Ein formales Prozeßmodell für die Software-Entwicklungsmethode SOMT

Erik Petersen, Jürgen Münch, Birgit Geppert

SFB 501 Bericht 3/98

Ein formales Prozeßmodell für die Software-Entwicklungsmethode SOMT

Erik Petersen[*], Jürgen Münch[†], Birgit Geppert[‡]
{muench, geppert}@informatik.uni-kl.de

3/1998

Sonderforschungsbereich 501

[†]Arbeitsgruppe *Software Engineering*
[‡]Arbeitsgruppe *Rechnernetze*
[*] Universität Kaiserslautern

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern
Germany

Zusammenfassung

Die systematische Verbesserung von Techniken zur Entwicklung und Betreuung von Software setzt eine explizite Darstellung der in einem Projekt ablaufenden Vorgänge (Prozesse) voraus. Diese Darstellungen (Prozeßmodelle) werden durch Software-Prozeßmodellierung gewonnen. Eine Sprache zur Beschreibung solcher Modelle ist MVP-L. Verschiedene Standard-Prozeßmodelle existieren bereits. Bisher gibt es jedoch kaum dokumentierte Software-Entwicklungsprozesse, die speziell für die Entwicklung reaktiver Systeme entworfen worden sind, d. h. auf die besonderen Anfordernisse bei der Entwicklung reaktiver Systeme zugeschnitten sind. Auch ist bisher nur wenig Erfahrung dokumentiert, für welche Art von Projektkontexten diese Prozesse gültig sind. Eine Software-Entwicklungsmethode, die - mit Einschränkungen - zur Entwicklung reaktiver Systeme geeignet ist, ist SOMT (SDL-oriented Object Modeling Technique). Dieser Bericht beschreibt die erfahrungsbasierte Modellierung der Software-Entwicklungsprozesse von SOMT mit MVP-L. Zunächst werden inhaltliche Grundlagen der Software-Entwicklungsmethode SOMT beschrieben. Insbesondere wird auf die eingesetzten Techniken und deren Kombination eingegangen. Anschließend werden mögliche Projektkontexte charakterisiert, in denen das SOMT-Modell im Sinne eines Erfahrungselements Gültigkeit hat. Darauf werden der Modellierungsvorgang sowie hierbei gemachte Erfahrungen dokumentiert. Eine vollständige Darstellung des Modells in grafischer MVP-L-Notation befindet sich im Anhang. Die Darstellung des Modells in textueller Notation kann der SFB-Erfahrungsdatenbank entnommen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

Ziel dieses Kapitels ist es, eine Einführung in die Prozeßmodellierung zu geben, den experimentellen Ansatz des Sonderforschungsbereichs 501 vorzustellen sowie die Ziele dieses Berichts aufzuzeigen.

1.1 Software-Prozeßmodellierung

Die Entwicklung großer Software-Systeme ist ein komplexer Prozeß, dessen Beherrschung sich vom Programmieren kleiner Programme in ähnlicher Weise unterscheidet wie das Management eines Industriekonzerns vom Betrieb eines Kleingewerbes. Die quantitative Vielzahl und die qualitative Vielfalt der verschiedenen Aufgaben sowie die große Menge der an einem Software-Projekt beteiligten Personen erfordern Prinzipien, Techniken, Methoden und Werkzeuge, den Software-Entwicklungsprozeß auf der Grundlage eines umfassenden und konsistenten Projektplanes intellektuell verständlich und im ingenieurmäßigen Sinne beherrschbar zu machen.

Obwohl *Software Engineering* als wissenschaftliche Teildisziplin der Informatik, die sich mit einzelnen Aspekten der Software-Entwicklung, der Projektplanung und der ingenieurmäßigen Durchführung von Software-Projekten beschäftigt, bereits seit Ende der 60‘er Jahre etabliert ist[1], sind heutige Entwicklungsprojekte oft durch ungenügende Qualität der Entwicklungstätigkeiten und Produkte charakterisiert. So sind z. B. Zeit- und Kostenüberschreitungen sowie Fehlverhalten in real eingesetzten Systemen keine Seltenheit. „Derzeit

- kostet ein durchschnittliches Software-Entwicklungsprojekt zumeist 50 Prozent mehr als veranschlagt,

---

1. Der Begriff „Software Engineering“ wurde 1968 auf einer von Prof. F. L. Bauer organisierten NATO-Konferenz in Garmisch-Patenkirchen geprägt.

