. Durch das Ableiten von Validierungszielen aus den Unternehmensvorgaben und der Festlegung eines Validierungszielverfahrens lassen sich die ersten beiden Fragen beantworten. Mit Hilfe der eingesetzten Simulationen und der Analyse der Simulationsdetails (protokollierte Daten) lassen sich auch die Engpässe finden und hoffentlich beseitigen. Der Validierungsprozess gibt dem Architekten zudem ein Werkzeug an die Hand, die Auswirkungen von Änderungen der Unternehmensvorgaben oder an der Architektur zu analysieren. Natürlich lässt sich der Validierungsprozess nicht nur für Neuentwicklungen, sondern

auch für bestehende Workflow-Systeme verwenden. Für einen ersten Schritt lässt sich auch nur eine Teilmenge der Architektur untersuchen.

9.3 Anwendung des Validierungsprozesses

Der Nachweis für die Übertragbarkeit des in der Arbeit vorgestellten Ansatzes erfolgt anhand einiger ausgesuchter Workflows aus einem Kundenprojekt der Firma SOPHIST[158]. Ähnliche Workflow-Beispiele lassen sich auch in Fallstudien in der Literatur [Gad12] [FS13] wiederfinden. Der Hauptfokus liegt auf einem zentralen Druckservice, der von verschiedenen Workflows verwendet wird. Für das aktuelle Beispiel sind dies:

- Kundenrechnung verschicken,
- Kundenkorrespondenz beantworten sowie
- allgemeine Druckaufträge bearbeiten.

Abgesehen von den Arbeitsabläufen unterscheiden sich die Workflows in den Merkmalen Verwendungshäufigkeit, Bearbeitungsdauer und Bearbeitungszeitpunkt. Kundenrechnungen beispielsweise werden monatlich zu einem bestimmten Stichtag erstellt und an die Kunden verschickt. Mit etwa 750.000 Rechnungen pro Monat ist der Druckservice zu diesem Zeitpunkt im Monat stark ausgelastet. Die Kundenkorrespondenz hingegen findet relativ regelmäßig über den Monat verteilt statt, wobei die Bearbeitungsdauer drei Tage nicht überschreiten darf. Im Mittel gibt es etwa 9.000 zu beantwortende Kundenkontakte im Monat. Neben diesen zwei Workflows muss der Druckdienst auch die intern anfallenden Druckaufträge bewältigen. Hier ist täglich mit bis zu 4.000 Druckseiten zu rechnen. Die Tabellen 9.2 und

[158] http://www.sophist.de

9.3 zeigen einen Auszug aus der Anforderungsspezifikation des Workflow-Systems. Die Anforderungen bilden die Grundlage für die Anwendung des Validierungsprozesses. Dabei werden nicht alle fachlichen und technischen Aspekte im Detail betrachtet. Der Fokus liegt auf der Veranschaulichung der Adaptierbarkeit des Ansatzes auf eine andere Domäne.

Tabelle 9.2: Auszug aus der Anforderungsspezifikation - Teil 1

ID	**Anforderung**
ZD1	Falls das aktuelle Datum gleich dem Kundenabrechnungsdatum ist, muss das System anhand der Kundentransaktionsdaten Rechnungen erstellen.
ZD1	Das System muss die Möglichkeit bieten, Rechnungen als "sachlich und rechnerisch korrekt" zu kennzeichnen.
ZD2	Nachdem eine Rechnung als sachlich und rechnerisch korrekt gekennzeichnet wurde, muss das System die Rechnung per Brief an den Kunden versenden.
ZD3	Das System muss fähig sein, eingegangene Briefkorrespondenz zu digitalisieren.
ZD5	Das System muss fähig sein, eingegangene Emailkorrespondenz zu empfangen.
ZD6	Das System muss digitalisierte Kundenkorrespondenz einem Kunden zuordnen.
ZD7	Das System muss die Möglichkeit bieten, auf Kundenkorrespondenz zu antworten.
ZD8	Das System muss die Möglichkeit bieten, Kontakt mit dem Kunden aufzunehmen.
ZD9	Das System muss die Möglichkeit bieten, unternehmensinterne Druckaufträge anzunehmen.

Tabelle 9.3: Auszug aus der Anforderungsspezifikation - Teil 2

ID	Anforderung
ZD10	Die Druckkosten einer A4-Seite (s/w) muss kleiner gleich 0,04AM-$ sein.
ZD11	Der Energieverbrauch der Druckstraße muss kleiner gleich 4KW sein.
ZD12	Das System muss die Unternehmensvorgaben zur Druckqualität einhalten.
ZD13	Das System muss Kundenrechnungen nach kleiner gleich 3 Tagen nach dem Kundenabrechnungsdatum verschicken.
ZD14	Falls die Anzahl der zu druckenden A4-Seiten kleiner gleich 20 ist, muss das System interne Druckaufträge in kleiner gleich 5 Minuten drucken.
ZD15	Die Kundenverwaltungskapazität des Systems muss größer gleich 750.000 sein.
ZD16	Die Bearbeitungskapazität des Systems für Kundenkontakte muss größer gleich 9.000 pro Monat sein.
ZD17	Die Kapazität des Systems für interne Druckaufträge muss größer gleich 4.000 pro Tag sein.

9.3.1 Entwicklungsmodell

Das Entwicklungsmodell enthält neben den Beschreibungen für die vom System realisierten Workflows auch die Beschreibung der System-Architektur. Als Modellierungssprache wird wie schon beim Radar-System die UML verwendet. Die Arbeitsabläufe der Workflows werden mit Hilfe von UML-Aktivitätsdiagrammen modelliert, die Architektur mit Komponen-

tendiagrammen. Die Verwendung der BPMN als Modellierungssprache im Entwicklungsmodell ist grundsätzlich auch denkbar. Da der Validierungsprozess für das Validierungsmodell die UML vorsieht, müssten die BPMN-Informationen allerdings in ein UML-Modell transformiert werden. Hierzu kann beispielsweise das UML-Profil für BPMN 2-Prozesse [Obj13] verwendet werden.

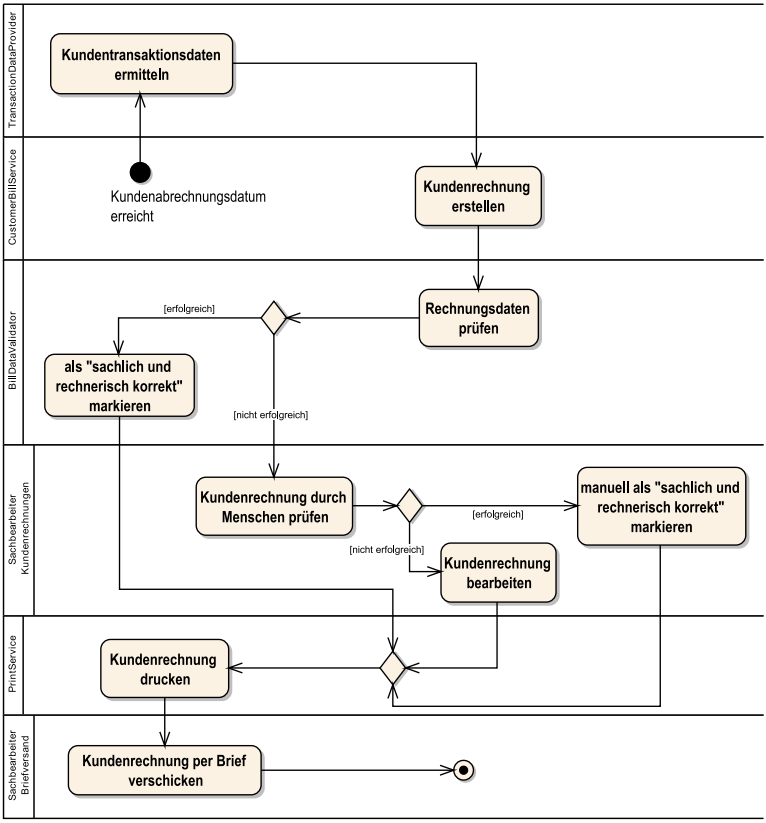


Abbildung 9.4: Ablauf des Workflows Kundenrechnungen verschicken

Abbildung 9.4 zeigt den Ablauf des Workflows *Kundenrechnung verschicken*. Er besteht aus acht Aufgaben, die mit Hilfe von Aktionen modelliert wurden. Die Aktionen sind verschiedenen Swimlanes zugeordnet, wodurch die Bearbeiter einer Aktion festgelegt sind. Sobald das Kundenabrechnungsdatum erreicht ist, werden die Kundentransaktionsdaten von der System-Komponente *CustomerTransactionDataProvider* ermittelt und anschließend als Kundenrechnung zusammengefasst. Sobald diese verfügbar sind, führt die Komponente *BillDataValidator* eine Prüfung durch. Der Großteil der Rechnungen besteht diese Prüfung und wird deshalb *als "sachlich und rechnerisch korrekt" markiert*. Ein kleiner Teil der Rechnungen muss durch einen Menschen geprüft werden (Aktion *Kundenrechnung durch Menschen prüfen*). Hierbei kann es sich um besonders wichtige oder auch säumige Kunden handeln.

Auch für ungewöhnlich hohe Umsätze oder verdächtige Transaktionen ist eine manuelle Prüfung erforderlich. Falls mit den Daten alles in Ordnung ist, markiert der *Sachbearbeiter Kundenrechnungen* die Rechnung "als sachlich und rechnerisch korrekt" und die Rechnung kann durch die System-Komponente *PrintService* gedruckt und anschließend per Brief an den Kunden verschickt werden (siehe Mitarbeiter *Sachbearbeiter Briefversand*). Falls der *Sachbearbeiter Kundenrechnungen* Probleme bei der manuellen Prüfung entdeckt, behabt er diese zunächst (Aktion *Kundenrechnung bearbeiten*), bevor er die Rechnung *manuell "als sachlich und rechnerisch korrekt" markiert*.

Abbildung 9.5 zeigt einen Ausschnitt aus der Deployment-Sicht des Workflow-Systems. Neben den bereits genannten Software-Komponenten sind auch die Hardware-Komponenten zu sehen. Das sind zum einen die zur Ausführung der Software benötigten Server und zum anderen die für den Druckdienst erforderlichen Drucker, die als Druckstraße zusammen-

gefasst sind. Die Kommunikationsverbindungen zwischen den Hardware-Komponenten sind in Abbildung 9.6 dargestellt.

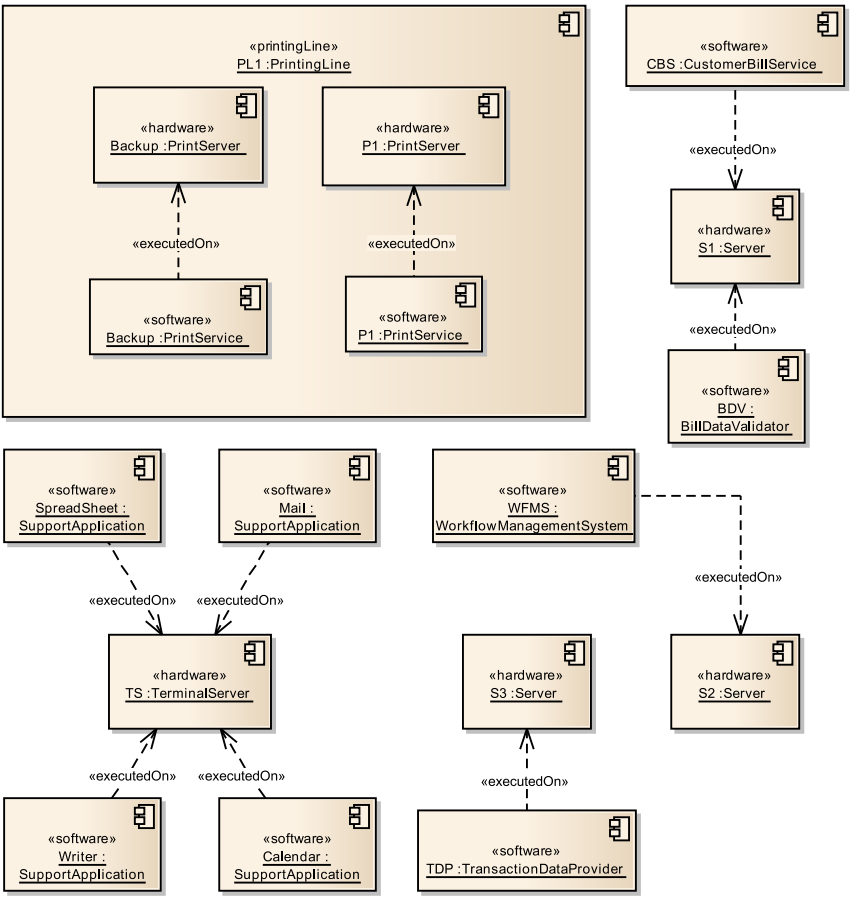


Abbildung 9.5: Ausschnitt aus der Deployment-Sicht des Workflow-Systems

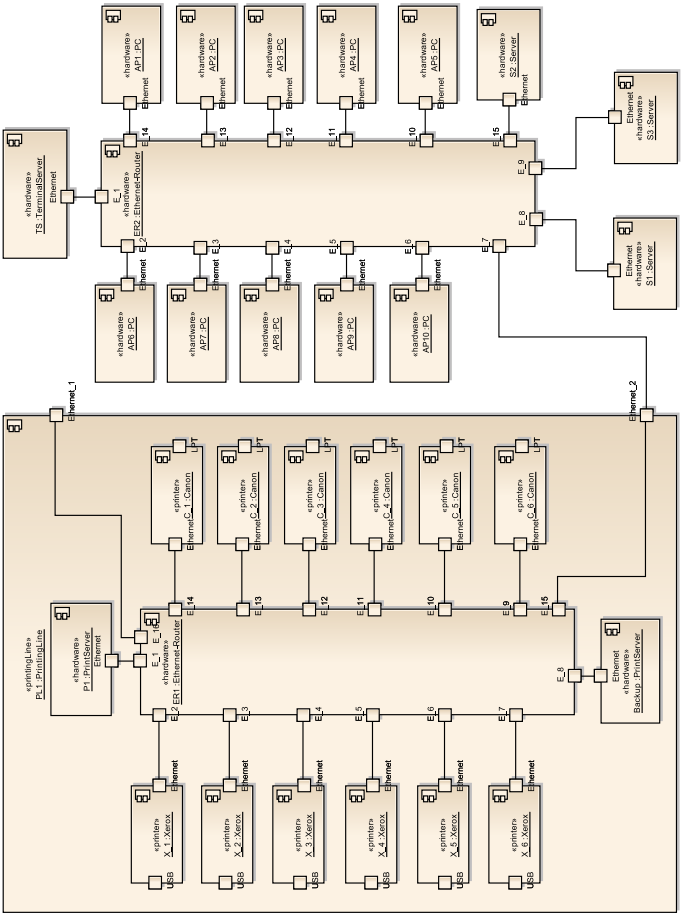


Abbildung 9.6: Ausschnitt aus der Hardware-Sicht des Workflow-Systems

9.3.2 Validierungsziele identifizieren

Gemäß dem in Abbildung 4.3 dargestellten Ablauf des Validierungsprozesses beginnt der Architekt mit dem Identifizieren von Validierungszielen aus der Menge der architekturrelevanten Anforderungen. Aus den Anforderun-

gen in den Tabellen 9.2 und 9.3 identifiziert er die Ziele und ordnet die für das Ziel relevanten Anforderungen zu. Das Ergebnis ist in Tabelle 9.4 zusammengefasst.