- wird ein Drittel aller umfangreichen Programme vor der Fertigstellung abgebrochen,
- funktionieren 75 Prozent aller komplexen Software-Systeme nicht wie geplant oder werden gar nicht erst eingesetzt“ [Spie95].

Da Software als Querschnittstechnologie in nahezu allen industriell gefertigten Produkten enthalten ist, hat die Einhaltung von Qualitätseigenschaften eine zunehmende Bedeutung, die bis hin zur Existenzfrage einer Unternehmung reichen kann. Die Ursachen hierfür sind vielfältig. Beispiele sind die unpräzise Definition von Qualitätszielen, der Trend hin zu immer komplexeren Systemen, ständig wachsende und sich rascher ändernde Anforderungen und insbesondere auch ein ungenügendes Verständnis des Entwicklungsprozesses.

Da es weder einen universellen Entwicklungsprozeß noch eine optimale Entwicklungsmethode gibt, muß der Entwicklungsprozeß immer in dem ihn umgebenden Kontext betrachtet werden. Der Entwicklungskontext umfaßt Aspekte wie z. B. die Anwendungsdomäne, die vorhandenen Entwicklungstechniken, die Erfahrung der Entwickler, den organisatorischen Rahmen und Budgetgrenzen. Prozesse müssen somit an die jeweiligen Rahmenbedingungen angepaßt werden können.

Einen neueren Ansatz zum besseren Verständnis des Software-Entwicklungsprozesses und der hiermit verbundenen Beherrschung von Komplexität ist die *Software-Prozeßmodellierung.*

Die *Software-Prozeßmodellierung* ist dasjenige Teilgebiet der Informatik, das sich mit der Definition, Verfeinerung und Erprobung von Prinzipien, Methoden, Techniken und Werkzeugen zur Unterstützung

- *der Kommunikation über Software-Entwicklungsprozesse,*
- *der Ablage von Software-Engineering-Prozessen basierend auf wiederverwendbaren Komponenten,*
- *der Analyse von Prozessen und dem Ziehen von Schlußfolgerungen,*
- *der Projekt-Steuerung und Kontrolle,*
- *der automatischen Unterstützung durch prozeß-sensitive Software-Entwicklungsumgebungen*

befaßt.

Mit der expliziten Repräsentation von Prozeßwissen kann man zunächst den Ist-Zustand des Software-Entwicklungsprozesses betrachten, um darauf aufbauend Verbesserungen durchzuführen. Watts Humphrey charakterisiert dies treffend: „If you don‘t know where you are, a map won‘t help“ [Hump89].

Die Relevanz systematischer Software-Entwicklungsprozesse sowie expliziten Prozeßwissens ist industriell erkannt und hat zu einer Bewertung von Unternehmen entsprechend sogenannter Reifegrade von Prozessen geführt [Hump89]. Die hierbei vorgenommene Klassifikation von Softwareentwicklungsorganisationen berücksichtigt den Stand des Wissens bez. Software Engineering und ordnet die Organisationen fünf Reifestadien zu. Das höchste anzustrebende Reifestadium (Ebene 5) ist durch die Existenz evolutionärer Verbesserungszyklen und das direkte Einbringen von Verbesserungen in den Prozeß gekennzeichnet. Ab Ebene 3 findet nach Humphrey Prozeßmodellierung statt, falls die Organisation bestimmte Vorraussetzungen erfüllt. Auch auf der unteren beiden Ebenen

läßt sich Prozeßmodellierung sinnvoll einsetzen, z. B. zum Verstehen von Prozessen. Dies ist jedoch nicht Bestandteil von Humphreys Konzeption. Der Reifegrad einer Softwareentwicklungsorganisation gewinnt als Beurteilungskriterium für die Vergabe von Entwicklungsaufträgen zunehmend an Bedeutung.

Ergebnis der Prozeßmodellierung sind Prozeßmodelle, die durch einen *Projektplan* instantiiert und miteinander verbunden werden. Der Projektplan ist Grundlage für die Durchführung eines konkreten Projekts. Er kann als Eingabe für eine prozeß-sensitive Software-Entwicklungsumgebung benutzt werden [FuGh94], so daß es möglich wird, eine Software-Entwicklungsumgebung an die speziellen Erfordernisse eines bestimmten Projekts anzupassen [RBLV93].