Tabelle 9.4: Validierungsziele und zugeordnete Anforderungen für das Workflow-System

VZ-ID	**Name des Validierungsziels**	**zugeordnete Anforderung**	**Prüfkriterium**
WF-V1	Druckkosten A4 (s/w)	ZD10	$\leq 0,04AM$-$
WF-V2	Energieverbrauch Druckstraße	ZD11, ZD15, ZD16, ZD17	$\leq 4KW$
WF-V3	Durchlaufzeit Kundenrechnung	ZD13, ZD15	$\leq 3Tage$
WF-V4	Durchlaufzeit kleine interne Druckaufträge	ZD14, ZD17	$\leq 5min$
WF-V5	Druckqualität	ZD12	entspricht Vorlage

Für das Validierungsziel WF-V2 sind beispielsweise die Anforderungen ZD11, ZD15, ZD16 und ZD17 von Bedeutung. ZD11 gibt die Leistung vor, die das System erbringen muss (max. 4KW Energieverbrauch der Druckstraße). Die Architektur muss diese Vorgabe erfüllen. Das Prüfkriterium ist deshalb $\leq 4KW$. Der Energieverbrauch ist von der Nutzungsintensität abhän-

gig, die in den Anforderungen ZD15 bis ZD17 für verschiedene Workflows vorgegeben werden.

In Bezug auf die Abhängigkeiten der Validierungsziele untereinander hält der Architekt eine Abhängigkeit zwischen WF-V3 und WF-V4 fest. Falls mehr Kundenrechnungen gedruckt werden müssen, steht den internen Druckaufträgen ggf. nicht mehr so viel Druckkapazität zur Verfügung, so dass sich längere Wartezeiten ergeben könnten. Analog verhält es sich, falls mehr interne Druckaufträge bearbeitet werden müssen. Eine weitere Abhängigkeit ist zwischen WF-V3 und WF-V2 sowie zwischen WF-V4 und WF-V2 zu erkennen: Eine Veränderung bei der Anzahl der zu druckenden Dokumente wirkt sich auf den Energieverbrauch aus. Im Gegensatz zur Abhängigkeit zwischen WF-V3 und WF-V4 ergeben sich diese Abhängigkeiten bereits durch die Analyse der zugeordneten Anforderungen. WF-V3 ist die Anforderung ZD15, WF-V4 die Anforderung ZD17 zugeordnet. Sowohl ZD15 als auch ZD17 sind wiederum WF-V2 zugeordnet. Des Weiteren hält der Architekt eine Verbindung zwischen Druckqualität (WF-V5) und Druckkosten (WF-V1) sowie Druckqualität zu den beiden Validierungszielen WF-V3 und WF-V4 fest.

9.3.3 Validierungszielverfahren festlegen

Auf Basis der erarbeiteten Validierungsziele legt der Architekt die Verfahren zur Prüfung der in den Anforderungen geforderten Werte fest.

Druckkosten Für die Druckkosten werden die vom Hersteller angegebenen Kosten für das Bedrucken eines A4-Blatts (s/w) verwendet. Hinzu kommen die Papierkosten.

Energieverbrauch Druckstraße Der Energieverbrauch der Druckstraße wird über die Summe der benötigten Energie der zur Druckstraße gehörenden Drucker ermittelt. Dabei wird zwischen dem Verbrauch im Standby und dem Verbrauch während des Druckens unterschieden. Die entsprechenden Werte werden den Datenblättern der Hersteller entnommen.

Durchlaufzeit Kundenrechnung Für eine Kundenrechnung werden die Transaktionsdaten eines festgelegten Zeitraums berücksichtigt. Dazu wird ein Stichtag, das Kundenabrechnungsdatum, ausgewählt. Ist beispielsweise das Kundenabrechnungsdatum der 15. eines jeden Monats, dann werden dieser Rechnung alle Transaktionsdaten einschließlich des 15. des Vormonats und dem 14. des aktuellen Monats zugeordnet. Die Kundenrechnung muss dann spätestens am 17. verschickt worden sein. Die Durchlaufzeit wird durch die Gesamtbearbeitungszeit der einzelnen Aufgaben des Workflows ermittelt. Die Gesamtbearbeitungszeit setzt sich aus der reinen Bearbeitungszeit (egal ob durch Mensch oder Maschine) und der ggf. anfallenden Wartezeit zusammen.

Durchlaufzeit kleine interne Druckaufträge Die Durchlaufzeit für interne Druckaufträge beginnt mit der Annahme des Druckauftrages und endet mit der Bereitstellung der Druckerzeugnisse. Sie setzt sich aus Wartezeiten, der benötigten Zeit zum Drucken sowie der Annahme und Bereitstellung zusammen.

Druckqualität Die Druckqualität wird durch den stichprobenartigen Vergleich von Druckerzeugnissen mit einer dazu passenden Vorlage überprüft. Dies ließe sich grundsätzlich mit einem gewissen technischen Aufwand auch automatisieren, die Kontrolle durch einen Menschen erscheint an dieser Stelle jedoch sinnvoller. Aus diesem Grund wird dieser Aspekt in den

nachfolgenden Schritten des Validierungsprozesses nicht weiter betrachtet.

9.3.4 Validierungsspezifische Notation erstellen

Mit den festgelegten Validierungszielverfahren kennt der Architekt alle benötigten Daten, mit denen er die Validierung der Architektur gegenüber den Anforderungen der ausgesuchten Validierungsziele durchführen kann. Als Datenquelle für die Verfahren dient das Validierungsmodell. Falls sich nicht alle Daten mit Hilfe vorhandener UML-Notationselemente modellieren lassen, erweitert der Architekt die UML mit Hilfe des Profilmechanismus. Diese validierungsspezifische Notation (VSN) ist in Abbildung 9.7 dargestellt.

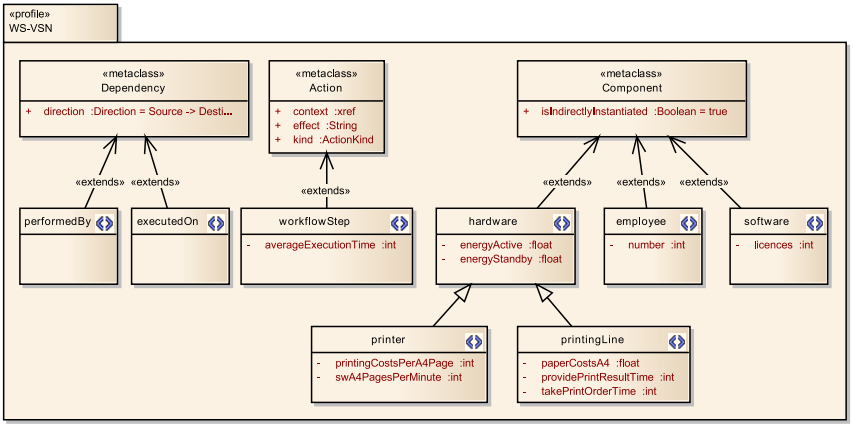


Abbildung 9.7: Validierungsspezifische Notation für das Workflow-System

Sie enthält einen Stereotyp *printer*, der vom Stereotyp *hardware* erbt. Die Tagged-Values *energyStandby* und *energyActive* stellen die Daten für die Ermittlung des Energieverbrauchs der Druckstraße bereit. Der Wert *swA4PagesPerMinute* gibt die Anzahl der A4-Seiten an, die der Drucker

innerhalb von einer Minute im Modus schwarz/weiß drucken kann. Die Druckkosten für eine A4-Seite gibt der Tagged-Value *printingCostsPerA4Page* an. Der Stereotyp *printingLine* erbt wie schon der Stereotyp *printer* von *hardware*. Er stellt mit den Tagged-Values *takePrintOrderTime* und *providePrintResultTime* die Zeitwerte für die Annahme von Druckaufträgen sowie die Bereitstellung der Druckerzeugnisse bereit. Der Wert *paperCostsA4* legt die Kosten für ein A4-Blatt innerhalb dieser Druckstraße fest.

Die einzelnen Arbeitsschritte der Workflows werden mit dem Stereotyp *workflowStep* versehen. Mit dem Tagged-Value *averageExecutionTime* wird die durchschnittliche Bearbeitungszeit für diese Aufgabe in Sekunden angegeben. Die einzelnen Workflow-Schritte werden entweder einer Software-Komponente oder einem Mitarbeiter zugeordnet. Die Zuordnung ist grafisch anhand der Position der Workflow-Schritte in den Swimlanes zu erkennen. Im Modell erfolgt die Zuordnung von der Aktion zu einer Software-Komponente bzw. einer Mitarbeiter-Komponente durch eine Abhängigkeitsbeziehung mit dem Stereotyp *performedBy*. Die Mitarbeiter-Komponente (Stereotyp *employee*) repräsentiert einen Pool von gleichwertig qualifizierten Mitarbeitern, die für die Abarbeitung der Workflow-Schritte zur Verfügung stehen. Die Anzahl wird über den Tagged-Value *amount* angegeben.

9.3.5 Validierungsmodell erstellen

Mit Hilfe der VSN kann der Architekt das Validierungsmodell erstellen und mit den für die Validierung erforderlichen Daten anreichern. Falls noch keins existiert, transformiert er unter Verwendung der VSN alle aus dem Entwicklungsmodell benötigten Daten in ein neues Modell und ergänzt anschließend die validierungsspezifischen Daten. Abbildung 9.8 zeigt die Zuordnung der WorkflowStep-Aktionen zu den Software-Komponenten bzw.

den Mitarbeiter-Komponenten, in Abbildung 9.9 ist die die Deployment-Sicht im Validierungsmodell visualisiert.

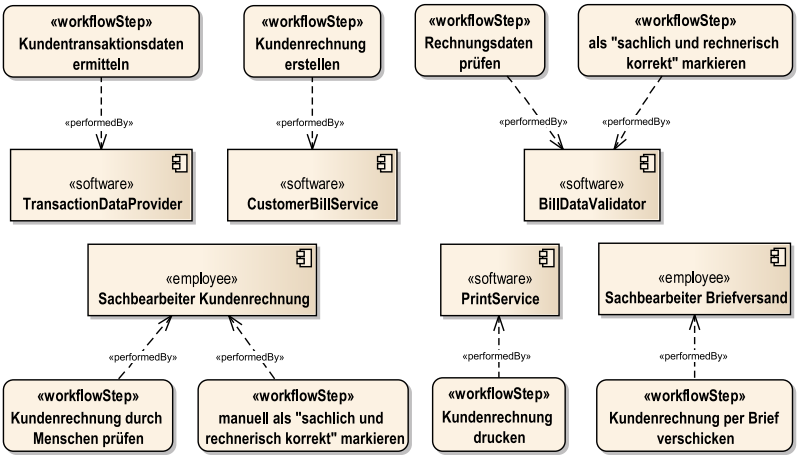


Abbildung 9.8: Zuordnung der WorkflowStep-Aktionen zu System-Komponenten im Validierungsmodell

Welche Daten aus dem Entwicklungsmodell übernommen und welche ergänzt wurden, wird am Beispiel der WorkflowStep-Aktionen erläutert. Die Aktion selbst, die Verbindung zu anderen Aktionen sowie die Zuordnung zur Swimlane werden aus dem Entwicklungsmodell übernommen. Dabei findet eine Transformation von Aktion auf WorkflowStep-Aktion statt. Der Architekt ergänzt dann noch die durchschnittliche Bearbeitungszeit (Tagged-Value *averageExecutionTime*). Falls in dem Modell eine Komponente existiert, die namensgleich mit der zur Aktion gehörenden Swimlane ist, wird eine *PerformedBy*-Beziehung zwischen der Aktion und der Komponente erstellt. Falls keine entsprechende Komponente vorhanden ist, muss der Architekt selber diese Daten ergänzen.

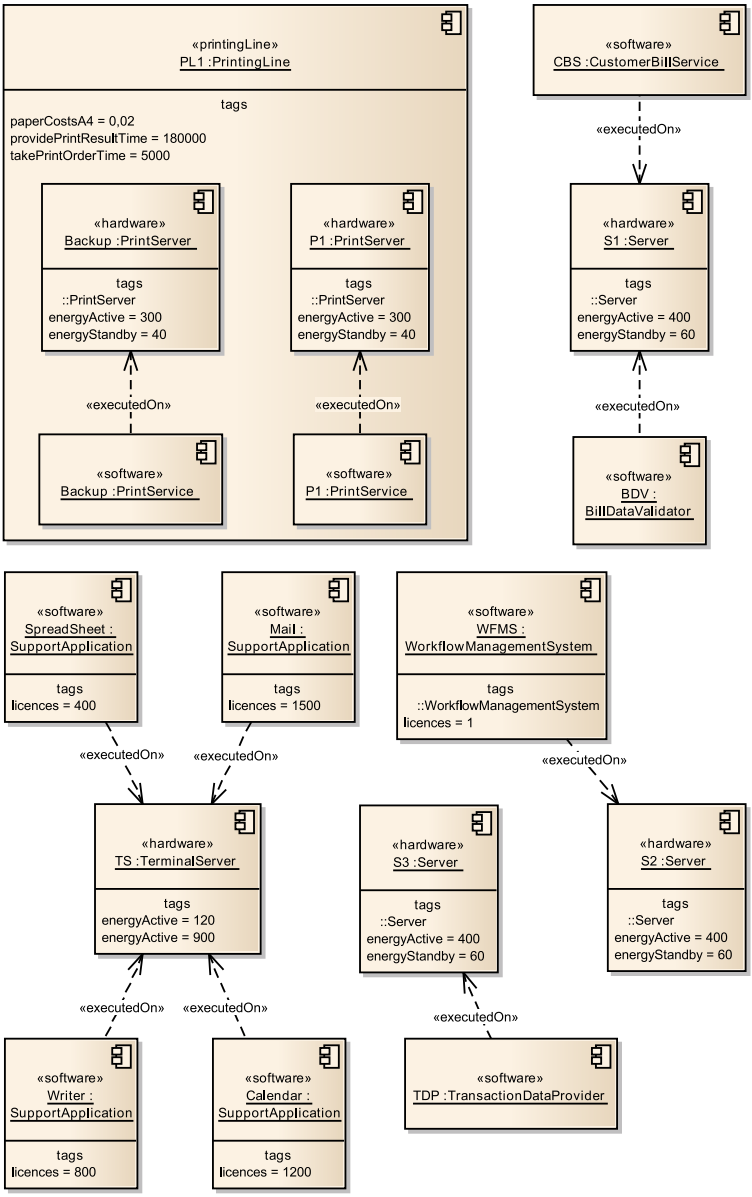


Abbildung 9.9: Deployment-Sicht des Workflow-Systems im Validierungsmodell

9.3.6 Validierung durchführen

Nachdem der Architekt die Validierungszielverfahren festgelegt und alle dafür erforderlichen Daten im Validierungsmodell dokumentiert hat, kann die Validierung der Workflow-System-Architektur bzgl. der ausgewählten Ziele erfolgen. Die Resultate sind in Tabelle 9.5 zusammengefasst[159]. Alle Validierungsziele werden erfüllt, weshalb auch alle Zellen in der Spalte Validierungsergebnisse grün hinterlegt sind.

Tabelle 9.5: Validierungsergebnisse

VZ-ID	VZ-Name	Prüfkriterium	Validierungsergebnis
WF-V1	Druckkosten A4 (s/w)	$\leq 0,04AM$-$	0,03AM-$
WF-V2	Energieverbrauch Druckstraße	$\leq 4KW$	3,2KW
WF-V3	Durchlaufzeit Kundenrechnung	$\leq 3Tage$	2,2 Tage
WF-V4	Durchlaufzeit kleine interne Druckaufträge	$\leq 5min$	3,6min

9.3.7 Impact-Analyse nach Änderung an den Anforderungen

Für die von der Unternehmensleitung angestrebte Wachstumsstrategie soll die Anzahl der Kunden gesteigert werden. Als Zielgröße werden 900.000 Kunden anvisiert. Durch die höhere Kundenanzahl wird auch von einer Erhöhung der Kundenanfragen auf 11.000 pro Monat ausgegangen. Für den

[159] Die Ergebnisse sind zum Teil einer Simulation entnommen. Auf genauere Details wird an dieser Stelle nicht eingegangen, da dies für den Nachweis der Übertragbarkeit nicht von Bedeutung ist.

Architekten stellt sich die Frage, ob das aktuelle Workflow-System diesen veränderten Anforderungen gerecht wird und falls nicht, was die Flaschenhälse im System sind.

Tabelle 9.6: Validierungsergebnisse nach Änderungen an den Anforderungen ZD15 und ZD16

VZ-ID	**VZ-Name**	**Prüfkriterium**	**Validierungsergebnis**
WF-V1	Druckkosten A4 (s/w)	$\leq 0,04AM$-$	0,03AM-$
WF-V2	Energieverbrauch Druckstraße	$\leq 4KW$	3,6KW
WF-V3	Durchlaufzeit Kundenrechnung	$\leq 3Tage$	2,4 Tage
WF-V4	Durchlaufzeit kleine interne Druckaufträge	$\leq 5min$	3,7min

Die Änderungen wirken sich auf die Anforderungen ZD15 und ZD16 aus. Anhand der Zuordnung der für die Validierungsziele relevanten Anforderungen weiß der Architekt, dass dies Auswirkungen auf den Energieverbrauch der Druckstraße (WF-V2) und die Durchlaufzeit der Kundenrechnung (WF-V3) hat. Durch die Abhängigkeitsbeziehungen zwischen den Validierungszielen weiß er, dass Auswirkungen auf WF-V3 auch Auswirkungen auf WF-V4 zur Folge haben können. Wie diese Auswirkungen im Detail aussehen, ergibt sich erst nach der Durchführung der Validierungszielverfahren. Die Resultate sind in Tabelle 9.6 enthalten. Die Validierungsergebnisse für WF-V2, WF-V3 und WF-V4 verändern sich, bleiben allerdings innerhalb der Vorgaben.

9.3.8 Impact-Analyse nach Änderung an den Prüfkriterien

Im Rahmen einer Gesetzesänderung werden vorhandene Subventionen bei den Energiepreisen für Unternehmen gestrichen. Die Unternehmensleitung möchte deshalb den Energieverbrauch im gesamten Unternehmen um 20% reduzieren. Von dieser Änderung ist die Anforderung ZD11 betroffen. Hier ändert sich der vorgebene Maximalwert für den Energieverbrauch von 4KW auf 3,2KW. Nach Änderungen an Prüfkriterien sieht der Validierungsprozess die Durchführung der Validierungszielverfahren vor. Das Validierungsergebnis für WF-V2 lautet 3,6KW (siehe auch Tabelle 9.7). Für den Architekten besteht also Handlungsbedarf.

Tabelle 9.7: Validierungsergebnisse nach Änderung des Prüfkriteriums für das Validierungsziel WF-V2

VZ-ID	**VZ-Name**	**Prüfkriterium**	**Validierungs-ergebnis**
WF-V1	Druckkosten A4 (s/w)	$\leq 0,04AM$-$	0,03AM-$
WF-V2	Energieverbrauch Druckstraße	$\leq 3.2KW$	3,6KW
WF-V3	Durchlaufzeit Kundenrechnung	$\leq 3Tage$	2,4 Tage
WF-V4	Durchlaufzeit kleine interne Druckaufträge	$\leq 5min$	3,7min

Er tauscht im Validierungsmodell einige Druckergeräte gegen energiesparsamere Geräte aus (siehe Abbildung 9.10). Dadurch ändern sich neben den Werten für den Energieverbrauch (Tagged-Values *energyStandby* und *energyActive*) auch die Werte für die Druckkosten (Tagged-

Value *printingCostsPerA4Page*) und die Druckgeschwindigkeit (Tagged-Value *swA4PagesPerMinute*). Wie genau sich diese Änderungen auswirken, erfährt der Architekt nach dem Ausführen der Validierungszielverfahren. Die Ergebnisse sind in Tabelle 9.8 zu sehen. Der Energieverbrauch ist auf 3,2KW gesunken und entspricht damit genau den Vorgaben. Mit dem geringeren Energieverbrauch gehen eine langsamere Druckgeschwindigkeit und höhere Druckkosten einher. Die Validierungsergebnisse sind aber alle noch im Rahmen der Vorgaben.

Tabelle 9.8: Validierungsergebnisse nach dem Austausch von Druckgeräten

VZ-ID	**VZ-Name**	**Prüfkriterium**	**Validierungs-ergebnis**
WF-V1	Druckkosten A4 (s/w)	$\leq 0,04AM$-$	0,035AM-$
WF-V2	Energieverbrauch Druckstraße	$\leq 3.2KW$	3,2KW
WF-V3	Durchlaufzeit Kundenrechnung	$\leq 3Tage$	2,6 Tage
WF-V4	Durchlaufzeit kleine interne Druckaufträge	$\leq 5min$	3,9min

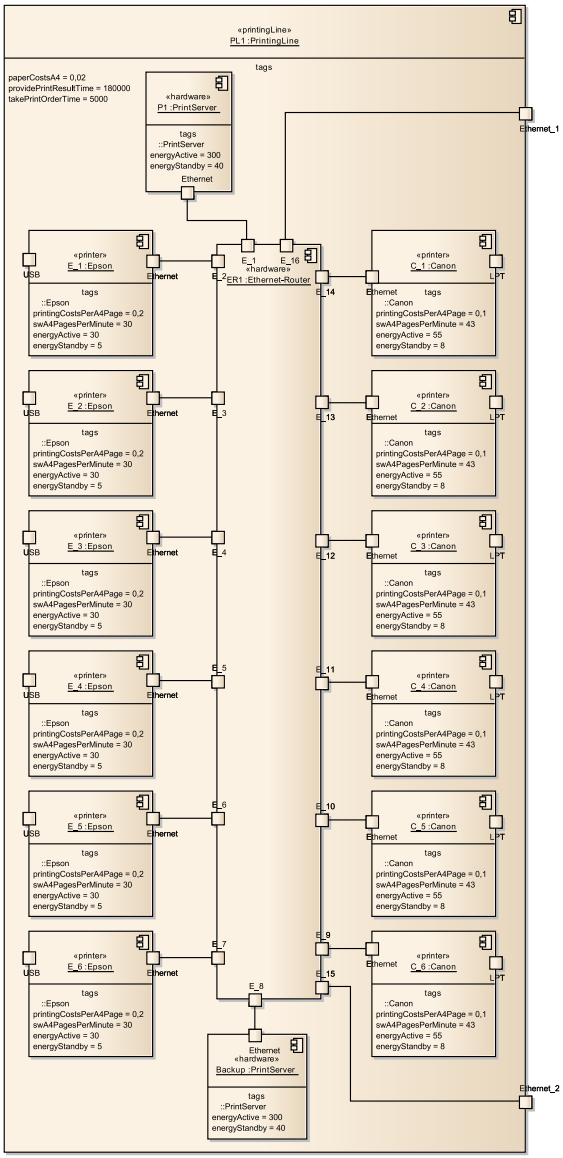


Abbildung 9.10: Ausschnitt aus der Hardware-Sicht im Validierungsmodell nach dem Tausch von Druckgeräten

In diesem Kapitel wurde die Übertragbarkeit des in der Arbeit vorgestellten Ansatzes auf eine organisatorisch geprägte Domäne nachgewiesen. Der Validierungsprozess wird zur Prüfung der Architektur oder Konfiguration eines Workflow-Systems angewendet. Dies geht über die Prüfung der Workflow-Modelle hinaus und betrachtet die Abhängigkeiten der zur Verfügung stehenden Ressourcen wie Software-Lizenzen, Server oder Bearbeiter. Auch das Identifizieren von Engpässen ist dem Architekten mit Hilfe der von den Simulationen bereitgestellten Details möglich. Die Anwendung des Ansatzes ist sowohl für Neuentwicklungen als auch für existierende Workflow-Systeme denkbar. Dabei muss nicht immer das vollständige System betrachtet werden, eine Überprüfung von Teilsystemen ist ebenfalls möglich. Die Festlegung auf die UML als Modellierungssprache stellt auch in Domänen mit anderen Notationssprachen (hier: BPMN) grundsätzlich kein Hindernis für den Einsatz des Validierungsprozesses dar.

Im nachfolgenden Kapitel wird der in der Arbeit vorgestellte Ansatz bewertet, indem die Ergebnisse von Erprobungen vorgestellt werden und anschließend eine kritische Betrachtung des Validierungsprozesses erfolgt.

10 Bewertung des Ansatzes

Dieses Kapitel 10 nimmt eine Bewertung des in der Arbeit vorgestellten Validierungsprozesses vor. Hierzu stellt Kapitel 10.1 die Resultate von praktischen Erprobungen des Ansatzes vor. Anhand der Ergebnisse lässt sich sagen, dass der Validierungsprozess sowohl von erfahrenen als auch weniger erfahrenen System-Architekten verwendet werden kann. Er ist in relativ kurzer Zeit erlern- und anwendbar. Die Qualität der Validierungsergebnisse hängt allerdings im Wesentlichen von der Erfahrung des Architekten, den verwendeten Verfahren und den zur Verfügung stehenden Daten ab. Kapitel 10.2 geht hierauf in Form einer kritischen Diskussion näher ein.

10.1 Nachweis der praktischen Anwendbarkeit

Der Validierungsprozess wurde anhand von zwei konkreten Projekten entwickelt und auf diese angewendet. Dadurch konnten einige Unterstützungsmöglichkeiten für den Architekten herausgearbeitet werden, welche die Benutzbarkeit des Prozesses verbessert haben. Das Ergebnis ist in Kapitel 7.1 beschrieben und seine Realisierbarkeit wurde durch einen Werkzeugprototypen belegt. Um die Anwendbarkeit des Validierungsprozesses auch für Personen, die nicht an der Entwicklung des Prozesses beteiligt waren, zu belegen, wurden zwei Erprobungen in Form von Workshops mit unterschiedlichen Testgruppen durchgeführt. Bei der ersten Testgruppe handelt es sich um erfahrene Berater des Forschungspartners SOPHIST GmbH, die zweite Testgruppe besteht aus Masterabsolventen der Westsächsischen Hochschule Zwickau (WHZ). Die Details sind in den Kapiteln 10.1.1 und 10.1.2 beschrieben, Kapitel 10.1.3 wertet die Ergebnisse der Erprobung aus. Zusammenfassend lässt sich sagen, dass nicht nur die erfahrenen Berater, sondern

auch die Masterabsolventen den Validierungsprozess nach dem Workshop anwenden konnten. Der zeitliche Aufwand zur Wissensvermittlung war bei den Masterabsolventen erwartungsgemäß etwas höher. Auf Basis der Ergebnisse wird die Hypothese aufgestellt, dass für die Anwendbarkeit des Ansatzes ein Schulungsaufwand von 1-2 Tagen für erfahrene und von 2-3 Tage für weniger erfahrene Systementwickler notwendig ist.

10.1.1 Testgruppe 1

Die erste Testgruppe bestand aus sechs Mitarbeitern der SOPHIST GmbH. Sie alle sind seit mindestens 10 Jahren in verschiedenen Kundenprojekten als Berater aktiv. Sie besitzen fundierte und detaillierte Kenntnisse über das Requirements-Engineering, die modellbasierte Dokumentation mit der UML sowie über die Analyse und das Design von Systemen. Die Dauer des Workshops betrug fünf Stunden und beinhaltete eine kurze thematische Einführung und Motivation, die Vorstellung des Validierungsprozesses und eine praktische Durchführung anhand eines Beispiels. Dabei wurde auch der Werkzeugprototyp eingesetzt. Die Teilnehmer hatten keine Vorkenntnisse zum Validierungsprozess, ihnen wurde vorher lediglich ein thematischer Überblick zur Verfügung gestellt. Aufgrund der vorhandenen Erfahrung konnte die Einführung und Motivation sehr kurz gehalten werden. Die theoretische Vorstellung des Prozesses erfolgte anhand des vereinfachten Aktivitätsdiagramms in Abbildung 4.3 auf Seite 70 und anhand von kleineren Beispielen. Die Anwendung des Prozesses auf das Beispiel hat den größten Teil der Zeit in Anspruch genommen. Die Prozessschritte wurden überwiegend selbstständig und ohne nennenswerte Unterstützung durchgeführt. Die Teilnehmer fühlten sich nach dem Workshop in der Lage, den Validierungsprozess an einem weiteren Beispiel ohne Unterstützung durchzuführen.

10.1.2 Testgruppe 2

Die zweite Testgruppe bestand aus fünf Masterabsolventen der WHZ. Im Gegensatz zur ersten Testgruppe haben sie nur wenig bis keine praktische Erfahrung mit der Entwicklung von Systemen. Sie verfügen aber über fundiertes Wissen aus den Bereichen UML und modellgetriebene Software-Entwicklung. Der Workshop hat zeitlich nach dem der ersten Testgruppe stattgefunden. Aufgrund der geringeren Vorkenntnisse aus den Bereichen der System-Analyse und System-Design wurde mehr Zeit für die Motivation und fachliche Einführung eingeplant. Die Dauer des Workshops wurde auf sieben Stunden festgelegt. Die Teilnehmer hatten wie schon bei der ersten Testgruppe keine Vorkenntnisse und haben vorab nur einen thematischen Überblick zum Validierungsprozess erhalten. Der Ablauf des Workshops entsprach in etwa dem der ersten Testgruppe, wobei mehr Zeit für Fragen im Bereich der Systemmodellierung erforderlich war. Die Anwendung des Validierungsprozesses auf das Beispiel erfolgte im Vergleich mit der ersten Testgruppe weniger selbstständig, was vor allem mit der zeitlichen Begrenzung zu begründen ist. Die Teilnehmer fühlten sich nach dem Workshop grundsätzlich in der Lage, den Validierungsprozess auf ein weiteres Beispiel anzuwenden, eine geführte Anwendung auf ein zweites Beispiel wäre aber zur Vertiefung von allen Teilnehmern bevorzugt worden.