Projektpläne als Spezifikation von Projektabläufen zur Erreichung vorgegebener Projektziele beinhalten Prozeßaspekte, Produktaspekte, Qualitätsaspekte und Ressourcenaspekte [Rom94a]. Die sich hieraus ergebenden Grundbausteine von Projektplänen werden im folgenden in Anlehnung an [RBBL94] und [Rom94a] klassifiziert.

- *Produktmodelle* beschreiben die Struktur und die Eigenschaften von Komponenten der Software. Darunter wird nicht nur den Kode verstanden, sondern sämtliche Artefakte, die bei der Entwicklung oder Betreuung anfallen oder benötigt werden.
- *Prozeßmodelle* sind Repräsentationen für Aktivitäten im Rahmen ingenieurmäßigen Software Engineerings.
- *Ressourcen-Modelle* sind Repräsentationen für Hilfsmittel (entweder Menschen oder Sachobjekte wie z. B. Rechner und Werkzeuge), die zur Durchführung von Prozessen benötigt werden.
- *Qualitätsmodelle* charakterisieren Qualitätseigenschaften. Sie ergänzen die Beschreibungen der ersten drei Modelltypen. Dadurch wird eine Trennung vollzogen zwischen den grundlegenden Konzepten (Produkte, Prozesse, Ressourcen) einerseits und den charakteristischen Eigenschaften andererseits. Zusätzlich existieren globale Qualitätsmodelle, die keinem Objekt der drei Modelltypen zugeordnet werden können. Sie beinhalten Informationen über externe Aspekte wie zum Beispiel „Zeit“.

Die Grundbausteine können verfeinert und zu größeren Gebilden kombiniert werden. Produkte, Prozesse, Ressourcen und Qualitäten sind Instanzen der entsprechenden Modelle und bilden in ihrer Gesamtheit den *Projektplan*. Die Erstellung eines Projektplans umfaßt neben der Identifikation und Beschreibung der Grundbausteine insbesondere deren Integration durch die genaue, vollständige und widerspruchsfreie Beschreibung ihrer Wechselbeziehungen.

Schon durch den Vorgang der expliziten Beschreibung von Prozessen kann ein besseres Verständnis eines Entwicklungsprojekts erreicht werden. Die Verfeinerung von Modellen ermöglicht eine Beherrschung der Komplexität nach dem „Teile und beherrsche“-Prinzip. In [KaCa93], [CuKO92] und [RoVe95] werden diese und weitere *Ziele der Prozeßmodellierung* aufgeführt, die sich folgendermaßen klassifizieren lassen:

(1) Vereinfachung des Verständnisses und Verbesserung der Kommunikation zwischen Projektmitgliedern,

(2) Unterstützung der Prozeßverbesserung,

(3) Unterstützung des Projektmanagements,

(4) Rechnergestützte Anleitung während der Projektdurchführung,

(5) Automatische Abwicklung von Prozeßmodellen,

(6) Unterstützung der Analyse des Entwicklungsprozesses.

Die Vielfältigkeit der Ziele verdeutlicht die hohen Anforderungen, die an die Prozeßmodellierung zu stellen sind. Es ist allerdings nicht Voraussetzung für einen Prozeßmodellierungsansatz, daß er alle Ziele in höchsten Maße unterstützt, vielmehr kann es u. U. für jedes Ziel einen eigenen Prozeßmodellierungsansatz geben.

Da das Ergebnis der Prozeßmodellierung, der Projektplan, Grundlage für die Steuerung des Software-Entwicklungsprozesses ist, muß eine hohe Übereinstimmung zwischen der *realen Welt* und der *Software-Engineering-Modellwelt* angestrebt werden.

Das Verständnis des Unterschiedes zwischen beiden Welten ist wichtig, da die reale Welt Grundlage für die Bildung der Modellwelt ist und umgekehrt aus der Modellwelt Schlüsse für Handlungen in der realen Welt gezogen werden. In der realen Welt werden Software-Projekte mit Menschen durchgeführt. Sie haben bestimmte Tätigkeiten durchzuführen, wobei sie Produkte konsumieren, produzieren oder modifizieren können. Dabei haben sie evtl. bestimmte Qualitätsvorgaben einzuhalten. In der Modellwelt existieren Prozeß-, Produkt-, Ressourcen- und Qualitätsmodelle sowie deren Zusammenfassung in einem Projektplan. Durch Instantiierung der Modelle erhält man Objekte, die bei der Projektdurchführung auf Elemente der realen Welt abgebildet werden.

Dowson und Fernström [DoFe94] beschreiben die Beziehungen zwischen der realen Welt und der Software-Engineering-Modellwelt durch drei Domänen:

- *Prozeß-Definitionsdomäne (engl.: Process definition domain).* Diese Domäne enthält Charakterisierungen von Prozessen, die instantiiert und abgewickelt werden können. Auf die Festlegung auf einen bestimmten Formalismus zur Repräsentation der Prozesse wird verzichtet. In unserem Zusammenhang handelt es sich hierbei um Prozeßmodelle, die Bestandteil der Modellwelt sind.
- *Prozeß-Ausführungsdomäne (engl.: Process performance domain).* Diese Domäne umfaßt die tatsächlichen Aktivitäten, die von Menschen oder Systemen innerhalb des Software-Projekts durchgeführt werden. Alle Projektaktivitäten sind Teil der Prozeß-Ausführungsdomäne. Diese Domäne ist Teil der realen Welt.
- *Prozeß-Definitions-Abwicklungsdomäne (engl.: Process definition enactment domain).* Diese Domäne enthält alle Mechanismen, die dazu dienen, die Ausführung von Prozessen anhand der Prozeß-Definitionen zu unterstützen. Innerhalb dieser Domäne werden Prozeßmodelle instantiiert. Aktivitäten der realen Welt werden als Prozesse repräsentiert. Sie bildet eine Schnittstelle zwischen der Modellwelt und der realen Welt.

Abbildung 1 verdeutlicht die Zusammenhänge. Obwohl der Schwerpunkt bei Dowson und Fernström im Bereich der Instantiierungs- und Ausführungsmechanismen für Prozeßmodelle liegt, soll das vorgestellte Konzept hier die Beziehungen zwischen der realen Welt und der Modellwelt verdeutlichen.