10.1.3 Auswertung der Ergebnisse

Die durchgeführten Erprobungen des in der Arbeit vorgestellten Ansatzes zur Validierung von System-Architekturen werden als positiv gewertet. Die Anwendung des Ansatzes durch erfahrene Systementwickler ist gelungen. In Anbetracht der Tatsache, dass ein Masterabsolvent ohne nennenswerte praktische Erfahrung im Bereich der Systementwicklung eher selten direkt

die Rolle eines System-Architekten in einem Projekt erhält, ist auch die zweite Erprobung positiv zu bewerten. Mit einem höheren zeitlichen Aufwand für die fachliche und thematische Einarbeitung ließe sich ein zur ersten Testgruppe vergleichbares Ergebnis erzielen. In beiden Erprobungen gab es keine grundlegenden Verständnisprobleme bei der Durchführung des Ansatzes. Die Fragen waren überwiegend auf Details konzentriert. Die Durchführung von zwei Erprobungen lässt natürlich keine allgemeinen Aussagen über die Anwendbarkeit des Ansatzes zu. Es lassen sich aber zwei Hypothesen formulieren:

1. Der Ansatz lässt sich innerhalb von 1-2 Tagen[160] von erfahrenen Systementwicklern erlernen.
2. Für Anwender mit geringer Erfahrung im Bereich der Systementwicklung sind 2-3 Tage zum Erlernen des Ansatzes erforderlich.

10.2 Kritische Betrachtung des Validierungsprozesses

Nach der erfolgreichen Durchführung des Validierungsprozesses existiert eine System-Architektur, die gegenüber den betrachteten architekturrelevanten System-Anforderungen valide ist. Die Güte des Validierungsergebnisses hängt allerdings von verschiedenen Faktoren ab, die der Architekt bei der Analyse der Validierungsergebnisse berücksichtigen sollte.

Die Anforderungsermittlung, insbesondere die der nicht-funktionalen Anforderungen, ist nicht Bestandteil des Validierungsprozesses. Wie in Kapitel 4.2 beschrieben, setzt der Prozess eine Menge ermittelter Anforderun-

[160] Die Dauer ist vom Erfahrungsgrad der Systementwickler abhängig.

gen voraus, deren Vollständigkeitsgrad ausreicht, einen (partiellen) Design-Entwurf zu validieren. Die Ermittlung einer vollständigen Menge nicht-funktionaler Anforderungen ist in der praktischen Umsetzung unmöglich. Zwar existieren für diese Problemstellung verschiedene Ansätze [Dör10] [HP08], die Vollständigkeit der Menge nicht-funktionaler Anforderungen basiert dabei aber stets auf Annahmen, die in der praktischen Anwendung nicht garantiert werden können[161]. Außerdem unterscheiden sich die Begriffsdefinitionen für eine nicht-funktionale Anforderung (siehe beispielsweise Kapitel 2.4.2 und 2.4.3), so dass je nach Definitionswahl nicht alle nicht-funktionalen Anforderungen durch die Verfahren ermittelt werden können[162]. Dem Architekten muss deshalb stets bewusst sein, dass der Validierungsprozess[163] nur die Validität der Architektur gegenüber den ermittelten Anforderungen sicherstellen kann. Diese Einschränkung gilt aber nicht nur für den vorgestellten Validierungsprozess, sondern für jeden anderen Ansatz zur Konformitätsprüfung von Architekturen.

Das Validierungsergebnis kann durch Fehler innerhalb des Validierungsprozesses verfälscht[164] werden. Im Prozessschritt *Validierungsziele identifizieren* (siehe Abbildung 4.3) ermittelt der Architekt anhand der Anforderungen die Validierungsziele und stellt Verbindungen zwischen diesen her. Die

[161] Ein Beispiel hierfür ist eine nicht-funktionale Anforderung, dessen Existenz den Stakeholdern nicht bewusst ist. Zwar gibt es verschiedene Techniken zur Ermittlung unbewusster Anforderungen, eine Garantie für die Ermittlung können diese Techniken aber nicht geben; der Faktor Mensch verhindert dies.

[162] Als Beispiel können hier die Projektrandbedingungen genannt werden, die für Rupp zu den nicht-funktionalen Anforderungen zählen, Dörr hingegen ordnet sie den organisatorischen Anforderungen zu. Der Ansatz von Dörr sieht die von seinem Verfahren behandelten nicht-funktionalen Anforderungen als Teil der Produktanforderungen, die auf einer Ebene neben den organisatorischen Anforderungen stehen.

[163] Eine korrekte Durchführung des Prozesses wird an dieser Stelle vorausgesetzt.

[164] Unter verfälscht wird an dieser Stelle jede Abweichung vom erwarteten Ergebnis verstanden. Dies kann von einem veränderten Ergebnis eines Validierungszielverfahrens bis hin zu veränderten Randbedingungen für die Gültigkeit des Validierungsergebnisses, beispielsweise dem Detaillierungsgrad eines Validierungszielverfahrens, reichen.

Validierungsziele und deren Verbindung mit den NFAs bilden das Datenfundament für das Verfahren. Auf Basis der Validierungsziele und den zugeordneten Anforderungen werden die Validierungszielverfahren ausgewählt. Vergisst der Architekt, Anforderungen einem Validierungsziel zuzuordnen, hat dies ggf. Auswirkungen auf das gewählte Validierungszielverfahren. Ist eine Anforderung mit mehreren Validierungszielen verbunden, beschränken sich die Auswirkungen nicht auf ein Validierungsziel. Über die Verbindungen der Anforderungen zu den Validierungszielen kann der Architekt Abhängigkeiten zwischen verschiedenen architekturspezifischen Aspekten erkennen. Falls diese nicht bei der Erstellung der Validierungszielverfahren berücksichtigt werden, ist das Verfahren ggf. für die Validierung des Ziels ungeeignet und das Ergebnis sollte nicht für die Validierung des Ziels verwendet werden.

Das Ergebnis eines Validierungszielverfahrens hängt zum einen von den Eingangsdaten und zum anderen vom gewählten Algorithmus ab. Es ist die Aufgabe des Architekten, für die Algorithmen passende Eingangsdaten bzw. für die vorhandenen Eingangsdaten geeignete Algorithmen bereitzustellen. Welche Güte die Validierungsergebnisse haben, hängt von der Kombination ab. Liegen detaillierte Eingangsdaten bereit und die Entwicklung der Algorithmen ist Teil der Systementwicklung, kann der Architekt zunächst vereinfachte Algorithmen verwenden und diese mit fortschreitender Entwicklung anpassen. Die Güte der Validierungsergebnisse steigt dann mit der Systementwicklung. Ähnlich verhält es sich im umgekehrten Fall: detaillierte Algorithmen wirken sich nur dann positiv auf die Güte der Validierungsergebnisse aus, wenn die Eingangsdaten zu ihnen passen. Liegen zu Beginn der Entwicklung wenig detaillierte Daten vor, ist die Aussagekraft der Ergebnisse nicht so hoch wie der Detaillierungsgrad der Algorithmen es zulassen

würde. Erst wenn die passenden Eingangsdaten vorhanden sind, verbessert sich die Güte der Validierungsergebnisse.

Bei Systemen lassen sich die Eingangsdaten in fachliche und technische Daten unterteilen. Zu der Menge der technischen Daten zählen hier alle Werte physikalischer Bausteine wie beispielsweise die Bandbreite eines Kommunikationsmediums oder die Leistung einer Verarbeitungseinheit. Der Detaillierungsgrad dieser Daten kann von Schätzungen, theoretischen Kennwerten bis zu empirisch ermittelten Daten variieren[165]. In langlaufenden Projekten stehen diese zum Zeitpunkt des Systemdesigns nur selten vollständig und bezüglich des Detaillierungsgrads einheitlich zur Verfügung, da sich der Hardware-Markt kontinuierlich weiterentwickelt und eine frühe Festlegung auf bestimmte Bauteile vermieden wird, um beispielsweise Preisvorteile oder eine höhere Leistung nutzen zu können. Hier wird häufig mit Schätzungen/Erfahrungen und theoretischen Kennwerten gearbeitet. Zur besseren Einordnung der Ergebnisse kann die Durchführung einer Fehlerrechnung sinnvoll sein. Dazu gibt der Architekt für geschätzte oder empirisch ermittelte Werte eine Abweichung an. Mit Hilfe der Fehlerfortpflanzungsberechnung, die sich durch die verarbeitenden Algorithmen ergibt, lässt sich eine Abweichung für das Validierungsergebnis ermitteln. Auf diese Weise kann der Architekt auch bei Eingangsdaten mit sehr unterschiedlichen Abweichungen die Güte seines Validierungsergebnisses beurteilen und ggf. Eingabedaten mit geringeren Abweichungen oder andere Algorithmen verwenden[166].

[165] Beispiel High-Speed-Kommunikationsschnittstelle: hoffnungsvoll geschätzt sind: 5.000 Mibit/s, 4.200 Mibit/s laut Datenblatt und empirisch gemessen sind es 3.000 Mibit/s. Ein gutes Beispiel sind auch die Wireless Local Area Network (WLAN)-Bandbreiten in der Theorie und im praktischen Einsatz.

[166] Für weitere Informationen zum Thema Fehlerrechnung und Fehlerfortpflanzung wird auf [Pap11] und [WE09] verwiesen.

Bei datenintensiven Systemen ist die Bereitstellung aller denkbaren fachlichen Eingangsdaten nicht immer möglich. Hier muss der Architekt, am besten zusammen mit dem Kunden, eine Auswahl treffen, mit der die Systemabnahme und damit auch die Architekturvalidierung durchgeführt werden soll. Gerade in frühen Projektphasen mit sich häufig ändernden Anforderungen ist dies eine schwierige Aufgabe. Auch hier gilt: Die Validierungsergebnisse sind nur für die verwendeten fachlichen Daten gültig. Bereits minimale Änderungen können vollständig andere Validierungsergebnisse hervorbringen.

Der Detaillierungsgrad der Daten und Algorithmen muss nicht für die gesamte Architektur gleich sein. Falls der Architekt nur von ihm als kritisch eingestufte Bereiche der Architektur validieren möchte, können andere Teile einen geringeren Detaillierungsgrad aufweisen. Die Detaillierung kann so weit gehen, dass die Validierungszielverfahren den realen Algorithmen entsprechen. Eine andere Möglichkeit ist die Verwendung detaillierter, empirisch ermittelter Werte als Eingangsdaten für die Validierung. Der Architekt sollte allerdings immer das Verhältnis von Aufwand und Nutzen im Auge behalten. Ein hoher Detaillierungsgrad für einen bestimmten Teil der Architektur kann nur selten ohne die Detaillierung anderer Bereiche erreicht werden.

Zusammenfassend lässt sich sagen, dass der vorgestellte Validierungsprozess die Konformität der modellierten Architektur gegenüber den im Prozess verwendeten Anforderungen feststellen kann. Architektonische Fehler bzw. Fehler in kritischen Teilen der Architektur können auf diese Weise auch in einer frühen Projektphase festgstellt werden. Genau hier liegt auch der anvisierte Nutzungszeitraum des Validierungsprozesses: die Erstellung der System-Architektur. Der Einsatz des Prozesses ist auch nach der grund-

sätzlichen Festlegung auf die System-Architektur möglich, jedoch sollte der Validierungsprozess nicht mit einem Prototypen verwechselt werden. Detaillierte Aussagen über die Validität der realen System-Architektur sind nur möglich, wenn die Eingangsdaten und Algorithmen dem Fortschritt der Systementwicklung angepasst werden (können). Hier sollte der Architekt Aufwand und Nutzen sorgfältig analysieren. Die Flexibilität bei der Algorithmenwahl ist allerdings bei keinem anderen Ansatz wiederzufinden.

11 Fazit

Eine der großen Herausforderungen in der Systementwicklung ist der Umgang mit den sich ändernden Anforderungen. Die Ursachen für diesen stetigen Wandel sind zum einen die immer kürzer werdenden Zeiträume von der Idee bis zum am Markt etablierten Produkt und zum anderen die kontinuierliche Weiterentwicklung des Wissens über das zu entwickelnde System. Dieser stetige Wandel findet immer statt, weshalb der Entwicklungsprozess und die eingesetzten Methoden ihn berücksichtigen sollten.

Der Umgang mit sich ständig ändernden Anforderungen stellt insbesondere bei der Erstellung der System-Architektur eine Herausforderung dar. Die System-Architektur bildet das Fundament eines Systems und lässt sich in den nachfolgenden Entwicklungsphasen meist nur mit einem erhöhten Ressourcenaufwand ändern. Der Architekt hat die Pflicht, eine zu den Anforderungen konforme Architktur zu entwerfen und diese Konformität nach Anforderungsänderungen erneut zu prüfen. Diese Validierungsaufgabe findet in der Praxis entweder nur auf dem Papier oder als einmaliges Ereignis am Ende der Design-Phase statt. Dabei wird häufig die informelle Prüftechnik Review verwendet, die zeitaufwändig ist, auf subjektiven Einschätzungen beruht und keine reproduzierbaren Ergebnisse erzeugt. Solch eine Technik ist für eine sich wiederholende, zeit- und kostenintensive Aufgabe wie die Architekturvalidierung insbesondere bei sich ständig ändernden Anforderungen ungeeignet. Eine Alternative stellen die formalen Prüftechniken wie z. B. das Model-Checking dar. Sie werden allerdings in der Praxis außerhalb von sicherheitskritischen System wegen ihrer geringen Akzeptanz nur selten eingesetzt.

Um den Architekten zu unterstützen, definiert der in der vorliegenden Arbeit vorgestellte Validierungsprozess ein systematisches Vorgehen für die Konformitätsprüfung einer System-Architektur gegenüber den System-Anforderungen (siehe Kapitel 4). Er setzt dafür auf eine dynamische Prüftechnik: die Simulation. Mit ihr lassen sich zuverlässig reproduzierbare Ergebnisse erzeugen, was wichtig für wiederholend stattfindende Aufgaben ist. Als Datenquelle für die Simulationen dient eine modellbasierte Dokumentation aller für die Validierung relevanter Daten. Als Dokumentationssprache wird der Quasi-Standard in der Industrie für modellbasierte Dokumentation, die UML, verwendet. Sie ermöglicht das Auslesen von Daten aus dem Modell, wodurch Modelländerungen sich unmittelbar auf die Simulationen und damit auf das Validierungsergebnis auswirken. Der Validierungsprozess unterstützt sowohl die initiale Validierung einer Architektur als auch die erforderliche Revalidierung nach Änderungen an den Anforderungen oder an der Architektur. Er lässt sich in den iterativen Entwurf von Architekturen integrieren und kann für neue oder bereits vorhandene Systeme verwendet werden.

Grundlage für die Architekturvalidierung bilden die sogenannten Validierungsziele. Mit ihnen legt der Architekt fest, welche architekturspzifischen Aspekte überprüft werden sollen. Voneinander abhängige Validierungsziele werden miteinander verbunden. Darüber hinaus werden die Validierungsziele mit allen den architekturspezifischen Aspekt betreffenden Anforderungen verbunden. Dadurch entsteht eine validierungsspezifische Sicht auf die Anforderungen, die es dem Architekten erlaubt, den Überblick zu behalten und direkte sowie indirekte Abhängigkeiten zwischen den Validierungszielen zu erkennen. Sie ist insbesondere bei der Impact-Analyse von Änderungen hilfreich. Die Erfüllung eines Validierungsziels ergibt sich anhand eines Prüfkriteriums, das sich aus den zum Validierungsziel gehörenden Anforde-

rungen ergibt. Ob ein Validierungsziel erfüllt ist, wird durch das dazugehörige Validierungszielverfahren ermittelt. Der Validierungsprozess verwendet für diese Verfahren Simulationen, der Einsatz anderer Prüftechniken ist aber grundsätzlich möglich.