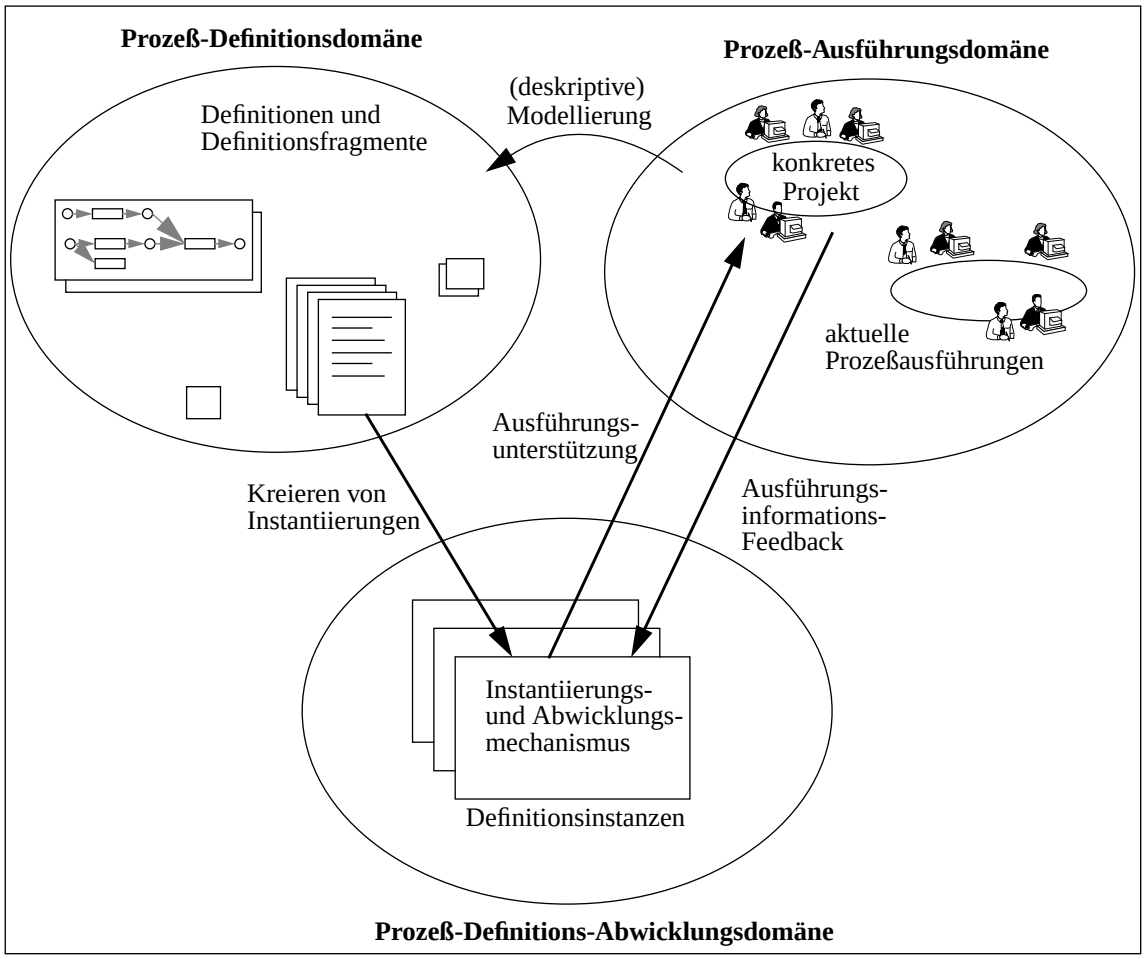

Abbildung 1: Prozeß-Domänen [DoFe94]

Bei der Frage nach der Darstellung von Software-Entwicklungaktivitäten stellt sich allgemein das Problem, in welchem Formalismus Software-Modelle beschrieben werden. In [CuKO92] werden aus den Zielen der Prozeßmodellierung Anforderungen an eine Sprache zur Repräsentation von Software-Modellen abgeleitet, eine Klassifikation für solche Sprachen vorgestellt und existierende Sprachen (z. B.: APPL/A, STATEMATE, MVP-L, MARVEL) entsprechend der Klassifikation typisiert. [ABGM92] und [RoVe95] enthalten Übersichten existierender Repräsentationsformalismen für Software-Prozesse.

In diesem Bericht wird *MVP-L (Multi-View Process Modeling Language)* [BLRV92] als Formalismus zur Repräsentation von Software-Modellen benutzt. MVP-L entstand im Rahmen des MVP-Projekts, das im folgenden kurz in Anlehnung an [RBLV93] und [RBBL94] erläutert wird.

Das *MVP-Projekt (Multi View Process Modeling)* wurde an der University of Maryland at College Park (USA) begonnen und wird seit Anfang 1992 an der Universität Kaiserslautern fortgesetzt. Einer der Gründe für den Start des Projekts waren Erfahrungen mit Entwicklern im Software Engineering Laboratory der NASA (NASA/SEL, Greenbelt, Maryland, USA), wo man erkannte, daß unterschiedliche Vorstellungen darüber bestanden, wie im NASA/SEL Software entwickelt und betreut wurde. Deshalb entschied man

sich, ein Projekt zu starten, daß es ermöglicht, explizite Repräsentationen der in einem Projekt ablaufenden Prozesse aus verschiedenen Blickwinkeln zu gewinnen.

Ziel des MVP-Projekts ist die Entwicklung eines Systems zur Definition, Planung, Abwicklung, Analyse und Wiederverwendung von Software-Prozessen aus verschiedenen Blickwinkeln. Besonderer Wert wird dabei auf die quantitative Charakterisierung von einzelnen Prozeß-Elementen gelegt.

Grundbausteine von Projektplänen werden entsprechend der obigen Klassifikation typisiert, wobei Qualitätseigenschaften durch sog. Attributmodelle repräsentiert werden. Zur Dokumentation der Prozeßmodelle wurde in bisherigen Anwendungen die Sprache MVP-L verwendet.

Das wesentliche Prinzip, für welches die Techniken des MVP-Projekts entwickelt werden, ist die kontinuierliche Verbesserung des Entwicklungsprozesses und der Entwicklungsumgebung nach dem *Quality Improvement Paradigm (QIP)* [BaCR94].

Die Prozeßmodellierung erfolgt beim QIP im dritten Schritt auf der Grundlage der in den ersten beiden Schritten erzielten Ergebnisse. In den letzten beiden Schritten werden u. a. die Prozeßmodelle auf Verbesserungsmöglichkeiten hin untersucht, damit gemachte Erfahrungen für zukünftige Projekte erhalten bleiben. Sie werden in einer Erfahrungsdatenbank gespeichert, so daß ein projektübergreifender Verbesserungsprozeß ermöglicht wird.

1.2 Experimenteller Ansatz des SFB 501

Die Forschung in der Informatik beschäftigt sich in vielen Fällen mit der Entwicklung von neuen Systemen, Techniken oder Werkzeugen. Dabei wird oft nicht explizit gemacht, aus welchem Kontext das System, die Technik oder das Werkzeug stammt, welcher Nutzen in verschiedenen Kontexten erbracht wird bzw. welche Eigenschaften das System, die Technik oder das Werkzeug besitzt, die es besser als andere Ansätze erscheinen läßt. Eine Möglichkeit der systematischen Gewinnung von Erkenntnissen über verschiedener Systeme, Techniken oder Werkzeuge besteht in der Durchführung von Experimenten. In anderen Wissenschaften wie z. B. der Medizin oder der Soziologie gehören Experimente zum wissenschaftlichen Standardrepertoire.

Ein auf experimentell gewonnenen Erfahrungen basierender Einsatz von Systemen, Techniken oder Werkzeugen ist in Software-Entwicklungsorganisationen bisher nicht weit verbreitet [Glas94]. Vielfach werden Techniken angewandt, ohne ihre Eignung für einen bestimmten Entwicklungskontext (Erfahrung der Entwickler, Anwendungsdomäne etc.) zu kennen bzw. zu berücksichtigen. Entsprechendes Wissen wird nicht ermittelt und in weiteren Projekten verwendet. Die meisten Organisationen lernen nicht systematisch aus durchgeführten Projekten, um die gewonnenen Erkenntnisse in nachfolgenden Projekten sinnvoll umzusetzen. So gehen wertvolle Erfahrungen verloren, die zur Verbesserung des Entwicklungsprozesses und somit zur Wettbewerbsfähigkeit der Organisationen beitragen könnten.

Wesentliche Gründe für die mangelnde Bereitschaft zur Durchführung von Experimenten in der Praxis sowie die unzureichende Fähigkeit, aus durchgeführten Projekten zu lernen, sind zum einen die hohen Kosten, die mit Experimenten verbunden sind, und zum anderen

ein Zeitmangel in realen Projekten, der Lernaktivitäten nicht zuläßt. Doch hat sich bereits in einigen Software-Entwicklungsorganisationen (z. B.: NASA [NASA94], CSC [BaMc95]) die Erkenntnis durchgesetzt, daß Investitionen hierfür besser sind als das Eingehen von Risiken beim Gebrauch neuer Werkzeuge oder Techniken, von denen man keinerlei Kenntnis über die mit ihrem Einsatz verbundenen Konsequenzen hat. Um der Bedeutung des Lernvorgangs gerecht zu werden, sollte er in einer organisatorischen Struktur innerhalb einer Software-Entwicklungsorganisation institutionalisiert sein. Ein Ansatz hierzu ist die „Experience Factory“ [BaCR94], die die Analyse und Wiederverwendung von in Projekten gesammelten Erfahrungen unterstützt und von der Projektorganisation, die das Software-System erstellt, organisatorisch getrennt ist.

Es gibt verschiedene Software-Entwicklungsprozesse, mit denen qualitativ hochwertige Software-Systeme entwickelt werden können. Da es weder einen universalen Entwicklungsprozeß noch eine optimale Entwicklungsmethode gibt, muß der Entwicklungsprozeß immer in dem ihn umgebenden Kontext betrachtet werden. Die nichtrationale Auswahl von Techniken, Methoden und Werkzeugen, möglicherweise intuitiv, führt selten zu einer optimalen Lösung. Wichtig ist die systematische Auswahl und Anpassung von Software-Entwicklungsprozessen im Hinblick auf die von der Entwicklungsorganisation angestrebten Ziele. Dabei muß beachtet werden, daß der Entwicklungsprozeß von Projekt zu Projekt eine veränderliche Größe ist. Lernen, wie Technologien an die Bedürfnisse von Software-Entwicklungsorganisationen angepaßt werden, umfaßt die Durchführung von Experimenten verschiedener Typen (Fallstudien, replizierte Projekte etc.).