Die Datenquelle für alle Validierungszielverfahren ist das Validierungsmodell. Dabei handelt es sich um ein UML-Modell, das alle zur Durchführung der Verfahren erforderlichen Informationen enthält. Falls eine modellbasierte Dokumentation für die Entwicklung existiert, kann der Architekt dort vorhandene Informationen in das Validierungsmodell übernehmen. Dieses Modell muss nicht zwingend auf der UML basieren (siehe Kapitel 9.3). Durch die Verwendung eines von der Entwicklung getrennten Modells für die Validierung ist eine Reduzierung auf die für die Validierung erforderlichen Modellelemente möglich. Außerdem wird das Entwicklungsmodell nicht mit validierungsspezifischen Informationen gefüllt, die für die Entwicklung nicht weiter von Bedeutung sind. Um alle für die Validierung benötigten Informationen modellieren zu können, verwendet der Architekt eine validierungsspezifische Notation (VSN). Sie wird mit Hilfe des UML-Profilmechanismus erstellt. Wie die VSN erstellt wird und welcher Dokumentationsort für die validierungsrelevanten Informationen verwendet wird, kann der Architekt den projektspezifischen Gegebenheiten anpassen (siehe Kapitel 8.2 und 8.3).

Ob die Validierungsziele erfüllt sind, ergibt sich aus dem Vergleich der Prüfkriterien mit den Ergebnissen der jeweiligen Validierungszielverfahren. Falls alle Validierungsziele erfüllt sind, ist die Architektur valide gegenüber den untersuchten Anforderugen. Falls ein Validierungsziel nicht erfüllt wird, muss der Architekt die Architektur anpassen. Hierzu analysiert er die von den Validierungszielverfahren bereitgestellten Ergebnisse. Simula-

tionen bieten hier gegenüber den formalen Verfahren wie Model-Checking den Vorteil, dass sie detaillierte Zwischenergebnisse bereitstellen können. Diese kann der Architekt für die Ursachenforschung verwenden. Die Anpassung der Architektur erfolgt im Validierungsmodell. Falls die Architekturvalidierung erfolgreich war, werden die für das Entwicklungsmodell relevanten Änderungen vom Validierungsmodell übernommen.

Die Anwendung des Validierungsprozesses wird ausführlich anhand eines Radar-Systems, das auf Basis realer Entwicklungsprojekte erstellt wurde, demonstriert (siehe Kapitel 6). Neben der initialen Erstellung der Konformitätsprüfung werden auch die Auswirkungen möglicher Änderungen anhand von Beispielen beschrieben. Den Nachweis, dass der Validierungsprozess nicht nur für Systeme aus der technischen Domäne beschränkt ist, erbringt Kapitel 9. Der Validierungsprozess wird zur Architekturvalidierung eines Workflow-Systems verwendet, das aus einer eher organisatorisch geprägten Domäne stammt. Auch wenn das Beispiel nicht dem Umfang des Radar-Systems entspricht, konnten bei der Adaption des Ansatzes keine nennenswerten Probleme festgestellt werden. Auf die Erprobungen des Validierungsprozesses mit Personen, die nicht an der Entwicklung beteiligt waren, geht Kapitel 10.1 ein. Die Ergebnisse lassen die Vermutung zu, dass der Ansatz verständlich ist und innerhalb weniger Tage vermittelt werden kann.

Die Konformitätsprüfung der Architektur gegenüber den Anforderungen ist eine zeit- und kostenintensive Aufgabe des Architekten Sie muss zudem bei jeder Änderung an den Anforderungen oder der Architektur erneut durchgeführt werden. Der Validierungsprozess unterstützt den Architekten nicht nur durch die Definition eines systematischen Vorgehens, er bietet auch Möglichkeiten zur teilautomatisierten Durchführung an. Durch den Prozess ergeben sich einige Schritte, die vorgehensspezifisch sind. Hierunter fällt

die Erstellung der VSN, die Erstellung und Pflege des Validierungsmodells und die Übertragung potentieller entwicklungsrelevanter Informationen aus dem Validierungsmodell in das Entwicklungsmodell. Bis auf die Erstellung der VSN lassen sich diese Schritte durch Modelltransformationen automatisieren und fallen für den Architekten deswegen nicht weiter ins Gewicht. Dem einmaligen Aufwand für die Erstellung der VSN stehen Automatisierungsmöglichkeiten gegenüber, die sich vor allem auf die wiederholenden Tätigkeiten bei Änderungen beziehen:

- die Ausführung der Validierungszielverfahren,
- das Auslesen der validierungsrelevanten Informationen aus dem Validierungsmodell als Eingangsdaten für die Validierungszielverfahren,
- der Vergleich der Prüfkriterien mit den Ergebnissen der Validierungszielverfahren und
- das Erkennen von Änderungen an den Anforderungen oder der Architektur.

Der Nachweis der praktischen Realisierbarkeit dieser Automatisierungen erfolgt durch den Werkzeugprototypen MEASURED (siehe Kapitel 7). Der Validierungsprozess reduziert damit insgesamt den Aufwand für die Architekturvalidierung und trägt zur Verbesserung der Effizienz des Systementwicklungsprozesses bei.

Ausblick Gleichwohl die Entwicklung des Validierungsprozesses grundsätzlich abgeschlossen ist, kann sie an einigen Stellen noch vertieft werden. Ein Beispiel ist die Werkzeugunterstützung (siehe auch Kapitel 7.3). MEASURED ist ein Prototyp, der vor allem zum Nachweis der Automatisierungsmöglichkeiten entwickelt wurde. Neben einer grundlegenden Verbesserung bei der Handhabung wäre es auch denkbar, eine Prüfung der Modellsyn-

tax vor der Durchführung aller Validierungszielverfahren durchzuführen. Auf diese Weise würden Modellierungsfehler vor der ggf. länger dauernden Durchführung der Verfahren entdeckt. Die Transformation von Modellelementen zwischen dem Entwicklungsmodell und dem Validierungsmodell setzt aktuell voraus, dass beide Modelle auf demselben Metamodell, dem der UML, basieren. Hier wäre eine Erweiterung auf andere Modellierungssprachen denkbar. Damit könnte beispielsweise die in Kapitel 9 beschriebene Transformation von BPMN im Entwicklungsmodell auf die UML mit dem Profil BPMN 2-Prozesse im Validierungsmodell unterstützt werden. Des Weiteren wäre die Beschäftigung mit Inhalten wie Modellschneidung [LKR10], Erhaltung von Modellkonsistenz bei selektiver Modellwiederherstellung [GE10] oder die Validierung von Modelltransformationen [RVV09] zur Verbesserung der Benutzerfreundlichkeit sinnvoll. Eine Berücksichtigung dieser Punkte bei der Entwicklung des Prototypen lag außerhalb des Rahmens dieser Arbeit.

Neben den Verbesserungen und der Weiterentwicklung des Werkzeugs wäre auch die Anwendung des Validierungsprozesses auf weitere Systeme wünschenswert. Auf diese Weise könnten weitere praktische Erfahrung gesammelt werden. Bei der Anwendung auf mehrere gleichartige Systeme könnte auch die in Kapitel 8.1 beschriebene projektübergreifende Verwendung von VSNs und Validierungszielverfahren näher untersucht werden.

Bei der Entwicklung des Validierungsprozesses konnte auf einen Ansatz zur Unterstützung des Architekten beim Architekturentwurf nicht weiter eingegangen werden. Es geht dabei um die Zuordnung von Software-Komponenten zu den Hardware-Komonenten einer System-Architektur. Bei einer größeren Komponentenanzahl ergeben sich selbst bei der Vorgabe von Einschränkungen, z. B. falls bestimmte Software-Komponenten einem be-

stimmten Typ von Verarbeitungseinheiten zugeordnet sein sollen, recht viele Zuordnungsmöglichkeiten. Hier wäre ein Verfahren wünschenswert, das in Abhängigkeit verschiedenster Einschränkungen alle sinnvollen Zuordnungsmöglichkeiten ermittelt. In Verbindung mit dem Validierungsprozess könnten dann noch die Architekturentwürfe ermittelt werden, die konform gegenüber den Anforderungen sind. Da die Architekten die Architekturentwürfe auf Basis von Erfahrungen erstellen und für eine ausführliche Suche nach Alternativen meist wenig Zeit bleibt, ergeben sich häufig sehr ähnliche Architekturentwürfe. Durch eine automatisierte Ermittlung ständen verschiedenste Architekturalternativen zur Verfügung, die dann beispielsweise mit Verfahren wie Architecture Tradeoff Analysis Method (ATAM) [CKK02] bewertet werden können.

A Anhang

A.1 NFA-Prozess nach Dörr

In diesem Kapitel wird zunächst der vollständige Prozess zur Erhebung, Analyse und Spezifikation von NFAs vorgestellt. Anschließend wird der Algorithmus zur Erhebung der NFAs im Detail betrachtet.

A.1.1 NFA-Prozess

Abbildung A.1 [Dör10, S. 90] zeigt den Prozess zur Erhebung einer vollständigen Menge NFAs nach Dörr. Links und rechts der Aktivitäten sind die beteiligten Artefakte dargestellt, die entweder erfahrungsbasiert oder projektspezifisch sind. In Abhängigkeit von den Pfeilrichtungen stellen sie Ein- oder Ausgabewerte für die damit verbundene Aktivität dar. Der Prozess ist in zwei Phasen unterteilt: Vorbereitung (P1.x) sowie Erhebung und Spezifikation (P2.x). Die vier Aktivitäten der Vorbereitungsphase dienen dazu, die Erhebung der NFAs so effizient wie möglich durchzuführen. Dazu werden zunächste aus der Liste der verfügbaren High Level Quality Attributes (HLQAs) die relevanten QAs ausgewählt. Es entsteht eine Liste priorisierter HLQAs, mit deren Hilfe das Referenzqualitätsmodell projektspezifisch angepasst wird und Abhängigkeiten zwischen den QAs ermittelt werden. Das Ergebnis ist ein projektspezifisches Qualitätsmodell, welches in der vierten Vorbereitungsaktivität als Grundlage für die projektspezifische Anpassung des Referenz-Templates zur Anforderungsdokumentation und der Referenz-Checklisten für das Erhebungsverfahren dient.

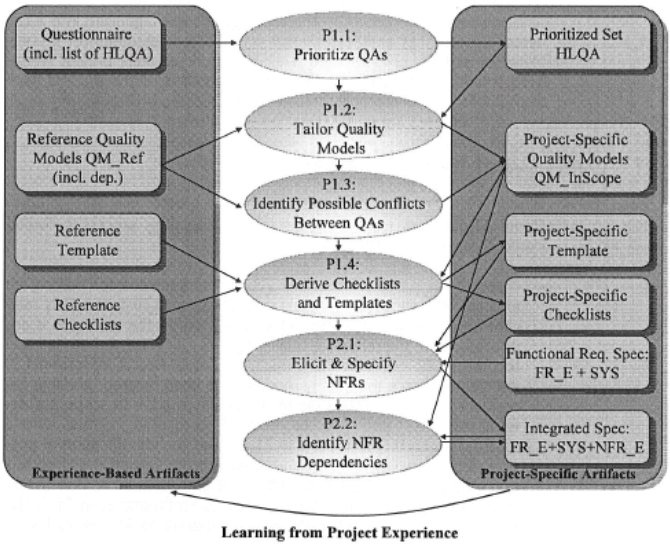


Abbildung A.1: Erhebungsprozess für nicht-funktionale Anforderungen

Nach Abschluss von Phase 1 kann die Erhebung der NFAs zusammen mit dem Kunden beginnen. Hierzu werden das projektspezifische Qualitätsmodell, das projektspezifische Template, die projektspezifischen Checklisten sowie die Spezifikation der funktionalen Anforderungen inklusive der Subsysteme benötigt. Die erhobenen NFAs werden entsprechend des projektspezifischen Templates in die Spezifikation integriert. Anschließend werden anhand des projektspezifischen Qualitätsmodells Abhängigkeiten zwischen den NFAs identifziert, um mögliche Konflikte aufzudecken. Abschließend werden die im Projekt gesammelten Erfahrungen mit den bereits vorhandenen Erfahrungen konsolidiert, was eine stetige Weiterentwicklung der erfahrungsbasierten Artefakte ermöglicht.

A.1.2 Erhebungsalgorithmus

Auf Basis der projektspezifischen Artefakte, des Qualitätsmodells sowie der Spezifikation der funktionalen Anforderungen inklusive der Informationen zu den Subsystemen, erfolgt in der zweiten Phase die Erhebung und Spezifikation der nicht-funktionalen Anforderungen. Das grundlegende Vorgehen dabei ist in Abbildung A.2 [Dör10, S. 78] mit Hilfe einer Vergleichsmatrix dargestellt. Die Zeilen entsprechen den FCEs (User Task, System Task, Data Item) und Subsystemen aus der Spezifikation. Ihnen gegenüber stehen die entsprechend zugeordneten EQAs. Die Erfassung der NFAs erfolgt über einen vollständigen paarweisen Vergleich von FCE oder Subsystemen und EQA.

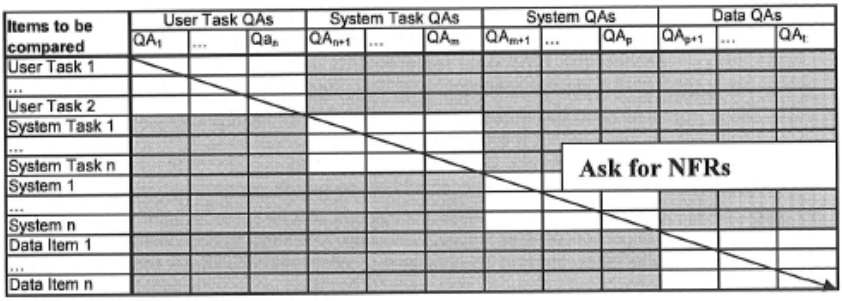


Abbildung A.2: Matrix für die zu vergleichenden Elemente im Erhebungsalgorithmus

Für jedes Paar wird der Kunde gefragt, ob ein NFA existiert. Die Befragung wird durch die projektspezifischen Checklisten unterstützt. Da jedem FCE oder Subsystem ein spezielles EQA zugeordnet ist (siehe Metamodell in Abbildung 5.6), ist es nicht notwendig jedes EQA mit jedem FCE oder Subsystem zu vergleichen. Solche Paarungen sind in der Vergleichsmatrix grau

hinterlegt. Für die Befragung des Kunden müssen nur die weiß hinterlegten Paarungen berücksichtigt werden.

Der Erhebungsalgorithmus berücksichtigt bei seinem Vorgehen die Verfeinerungsbeziehungen zwischen den User Tasks und System Tasks. User Tasks werden häufig in System Tasks unterteilt, was sich auch auf die QAs auswirkt. Dabei geht Dörr von der Annahme aus, dass die Wahrscheinlichkeit eine NFA zu erheben größer ist, wenn der Kunde in Verfeinerungen denkt anstatt direkt danach gefragt zu werden. Aus diesem Grund besteht das Erheben der NFAs aus den folgenden drei Schritten:

1. nach NFAs fragen
2. NFAs verfeinern
3. Paarkombination FCE oder Subsystem und EQA als erledigt markieren

Der letzte Schritt ist besonders wichtig für eine inkrementelle Durchführung des Erhebungsalgorithmus. Verändern sich Anforderungen und damit die FCEs oder sollen neue EQAs überprüft werden, kann anhand der Markierung festgestellt werden welche Paarungen von den Änderungen betroffen sind. Nur diese müssen dann bei der nächsten Durchführung des Algorithmus berücksichtigt werden.

Abbildung A.3 [Dör10, S. 82] zeigt den Ablauf des Algorithmus an einem Beispiel, in dem ein User Task durch verschiedene System Tasks verfeinert wird. Der Ablauf des Algorithmus ist durch die Nummern und Pfeile dargestellt. Wie schon bei Abbildung A.2 ergibt sich durch die Hinterlegung der Zellen, welche Paare aus FCE oder Subsystemen und EQAs bei der Befragung berücksichtigt (weiß) bzw. ausgelassen (grau) werden. Die Erhebung

beginnt mit dem $User\,Task\,QA_1$ für alle User Task-Elemente ohne mögliche Verfeinerungen und wird sequentiell für alle User Task QAs fortgesetzt (Nummer 1). Danach wird zunächst für die verfeinerten System Tasks 1 bis n und dem $System\,Task\,QA_{n+1}$ nach NFAs gefragt (Nummer 2). Anschließend wird für die restlichen System Tasks $n+1$ bis m nach NFAs für das $System\,Task\,QA_{n+1}$ gefragt (Nummer 3). Es folgt die Befragung für die verfeinerten System Tasks und dem $System\,Task\,QA_{n+2}$ (Nummer 4) sowie für die System Tasks und $System\,Task\,QA_{n+2}$ (Nummer 5). Dieses Vorgehen wiederholt sich analog für die restlichen System Task QAs (Nummer 6). Sind die NFAs für die System Tasks erhoben, folgt die Befragung für die Subsystem und die Data Items mit den entsprechenden EQAs (Nummer 7 und 8).

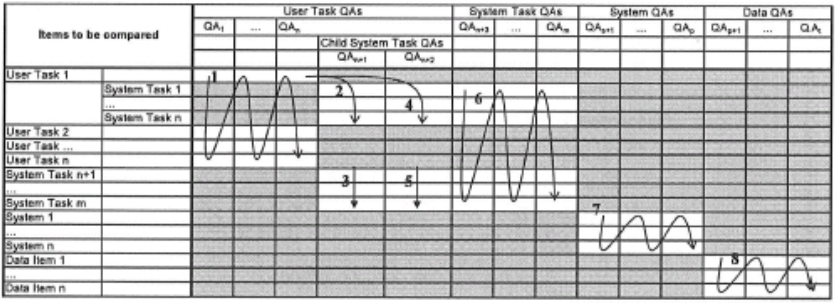


Abbildung A.3: Beispiel für den Ablauf des Erhebungsalgorithmus in einer Vergleichsmatrix

Das Hauptziel des NFA-Prozesses ist die Erhebung einer vollständiger Menge NFAs. Dörr definiert eine Menge an NFAs als vollständig, falls für jedes Paar aus FCE oder Subsystem mit dem entsprechenden EQA eine nichtfunktionale Anforderung existiert oder dafür kein Bedarf von Seiten der Projektbeteiligten existiert. Da der Algorithmus einen solchen paarweisen Ver-

gleich durchführt, garantiert er die Erhebung einer vollständigen Menge an NFAs nach der genannten Definition.

A.2 Verarbeitung von Datenströmen

Zur detaillierten Beschreibung der Signalverarbeitung wird meist die Verarbeitung einer bestimmten Menge an Echosignalen durch die einzelnen Software-Komponenten verwendet. Es wird dabei der Einfachheit halber angenommen, dass die Verarbeitung eines zweiten Datenpakets erst nach der vollständigen Abarbeitung des ersten beginnt, die Verarbeitung also sequentiell durchgeführt wird. In einem realen Radar-System beginnt die Verarbeitung der zeitlich nachfolgenden Datenpakete allerdings bereits vor der vollständigen Abarbeitung des ersten Datenpakets. Zum einen werden für die Verarbeitung eines Datenpakets Informationen über das vorherige benötigt, zum anderen ist nur auf diese Weise eine Verarbeitung des Datenstroms in Echtzeit möglich. Die Auswirkungen dieser parallelen Verarbeitung auf die Gesamtverarbeitungszeit und die Verarbeitungszeit der einzelnen Datenpakete werden im Folgenden anhand eines Beispiels schematisch verdeutlicht.

Abbildung A.4 zeigt drei Software-Komponenten *SW1*, *SW2* und *SW3*. Diese werden auf den beiden Verarbeitungseinheiten[167] *HW1* und *HW2* ausgeführt, wobei durch die Abhängigkeitsbeziehung mit Stereotype *executedOn* zu erkennen ist, dass *SW1* und *SW2* auf *HW1* ausgeführt werden und *SW3* auf *HW2*. Die Abarbeitungsreihenfolge ergibt sich über die Abhängigkeitsbeziehung zwischen den Software-Komponenten, die mit dem Stereotyp *dataExchange* versehen ist.

[167] An dieser Stelle wird nicht der Begriff Prozessor verwendet, da auch FPGAs zur Signalverarbeitung eingesetzt werden.

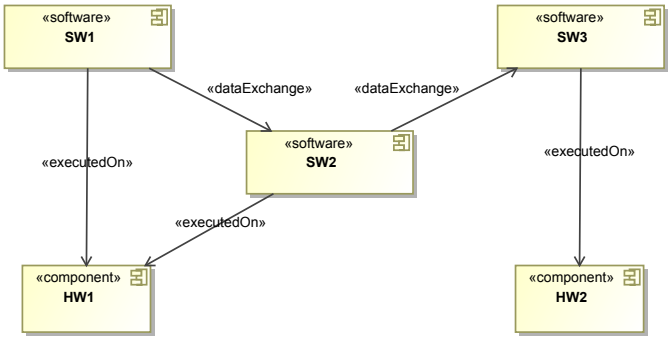


Abbildung A.4: Zuordnung der Software-Komponenten zu den Hardware-Komponenten

Jede Software-Komponente benötigt eine bestimmte Menge an Zeiteinheiten zur Abarbeitung eines Datenpakets. Der Wert basiert auf der Annahme, dass die Leistung der Verarbeitungseinheit der Software-Komponenten vollständig zur Verfügung steht. Sind zwei Software-Komponenten auf derselben Verarbeitungseinheit gleichzeitig aktiv, verdoppelt sich der angegebene Wert.

Unter der Annahme, dass *SW1* 20 Zeiteinheiten zur Verarbeitung eines Datenpakets benötigt, *SW2* 50 Zeiteinheiten und *SW3* 80 Zeiteinheiten, ergibt sich für die die Gesamtverarbeitungszeit von drei Datenpaketen bei einer sequentiellen Verarbeitung (Reihenfolge SW1, SW2, SW3) ein Wert von 450 Zeiteinheiten. Abbildung A.5 zeigt die benötigten Zeitintervalle für die dazugehörigen Kombinationen aus Datenpaket und verarbeitender Software-Komponente. Der hellgraue Hintergrund der Rechtecke weist darauf hin, dass nur eine Software-Komponente auf der zugeordneten Verarbeitungseinheit aktiv ist. Datenpaket 1 (D1) wird also zunächst von SW1 für 20 Zeiteinheiten verarbeitet (D1/SW1), dann 50 Zeiteinheiten von SW2 (D1/SW2)

und anschließend 80 Zeiteinheiten von SW3 (D1/SW3). Nach 150 Zeiteinheiten ist die Verarbeitung des ersten Datenpakets abgeschlossen und die Verarbeitung von *D2* beginnt.

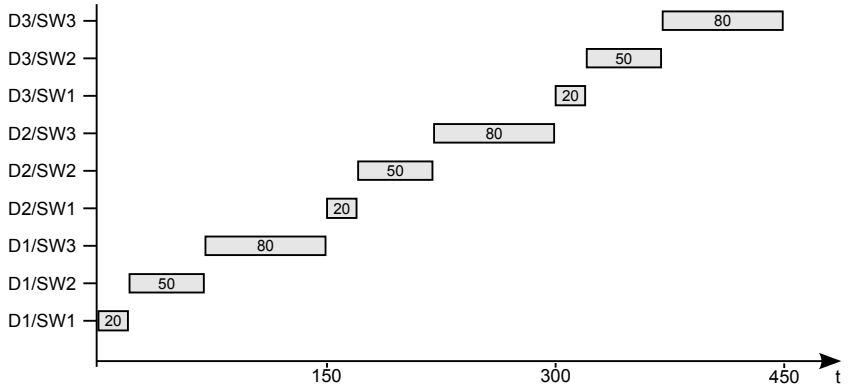


Abbildung A.5: Verarbeitungszeit der Datenpakete bei sequentieller Datenverarbeitung

Abbildung A.6 zeigt die Zeiten für eine parallele Verarbeitung der Daten. Der dunkelgraue Hintergrund der Rechtecke weist darauf hin, dass für die Datenverarbeitung nur die Hälfte der Leistung zur Verfügung steht, da parallel die zweite Software-Komponente ein anderes Datenpaket verarbeitet. Die Verarbeitung von *D2* durch *SW1* benötigt deshalb nicht nur 20, sondern 40 Zeiteinheiten. Für diesen Zeitraum steht auch *SW2* nur die Hälfte der Leistung zur Verfügung, so dass nach den 40 Zeiteinheiten normiert auf die benötigte Verarbeitungszeit, die sich auf eine hundertprozentige Verfügbarkeit der Verarbeitungsressource beziehen, nur 20 von den 50 Zeiteinheiten vergangen sind. Da nach dem Ende der Verarbeitung von *D2* durch *SW1* der Software-Komponente *SW2* die Verarbeitungseinheit wieder exklusiv zur Verfügung steht, vergehen 30 Zeiteinheiten bis zur vollständigen Verar-

beitung von *D1*. Für die Gesamtverarbeitungszeit werden 330 Zeiteinheiten benötigt.

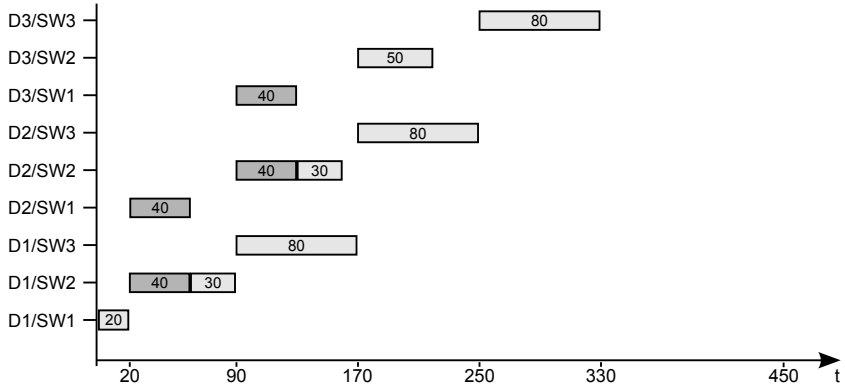


Abbildung A.6: Verarbeitungszeit der Datenpakete bei paralleler Datenverarbeitung

Wie sich bei der parallelen Verarbeitung von D1 und D2 in Abbildung A.6 bereits andeutet, ähnelt die Simulation der Radarsignalverarbeitung der Pipelineverarbeitung. Dies wird noch deutlicher, wenn die Anzahl der Datenpakete und Software-Komponenten erhöht wird. Die Pipeline-Stufe entspricht dabei den gleichzeitig verarbeiteten Datenpaketen, wobei die Zykluszeit variabel ist und vor Beginn jedes Zyklus neu bestimmt werden muss. Kapitel 6 geht auf die Simulation eines kontinuierlichen Radardatenstroms detailliert ein, um die Simulation eines Radar-Systems unter Berücksichtigung mehrerer architekturspezifischer Aspekte zu erläutern.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AADL	Architecture Analysis & Design Language. 140, 155
ADL	Architecture Description Language. 141, 144
API	Application Programming Interface. 293
ARIS	Architektur integrierter Informationssysteme. 296, 297
ATAM	Architecture Tradeoff Analysis Method. 17, 133, 140, 141, 148, 337
AUTOSAR	AUTomotive Open System ARchitecture. 138, 151, 155
BPM	Business Process Management. 293
BPMN	Business Process Model and Notation. 284, 286, 297, 298, 303, 319, 336
CIO	Chief Information Officer. 123
CNL	Controlled Native Language. 77
COSMOD	sCenario and gOal based SysteM delOpment methoD. 54, 124
CTS	Configuration Transition Model. 147
DE	Design Engineering. 127
DIN	Deutsche Institut für Normung. 41

DMSO	Defence Modeling and Simulation Organization. 43
EA	Enterprise Architect. 220, 230, 236, 239, 255, 257
EAST-ADL	Electronics Architecture and Software Technology - Architecture Description Language. 152, 155
ECU	Electronic Control Unit. 137, 138
EEA	Electric and Electronic Architecture. 144
eEPK	erweiterte ereignisgesteuerte Prozesskette. 286, 297
EMF	Eclipse Modelling Framework. 229, 230, 255
EMP	Eclipse Modeling Project. 257
EN	Europäische Norm. 41
EQA	Elementary Quality Attribute. 35, 36, 107, 108, 110, 111, 114, 130–132, 341–343
Evo	Evolutionary Project Management. 127
FCE	Functional Conceptual Element. 35, 36, 107–110, 114, 341–343
Flop	Floating Point Operation. 81, 82, 85, 200, 243, 246, 250

FPGA	Field Programmable Gate Array. 90, 95, 96, 161, 162, 179, 182, 183, 236, 344
GeneSEZ	Generative Software Engineering Zwickau. 230
GI	Gesellschaft für Informatik. 105
HDD	Hard Disk Drive. 161, 162
HLQA	High Level Quality Attribute. 339
IE	Impact Estimation. 127
IEC	International Electrotechnical Commission. 26, 32, 120
IEEE	Institute of Electric and Electronic Engineers. 27, 30, 40, 41, 67
INCOSE	International Council on Software Engineering. 23, 25, 41, 42, 121, 122
IREB	International Requirements Engineering Board. 31
ISO	International Organization for Standardization. 23, 26, 32, 41, 75, 120
LTL	lineare temporale Logik. 117, 149
MADES	Model-based methods and tools for Avionics and surveillance embeddeD SystEmS. 149–151, 156, 258

MARTE	Modeling and Analysis of Real Time and Embedded systems. 148, 149, 151, 153, 155, 156
MEASURED	Modellgetriebene Validierung von System-Architekturen mit der UML. 20, 22, 209, 225, 226, 230, 254, 335
ModelicaML	Modelica Modeling Language. 135, 136, 151
MOF	Meta Object Facility. 269, 270
NFA	nicht-funktionale Anforderung. 27, 33–37, 48, 67, 74, 103, 105, 119, 128, 130–132, 167, 223, 326, 339–344
OCL	Object Constraint Language. 257
OMG	Object Management Group. 83, 149, 155, 216, 269
PSL	Property Specification Language. 117–119
QA	Quality Attribute. 36, 37, 110–114, 339, 342, 343
RE	Requirements Engineering. 54, 124
ReqIF	Requirements Interchange Format. 216
RS	Requirements Specification. 127

RTSC	Real Time Statechart. 137, 151
RUP	Rational Unified Process. 54
RVM	Requirements Violation Monitor. 134, 135
SAAM	Software Architecture Analysis Method. 17, 133, 140
SAVI	System Architecture Virtual Integration. 140
SE	Systems Engineering. 25
SPES	Software Platform Embedded Systems. 137
SQC	Specification Quality Control. 127
SSD	Solid State Disk. 187, 191, 195
SysML	System Modeling Language. 137, 140, 141, 147, 149, 151, 153–155, 164
TBC	to be checked. 80, 81
TBD	to be defined. 80, 81
TLM	Transaction Level Modeling. 152, 153
TRIO	Tempo Reale ImplicitO. 149

Literaturverzeichnis

[AGMG11] Nico Adler, Philipp Graf, und Klaus D. Müller-Glaser. Model-based Consistency Checks of Electric and Electronic Architectures against Requirements. In Stefan van Baelen, Sébastien Gérard, Ileana Ober, Thomas Weigert, Huascar Espinoza, und Iulian Ober, Hrsg., *Proceedings of the Fourth International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2011): held as part of the International Conference on Model Driven Engineering Languages and Systems (MODELS'11)*, Volume CEUR-WS vol. 795, Seite 71–83, 2011.