Bei der Entwicklung großer Systeme innerhalb des SFB sollen Softwareprodukte, Entwicklungsschritte und Erfahrungen aus abgelaufenen Projekten mit Hilfe von generischen Methoden[2] wiederverwendet werden. Dies setzt voraus, daß während der Durchführung von Software-Entwicklungsprojekten schritthaltend Informationen gesammelt werden. Solche Informationen sind u. a. Teilprodukte, Entwicklungsschritte oder Ergebnisse von Messungen mit den zugehörigen Begründungen, warum sie in dem jeweiligen Projekt verwendet bzw. durchgeführt wurden. In ersten Projekten werden solche Begründungen auf bereits vorhandenen Erfahrungen, zur Verfügung stehenden Informationen oder subjektiven Einschätzungen beruhen, in späteren Projekten auf dem dokumentierten Wissen und Erfahrungen aus vorausgegangenen Projekten.

Um den Einsatz von bestimmten Technologien innerhalb des SFB mittelfristig objektiv begründen zu können und um langfristig eine generische Methodik für ihre Auswahl und Anpassung an eine gegebene Situation zu entwickeln, werden Software-Entwicklungsprojekte innerhalb des SFB grundsätzlich als Experimente verstanden. Die Software-Entwicklung wird im Kontext des SFB als Labordisziplin verstanden. Entsprechend umfaßt die Rahmenarchitektur des SFB neben Projektorganisationen, die Software-Systeme erstellen, einen Software-Engineering-Kern (kurz: SE-Kern), in dem Informationen und Erfahrungen (in Form von Produktmodellen, Prozeßmodellen, Vorgehensmodellen, Qualitätsmodellen und Beschreibungstechniken) aus vergangenen Projekten analysiert, aufbereitet und gespeichert werden und in einem Rückkopplungsprozeß zur Verbesserung der Modelle und Beschreibungstechniken zukünftigen Projekten zur Verfügung gestellt werden.

2. Unter dem Begriff „generische Methoden“ werden im SFB alle Beschreibungs- und Generierungstechniken mit den dazugehörigen Werkzeugen zusammengefaßt, durch die eine Wiederverwendung von existierenden Softwareprodukten, Entwicklungsschritten und Erfahrungen bei der Entwicklung eines neuen Systems methodisch unterstützt wird.

Der SE-Kern spiegelt sich im Software-Engineering-Labor (kurz: SE-Labor, Teilprojekt A1) des SFB wider. Hier sammeln sich methodische Ergebnisse und sonstige Erkenntnisse des SFB an. Bereits vorhandene bzw. neu entwickelte Werkzeuge oder Techniken müssen erprobt werden, um sie in das Labor zu integrieren. Das SE-Labor institutionalisiert somit den projektübergreifenden Lernprozeß innerhalb des SFB. Die Entwicklung der prototypischen Anwendungssysteme in den Projektorganisationen erfolgt unter Benutzung des SE-Labors. Zusammen mit Teilbereich D (Prototypanwendungen) bildet das SE-Labor das experimentelle Umfeld für Entwicklungsprojekte des SFB.

Im Projektbereich B des SFB werden die methodischen Grundlagen für die Entwicklung großer Systeme erarbeitet. Hierzu gehört insbesondere die Erarbeitung von aufeinander abgestimmten Produkt-, Prozeß- und Qualitätsmodellen, die Entwicklung von generischen Methoden zur Software-Entwicklung und deren Einbettung in ein übergeordnetes Rahmenmodell.

Der Projektbereich C beschäftigt sich mit Beschreibungstechniken, die zur Erfassung aller relevanten Aspekte der im Projektbereich B zu entwickelnden Modelle und Methoden geeignet sind.