[APUW09] Enime G. Aydal, Richard F. Paige, Mark Utting, und Jim Woodcock. Putting Formal Specifications under the Magnifying Glass: Model-based Testing for Validation. In *Proceedings of the International Conference on Software Testing, Verification and Validation: Denver, Colorado, April 2009*, Seite 131–140, Piscataway, NJ, 2009. IEEE Computer Society.

[BB02] Janet E. Burge und David C. Brown. NFRs: Fact or Fiction? Computer Science Technical Report. WPI-CS-TR-02-01, 2002.

[Bér01] Béatrice Bérard. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer, Berlin, 2001.

[BH12] Simon Burton und Albert Habermann. Automotive Systems Engineering und Functional Safety: The Way Forward. In *Embedded Real Time Software and Systems*, 2012.

[BK08] Christel Baier und Joost-Pieter Katoen. *Principles of model checking*. MIT Press, Cambridge, MA, 2008.

[BMMR10] Luciano Baresi, Angelo Morzenti, Alfredo Motta, und Matteo Rossi. From Interaction Overview Diagrams to Temporal Logic. In Stefan van Baelen, Ileana Ober, Huascar Espinoza, Thomas Weigert, Iulian Ober, und Sébastien Gérard, Hrsg., *Proceedings of the Third International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2010): held as part of the 2010 International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, Volume CEUR-WS vol. 644, Seite 37–51, Oslo, Norwegen, 2010.

[Boe81] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall, Upper Saddle River, NJ, 1981.

[Bro06] Manfred Broy. Challenges in Automotive Software Engineering. In Leon J. Osterweil, Dieter Rombach, und Mary Lou Soffa, Hrsg., *Proceedings of the 28th International Conference on Software Engineering*, Seite 33–42, New York, NY, 2006. ACM.

[Car09] Carnegie Mellon University. SAE Architecture Analysis & Design Language (AADL). http://www.aadl.info, 2009.

[CdPL01] Luiz Marcio Cysneiros und Julio César Sampaio do Prado Leite. Using the Language Extended Lexicon to Support Non-Functional Requirements Elicitation. In *Workshop em Engenharia de Requisitos*, Seite 139–153, 2001.

[CG03] Lukai Cai und Daniel Gajski. Transaction Level Modeling: An Overview. In Rajesh Grupta, Yukihiro Nakamura, Alex Orailoglu, und Pai H. Chou, Hrsg., *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Seite 19–24. ACM, 2003.

[CGL94] Edmund M. Clarke, Orna Grumberg, und David E. Long. Model Checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[CGNA+11] Juan M. de Carrillo Gea, Joaquín Nicolás, José L. Fernández Alemán, Ambrosio Toval, Christof Ebert, und Aurora Vizcaíno. Requirements Engineering Tools. *IEEE Software*, 28(4):86–91, 2011.

[CKK02] Paul Clements, Rick Kazman, und Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. SEI series in software engineering. Addison-Wesley, Boston, 2002.

[CNY95] L. Chung, Brian A. Nixon, und Eric Yu. Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design. In D. Garlan, Hrsg., *1st International Workshop on Architectures for Software Systems*, Seite 31–43, 1995.

[CNYM99] Lawrence Chung, Brian A. Nixon, Eric Yu, und John Mylopoulos. *Non-Functional Eequirements in Software Engineering*, Volume 5 of *Kluwer international series in software engineering*. Kluwer Academic Publishers, 1999.

[Das05] Jürgen Dassow. *Logik für Informatiker*. Teubner, Stuttgart, 1. Auflage, 2005.

[DBH13] Maya Daneva, Luigi Buglione, und Andrea Herrmann. Software Architects' Experiences of Quality Requirements: What We Know and What We Do Not Know. In Jörg Dörr und Andreas L. Opdahl, Hrsg., *Requirements Engineering: Foundation for Software Quality*, Volume 7830 of *Lecture Notes in Computer Science*, Seite 1–17, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[DHJ+10] Mourad Debbabi, Fawzi Hassaïne, Yosr Jarraya, Andrei Soeanu, und Luay Alawneh. *Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models*. Springer, Berlin, Heidelberg, 2010.

[Dör10] Jörg Dörr. *Elicitation of a Complete Set of Non-Functional Requirements*. PhD thesis, Technische Universität Kaiserslautern, Kaiserslautern, 2010.

[EAS10] EAST-ADL Association. EAST-ADL. http://www.east-adl.info/, 2010.

[Ecl13] Eclipse Foundation. Webseite Projekt EMF Compare. http://www.eclipse.org/emf/compare/, 2013.

[Emm10] Torsten Emmanuel. Planguage - Spezifikation nichtfunktionaler Anforderungen. http://tiny.cc/nh2zix, 2010.

[ER03] Albert Endres und Dieter Rombach. *A Handbook Of Software and Systems Engineering: Empirical Observations, Laws, and Theories*. Pearson Addison Wesley, Harlow, 2003.

[FD96] A. Finkelstein und J. Dowell. A comedy of errors: the London Ambulance Service case study. In *Proceedings of the 8th International Workshop on Software Specification and Design*, Seite 2–4, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[Fri11] Peter Fritzson. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. John Wiley & Sons, Hoboken, NJ, 2011.

[FS13] Otto K. Ferstl und Elmar J. Sinz. *Grundlagen der Wirtschaftsinformatik*. Oldenbourg, München, 7. Auflage, 2013.

[FWH10] P. Feiler, Wrage L., und Hansson J. System Architecture Virtual Integration: A Case Study. In *Embedded Real Time Software and Systems*, 2010.

[GA11] Ian Gray und Neil C. Audsley. Targeting complex embedded architectures by combining the Multicore Communications API (MCAPI) with Compile-Time Virtualisation. In Jan Vitek und Bjorn de Sutter, Hrsg., *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, Seite 51–60. ACM, 2011.

[Gad12] Andreas Gadatsch. *Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: eine Einführung für*

Studenten und Praktiker. Studium. Springer Vieweg, Wiesbaden, 7. Auflage, 2012.

[GB05] Tom Gilb und Lindsey Brodie. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Butterworth-Heinemann, Oxford, 2005.

[GE10] Iris Groher und Alexander Egyed. Selective and Consistent Undoing of Model Changes. In Dorina C. Petriu, Nicolas Rouquette, und Haugen Øystein, Hrsg., *Prodeedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Oslo, Norwegen, Oktober 2010*, Volume II of *Lecture Notes in Computer Science*, Seite 123–137, Berlin, 2010. Springer.

[Gen13] GeneSEZ Forschungsgruppe. Generative Software Engineering Zwickau. http://www.genesez.org, 2013.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison-Wesley, Boston, 1. Auflage, 1994.

[Gra98] Jeffrey O. Grady. *System Validation and Verification*. CRC Press, Boca Raton, 1998.

[Hal08] Robert J. Hall. Validating Real-Time Specifications using Real-Time Event Queue Modeling. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software En-*

gineering: L'Aquila, Italien, September 2008, Seite 79–88. IEEE Computer Society, 2008.

[HBG+09] Tobias Haubold, Georg Beier, Wolfgang Golubski, Nico Herbig, Gerrit Beine, und Oliver Arnold. The Technical Foundation of the GeneSEZ MDSD Approach. In Hamido Fujita und Vladimir Marik, Hrsg., *New Trends in Software Methodologies, Tools and Techniques*, Volume 199 of *Frontiers in Artificial Intelligence and Applications*, Seite 39–60. IOS Press, 2009.

[HBGH08] Tobias Haubold, Georg Beier, Wolfgang Golubski, und Nico Herbig. The GeneSEZ approach to model-driven software development. In Khaled M. Elleithy, Hrsg., *Advanced Techniques in Computing Sciences and Software Engineering: part of the International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering*, Seite 395–400, Bridgeport, Connecticut, USA, 2008. Springer.

[HCK03] Michael Hammer, James Champy, und Patricia Künzel. *Business Reengineering*. Campus-Verl, Frankfurt/Main, 7. Auflage, 2003.

[HKD09] Andrea Herrmann, Daniel Kerkow, und Joerg Doerr. Exploring the Characteristics of NFR Methods: A Dialogue About Two Approaches. In *Proceedings of the 29th International Conference on Software Engineering: Minneapolis, MN, USA, May 20-26, 2007*, Piscataway, 2009. IEEE Computer Society.

[HMM11] Jörg Holtmann, Jan Meyer, und Matthias Meyer. A Seamless Model-Based Development Process for Automotive Systems. In Ralf Reussner und Walter Alexander Pretschner, Hrsg., *Software Engineering 2011: 21.-25. Februar 2011 in Karlsruhe*, Volume P-184 of *GI-Edition - Lecture Notes in Informatics*, Seite 79–88, Bonn, 2011. Bonner Köllen Verlag.

[Hol10] Jörg Holtmann. Mit Satzmustern von textuellen Anforderungen zu Modellen. *OBJEKTspektrum*, RE/2010, 2010.

[Hop93] V. David Hopkin. Verification and Validation: Concepts, Issues, and Applications. In John A. Wise, V. David Hopkin, und Paul Stager, Hrsg., *Verification and Validation of Complex Systems: NATO Advanced Study Institute and Validation of Complex and Integrated Human-Machine Systems*, Seite 9–33, Berlin, Heidelberg, 1993. Springer.

[HP08] Andrea Herrmann und Barbara Paech. MOQARE: misuse-oriented quality requirements engineering. *Requirements Engineering*, 13(1):73–86, 2008.

[IAB10] Muhammad Zohaib Iqbal, Andrea Arcuri, und Lionel Briand. Environment Modeling with UML/MARTE to Support Black-Box System Testing for Real-Time Embedded Systems: Methodology and Industrial Case Studies. In Dorina C. Petriu, Nicolas Rouquette, und Øystein Haugen, Hrsg., *Prodeedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Oslo, Norwegen, Oktober 2010*, Volume I of

Lecture Notes in Computer Science, Seite 286–300, Berlin, 2010. Springer.

[IEE90] IEEE Computer Society. IEEE Standard Glossary of Software Engineering Terminology, 1990.

[IEE98] IEEE Computer Society. IEEE Recommended Practice for Software Requirements Specifications, 1998.

[IEE10] IEEE Computer Society. IEEE Standard for Property Specification Language (PSL), 2010.

[IG11] IREB und Martin Glinz. A Glossary of Requirements Engineering Terminology. http://www.certified-re.de/fileadmin/IREB/Download/Homepage%20Downloads/IREB_CPRE_Glossary_11.1.pdf, 2011.

[IHO02] M. Ionita, D. Hammer, und H. Obbink. Scenario-based Software Architecture Evaluation Methods: An Overview. In *Workshop on Methods and Techniques for Software Architecture Review and Assessment at the International Conference on Software Engineering*, 2002.

[INC11] INCOSE. INCOSE Systems Engineering Handbook. http://www.incose.org/ProductsPubs/products/sehandbook.aspx, 2011.

[IQG+12] Leandro Soares Indrusiak, Imran Quadri, Ian Gray, Neil Audsley, und Andrey Sadovykh. A MARTE subset to enable

application-platform co-simulation and schedulability analysis of NoC-based embedded systems. In Leandro Soares Indrusiak, Guy Gogniat, und Nikolaos Voros, Hrsg., *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip ()*, Seite 1–7, 2012.

[IRE13] IREB. Webseite des International Requirements Engineering Board e.V. http://www.certified-re.de, 2013.

[ISO01] ISO International Organization for Standardization. Software engineering - Product quality - Part 1: Quality model, 2001.

[ISO05] ISO International Organization for Standardization. Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE, 2005.

[ISO08] ISO International Organization for Standardization. Systems and software engineering – System life cycle processes, 2008.

[ISO11] ISO International Organization for Standardization. Software-Engineering – Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) – Qualitätsmodell und Leitlinien, 2011.

[Jac12] Daniel Jackson. *Software Abstractions: Logic, Language and Analysis*. MIT Press, Cambridge, MA, 2012.

[KK06] Roland Kapeller und Stefan Krause. So natürlich wie Sprechen - Embedded Systeme modellieren. *Design & Elektronik*, 8:64–67, 2006.

[Kli07] Thomas Klingenberg. Das System V: Das V-Modell XT in der Praxis. http://www.ap-verlag.de/Online-Artikel/20070506/20070506c%20Microtool%20V-Modell%20XT%20IT-Projekte%20Vorgehensweise.htm, 2007.

[KNS92] Gerhard Keller, Markus Nüttgens, und August-Wilhelm Scheer. Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". In August-Wilhelm Scheer, Hrsg., *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Volume 89. 1992.

[Kru95] Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.

[LKR10] Kevin Lano und Shekoufeh Kolahdouz-Rahimi. Slicing of UML Models Using Model Transformations. In Dorina C. Petriu, Nicolas Rouquette, und Haugen Øystein, Hrsg., *Prodeedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Oslo, Norwegen, Oktober 2010*, Volume II of *Lecture Notes in Computer Science*, Seite 228–242, Berlin, 2010. Springer.

[LLS+12] Shuang Liu, Yang Liu, Jun Sun, Jin Song Dong, und Bimlesh Wadhwa. Formalizing UML State Machine Semantics for Automatic Verification. http://www.comp.nus.edu.sg/~lius87/uml/techreport/uml_sm_semantics.pdf, 2012.

[Lud08] Albrecht K. Ludloff. *Praxiswissen Radar und Radarsignalverarbeitung*. Praxis. Vieweg + Teubner, Wiesbaden, 4. Auflage, 2008.

[MAD11] MADES. D3.3 Formal Dynamic Semantics of the Modelling Notation. http://www.mades-project.org, 2011.

[Mat12] Mathworks. Simulink - Simulation und Model-Based Design. http://www.mathworks.de/products/simulink/, 2012.

[MB02] Stephen J. Mellor und Marc J. Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley, Boston, 2002.

[MKS00] Mohamed Mancona Kandé und Alfred Strohmeier. Towards a UML Profile for Software Architecture Descriptions. In Andy Evans, Stuart Kent, und Bran Selic, Hrsg., *Proceedings ot the Third International Conference of «UML» 2000: Advancing the Standard*, Volume 1939 of *Lecture Notes in Computer Science*, Seite 513–527, Berlin, Heidelberg, 2000. Springer.

[MR09] Mark W. Maier und Eberhardt Rechtin. *The Art of Systems Architecting*. CRC Press, Boca Raton, 3. Auflage, 2009.

[MW11] Zoltán Micskei und Hélène Waeselynck. The many meanings of UML 2 Sequence Diagrams: a survey. *Software & Systems Modeling*, 10(4):489–514, 2011.

[NGTS10] Florian Noyrit, Sébastien Gérard, Francois Terrier, und Bran Selic. Consistent Modeling Using Multiple UML Profiles. In Dorina C. Petriu, Nicolas Rouquette, und Øystein Haugen, Hrsg., *Prodeedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Oslo, Norwegen, Ok-*

tober 2010, Volume I of *Lecture Notes in Computer Science*, Seite 392–406, Berlin, 2010. Springer.

[Obj10] Object Management Group. OMG Systems Modeling Language. http://www.omgsysml.org/, 2010.

[Obj11a] Object Management Group. Modeling and Analysis of Real-Time and Embedded Systems (MARTE). http://www.omgmarte.org/, 2011.

[Obj11b] Object Management Group. OMG Unified Modeling Language (OMG UML): Superstructure. http://www.omg.org/spec/UML/2.4.1/, 2011.

[Obj11c] Object Management Group. Requirements Interchange Format (ReqIF). http://www.omg.org/spec/ReqIF/1.0.1/, 2011.

[Obj13] Object Management Group. UML Profile for BPMN 2 Processes Version. http://www.omg.org/spec/BPMNProfile/, 2013.

[OD11] Iulian Ober und Iulia Dragomir. Unambiguous UML Composite Structures: The OMEGA2 Experience. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith G. Jeffery, Ratislav Královic, Marko Vukolic, und Stefan Wolf, Hrsg., *SOFSEM 2011: Theory and Practice of Computer Science*, Volume 6543 of *Lecture Notes in Computer Science*, Seite 418–430. Springer, 2011.

[Pap11] Lothar Papula. *Mathematik für Ingenieure und Naturwissenschaftler Band 3: Vektoranalysis, Wahrscheinlichkeitsrechnung, Mathematische Statistik, Fehler- und Ausgleichsrechnung*. Vieweg + Teubner, Wiesbaden, 6. Auflage, 2011.

[Pel06] Radek Pelánek. *Reduction and Abstraction Techniques for Model Checking*. PhD thesis, Masaryk University, Brno, 2006.

[PGQ11a] André Pflüger, Wolfgang Golubski, und Stefan Queins. Model Driven Validation of System Architectures. In Taghi M. Khoshgoftaar, Swapna Gokhale, und Ankur Agarwal, Hrsg., *HASE 2011*, Seite 25–28, Los Alamitos, CA, [Piscataway, NJ], 2011. IEEE Computer Society and IEEE.

[PGQ11b] André Pflüger, Wolfgang Golubski, und Stefan Queins. Modellgetriebene Validierung von System-Architekturen. In Ralf Reussner und Walter Alexander Pretschner, Hrsg., *Software Engineering 2011: 21.-25. Februar 2011 in Karlsruhe*, Volume P-184 of *GI-Edition - Lecture Notes in Informatics*, Seite 119–128, Bonn, 2011. Bonner Köllen Verlag.

[PGQ11c] André Pflüger, Wolfgang Golubski, und Stefan Queins. System Architecture Validation with UML. In Markus Nüttgens, Oliver Thomas, und Barbara Weber, Hrsg., *Enterprise Modelling and Information Systems Architectures*, Volume 190 of *GI-edition Proceedings*, Seite 207–212, Bonn, 2011. Ges. für Informatik.

[PGQ12] André Pflüger, Wolfgang Golubski, und Stefan Queins. Tool-Supported Model-Driven Validation Process for System Archi-

tectures. In Iulian Ober, Hrsg., *Proceedings of the 5th International Workshop on Model Based Architecting and Construction of Embedded Systems*, Seite 1–6, New York, NY, 2012. ACM.

[PGQ13a] André Pflüger, Wolfgang Golubski, und Stefan Queins. Process for the Validation of System Architectures against Requirements. In Vicente García Díaz, Juan Manuel Cueva Lovelle, Begona Cristina Pelayo García-Bustelo, und Oscar Sanjuán Martínez, Hrsg., *Progressions and Innovations in Model-Driven Software Engineering*, Seite 209–229. IGI Global, 2013.

[PGQ13b] André Pflüger, Wolfgang Golubski, und Stefan Queins. Validation of System Architectures Against Requirements. In Tarek M. Sobh und Khaled Elleithy, Hrsg., *Emerging trends in computing, informatics, systems sciences, and engineering*, Volume 151 of *Lecture Notes in Electrical Engineering*, Seite 423–435, New York, NY, 2013. Springer.

[PHAB12] Klaus Pohl, Harald Hönninger, Reinhold Achatz, und Manfred Broy, Hrsg. *Model-Based engineering of Embedded Systems: The SPES 2020 Methodology*. Springer, Berlin, New York, 2012.

[Poh08] Klaus Pohl. *Requirements Engineering: Grundlagen, Prinzipien, Techniken*. dpunkt Verlag, Heidelberg, 2. Auflage, 2008.

[Poß12] Christian Poßögel. *Modellgetriebene Entwicklung eines Verfahrens zur Validierung von System-Architekturen am Beispiel eines Radarsystems*. PhD thesis, Westsächsische Hochschule Zwickau, Zwickau, 2012.

[Pra10] Matteo Pradella. Zot. http://home.dei.polimi.it/pradella/Zot/index.html, 2010.

[PS07a] Klaus Pohl und Ernst Sikora. COSMOD-RE: Supporting the Co-Design of Requirements and Architectural Artifacts. In Alistair Sutcliffe und Pankaj Jalote, Hrsg., *Proceedings of the 15th IEEE International Requirements Engineering Conference: Neu Delhi, Indien, October 2007*, Seite 258–261, Los Alamitos, CA, 2007. IEEE Computer Society.

[PS07b] Klaus Pohl und Ernst Sikora. Structuring the Co-design of Requirements and Architecture. In Patrick Heymans, Barbara Paech, und Pete Sawyer, Hrsg., *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality: Trondheim, Norwegen, Juni 2007*, Volume 4542 of *Lecture Notes in Computer Science*, Seite 48–62, Berlin, Heidelberg, 2007. Springer.

[PST97] Ben Potter, Jane Sinclair, und David Till. *An introduction to formal specification and Z*. Prentice Hall international series in computer science. Prentice Hall, London, 2. Auflage, 1997.

[QBG+12] Imran R. Quadri, Etienne Brosse, Ian Gray, Nicholas Matragkas, Leandro Soares Indrusiak, Matteo Rossi, Alessandra Bagnato, und Andrey Sadovykh. MADES FP7 EU project: Effective high level SysML/MARTE methodology for real-time and embedded avionics systems. In Leandro Soares Indrusiak, Guy Gogniat, und Nikolaos Voros, Hrsg., *7th International Work-*

shop on Reconfigurable and Communication-Centric Systems-on-Chip (), Seite 1–8, 2012.

[RH09] Ralf Reussner und Wilhelm Hasselbring. *Handbuch der Software-Architektur*. dpunkt Verlag, Heidelberg, 2. Auflage, 2009.

[RHW+10] Louis M. Rose, Markus Herrmannsdoerfer, James R. Williams, Dimitrios S. Kolovos, Kelly Garcés, Richard F. Paige, und Fiona A.C Polack. A Comparison of Model Migration Tools. In Dorina C. Petriu, Nicolas Rouquette, und Øystein Haugen, Hrsg., *Prodeedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Oslo, Norwegen, Oktober 2010*, Volume I of *Lecture Notes in Computer Science*, Seite 61–75, Berlin, 2010. Springer.

[RMP+12] Alek Radjenovic, Nicholas Matragkas, Richard F. Paige, Matteo Rossi, Alfredo Motta, Luciano Baresi, und Dimitrios S. Kolovos. MADES: A Tool Chain for Automated Verification of UML Models of Embedded Systems. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Antonio Vallecillo, Juha-Pekka Tolvanen, Ekkart Kindler, Harald Störrle, und Dimitris Kolovos, Hrsg., *Modelling Foundations and Applications*, Volume 7349 of *Lecture Notes in Computer Science*, Seite 340–351. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Rom85] G. Roman. A taxonomy of current issues in requirements engineering. *Computer*, 18(4):14–23, 1985.

[RP10] Thomas Robert und Vincent Perrier. CoFluent Methodology for UML: UML SysML MARTE Flow for CoFluent Studio, 2010.

[RR06] James Robertson und Suzanne Robertson. *Mastering the requirements process*. Addison-Wesley, Harlow, 2. Auflage, 2006.

[Rup09a] Chris Rupp. *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser Verlag, München, Wien, 5. Auflage, 2009.

[Rup09b] Chris Rupp. *Systemanalyse kompakt*. Spektrum Akademischer Verlag, Berlin, Heidelberg, 2. Auflage, 2009.

[RVV09] István Ráth, Gergely Varró, und Dániel Varró. Change-Driven Model Transformations: Derivation and Processing of Change Histories. In Andreas Schürr und Bran Selic, Hrsg., *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems: Denver, CO, USA, October 2009*, Volume 5795 of *Lecture Notes in Computer Science*, Seite 342–356, Berlin, Heidelberg, 2009. Springer.

[RWCP10] David Redman, Donald Ward, John Chilenski, und Greg Pollari. Virtual Integration for Improved System Design. *AVICPS 2010*, Seite 57–64, 2010.

[Sch09] Wladimir Schamai. Modelica Modeling Language (ModelicaML): A UML Profile for Modelica, 2009.

[SFP13] Wladimir Schamai, Peter Fritzson, und Christian J. J. Paredis. Translation of UML state machines to Modelica: Handling semantic issues. *SIMULATION*, 89(4):498–512, 2013.

[SHFP10] Wladimir Schamai, Philipp Helle, Peter Fritzson, und Paredis Christian JJ. Virtual Verification of System Designs against System Requirements. In Stefan van Baelen, Ileana Ober, Huascar Espinoza, Thomas Weigert, Iulian Ober, und Sébastien Gérard, Hrsg., *Proceedings of the Third International Workshop on Model Based Architecting and Construction of Embedded Systems (ACESMB 2010): held as part of the 2010 International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, Volume CEUR-WS vol. 644, Seite 6–20, Oslo, Norwegen, 2010.

[Sin96] Elmar J. Sinz. Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme. Entwicklung, aktueller Stand und Trends. In Heidi Heilmann, Lutz J. Heinrich, und Friedrich Roithmayr, Hrsg., *Information Engineering*, Seite 123–143. Oldenbourg, München, 1996.

[Sof13] Software Engineering Institute. System and Software Architecture Tradeoff Analysis Method. http://www.sei.cmu.edu/architecture/tools/evaluate/systematam.cfm, 2013.

[Som04] Ian Sommerville. *Software Engineering*. Addison-Wesley, Boston, 7. Auflage, 2004.

[Sta11] Gernot Starke. *Effektive Software-Architekturen: Ein praktischer Leitfaden*. Hanser, München, 5. Auflage, 2011.

[SVEH07] Thomas Stahl, Markus Völter, Sven Efftinge, und Arno Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt Verlag, Heidelberg, 2. Auflage, 2007.

[SWG09] Yu Sun, Jules White, und Jeff Gray. Model Transformation by Demonstration. In Andreas Schürr und Bran Selic, Hrsg., *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems: Denver, CO, USA, October 2009*, Volume 5795 of *Lecture Notes in Computer Science*, Seite 712–726, Berlin, Heidelberg, 2009. Springer.

[TU 12] TU München. Software Plattform Embedded Systems (SPES) 2020. http://spes2020.informatik.tu-muenchen.de/, 2012.

[TU 13] TU Kaiserslautern. Webseite des V-Modell XT. http://www.v-modell-xt.de/, 2013.

[Ver10] Verein Deutscher Ingenieure. Simulation von Logistik-, Materialfluss- und Produktionssystemen - Grundlagen, 2010.

[vL01] Axel van Lamsweerde. Building Formal Requirements Models for Reliable Software. In Dirk Craeynest und Alfred Strohmeier, Hrsg., *Reliable SoftwareTechnologies - Ada-Europe 2001*, Volume 2043 of *Lecture Notes in Computer Science*, Seite 1–20, Berlin, Heidelberg, 2001. Springer.

[Wal11] Ernest Wallmüller. *Software quality engineering: Ein Leitfaden für bessere Software-Qualität*. Hanser Verlag, München, 3. Auflage, 2011.

[WE09] Wilhelm Walcher und Matthias Elbel. *Praktikum der Physik*. Studium. Vieweg + Teubner, Wiesbaden, 9. Auflage, 2009.

[Weg03] Ingo Wegener. *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen*. Springer, Berlin, 2003.

[WfM13] WfMC. Workflow Management Coalition. http://www.wfmc.org, 2013.

[WH11] Donald T. Ward und Steven B. Helton. Estimating Return on Investment for SAVI (a Model-Based Virtual Integration Process). *SAE International Journal of Aerospace*, 4(2):934–943, 2011.

[WLBF09] Jim Woodcock, P. Larsen, J. Bicarregui, und J. Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*, 4(41):1–36, 2009.

[Wor99] Workflow Management Coalition. Terminology & Glossary: WFMC-TC-1011, Version 3. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf, 1999.

[WZEK10] Gereon Weiss, Marc Zeller, Dirk Eilers, und Rudi Knorr. Approach for Iterative Validation of Automotive Embedded Systems. In Stefan van Baelen, Ileana Ober, Huascar Espinoza,

Thomas Weigert, Iulian Ober, und Sébastien Gérard, Hrsg., *Proceedings of the Third International Workshop on Model Based Architecting and Construction of Embedded Systems (ACES-MB 2010): held as part of the 2010 International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, Volume CEUR-WS vol. 644, Seite 69–83, Oslo, Norwegen, 2010.

[ZBR13] Yu Zhou, Luciano Baresi, und Matteo Rossi. Towards a Formal Semantics for UML/MARTE State Machines Based on Hierarchical Timed Automata. *Journal of Computer Science and Technology*, 28(1):188–202, 2013.

University
of Bamberg
Press

Die Entwicklung von Systemen bestehend aus Hardware und Software ist eine Herausforderung für den System-Architekten. Bei der Architekturerstellung muss er die stetig steigende Anzahl an System- Anforderungen, inklusive ihrer Beziehungen untereinander, berücksichtigen und gleichzeitig mit kürzer werdenden Time-to-Market-Zeiten sowie späten Anforderungsänderungen durch den Kunden zurechtkommen. Die vorliegende Arbeit stellt einen Ansatz vor, der den Architekten bei der Validierung der System-Architektur gegenüber den architekturrelevanten Anforderungen sowie bei der Auswirkungsanalyse von Anforderungs- oder Architekturänderungen unterstützt. Dieser Prozess ist Bestandteil der System-Design-Phase und lässt sich in die iterative Entwicklung der System-Architektur integrieren. Um den Überblick zu behalten, fasst der Architekt die Anforderungen anhand von architekturspezifischen Aspekten, sogenannte Validierungsziele, zusammen. Für jedes Validierungsziel werden Validierungszielverfahren und Prüfkriterien zur Bestimmung des Validierungsstatus festgelegt. Falls alle betrachteten Validierungsziele erfüllt sind, erfüllt auch die System-Architektur die relevanten Anforderungen. Als Prüftechnik bevorzugt der Ansatz Simulation anstelle von formalen Verfahren. Da die zur Konfiguration und Durchführung der Simulation benötigten Daten in UML-Modellen dokumentiert und für die Validierung aus diesen ausgelesen werden, wirken sich Modelländerungen direkt auf die Validierungsergebnisse aus. Die wesentlichen Prozessschritte werden mit Hilfe eines Werkzeugs partiell automatisiert, wodurch die Effizienz des Systementwicklungsprozesses verbessert wird.

eISBN: 978-3-86309-253-5

www.uni-bamberg.de/ubp