1.3 Ziele

Die Entwicklung von Modellen, Methoden und Techniken in den Projektbereichen B und C sollte einerseits auf bereits gemachten Erfahrungen beruhen, andererseits auch durch den Einsatz in zukünftigen Projekten später beeinflußt werden. Idealerweise sollte ein Modell, eine Methode oder eine Technik nicht zur Wiederverwendung aufbereitet werden, bevor sie in einem realistischen Projektkontext ausprobiert wurde. Diese Erprobung kann mit Hilfe von Experimenten erfolgen, durch die sich systematisch Erkenntnisse gewinnen lassen. Ein erster Schritt in Richtung einer erfahrungsbasierten Software-Entwicklung ist die Bereitstellung expliziter Prozeßmodelle für Software-Entwicklungsmethoden, die für den SFB 501 relevant sind. Verschiedene Standard-Prozeßmodelle existieren bereits. Bisher gibt es jedoch kaum Software-Entwicklungsprozesse, die speziell für reaktive Systeme entworfen worden sind, d. h. auf die besonderen Anfordernisse bei der Entwicklung reaktiver Systeme zugeschnitten sind. Auch ist bisher nur wenig Erfahrung dokumentiert, für welche Art von Projektkontexten diese Prozesse gültig sind.

Dieser Bericht beschreibt die Formalisierung der Software-Entwicklungsmethode SOMT (SDL-oriented Object Modeling Technique) mit MVP-L. Bei der Formalisierung wurden vorliegende informelle Beschreibungen von SOMT basierend auf Erfahrungen teilweise modifiziert bzw. um zusätzliche Prozeßinformationen ergänzt. Zusätzlich erfolgte eine erste Charakterisierung der MVP-L-Modelle anhand des modell-orientierten Schemas (Comprehensive Reuse Model) von Basili und Rombach.

Mit diesem Bericht werden folgende Ziele verfolgt:

- *Explizite Formalisierung einer Software-Entwicklungsmethode, die als Vorgabe (präskriptiv) in Fallstudien des SFB 501 bzw. zur Anleitung von Entwicklern verwendet werden kann.*

- *Beitrag zur methodischen und theoretischen Fundierung des SE-Kerns. Der Lernprozeß innerhalb des SFB bzw. allgemein innerhalb von Software-Entwicklungsorganisationen ist iterativ. Somit gehört die explizite Beschreibung von Erfahrungen über Methoden bzw. Beschreibungstechniken zu einem festen Bestandteil eines kontinuierlichen Verbesserungsprozesses.*
- *Verfeinerung und Formalisierung der Schritte von SOMT. Die Formalisierung trägt insbesondere dazu bei, verborgenes (d. h. implizites) Prozeßwissen explizit zu formulieren und somit in Form von MVP-L-Modellen als Erfahrung wiederverwendbar zu machen.*

Die Modellierung weiterer Software-Entwicklungsmethoden (OMT und NRL/SCR) kann [Pete98] entnommen werden.

1.4 Aufbau des Berichts

Der Bericht ist wie folgt aufgebaut: In Kapitel 2 werden inhaltliche Grundlagen der Software-Entwicklungsmethode SOMT beschrieben. Insbesondere wird auf die eingesetzten Techniken und deren Kombination eingegangen. In Kapitel 3 werden mögliche Projektkontexte charakterisiert, in denen das SOMT-Modell im Sinne eines Erfahrungselements Gültigkeit hat. Darauf werden der Modellierungsvorgang sowie hierbei gemachte Erfahrungen dokumentiert. Eine vollständige Darstellung des Modells in graphischer MVP-L-Notation befindet sich im Anhang. Die Darstellung des Modells in textueller Notation kann der SFB-Erfahrungsdatenbank entnommen werden.

Kapitel 2

Die Software-Entwicklungsmethode SOMT

SOMT (SDL-oriented Object Modeling Technique) ist eine Methode zur Entwicklung verteilter, reaktiver Systeme und wurde ebenso wie die zugehörige Entwicklungsumgebung SDT (SDL Design Tool) [Tele97] von der Firma Telelogic entwickelt. Grundlage für die vorliegende Arbeit war die Version SDT 3.1 [SDTM96].

Im folgenden werden zunächst die wichtigsten in SOMT verwendeten Beschreibungstechniken vorgestellt und im Anschluß daran die einzelnen Entwicklungsphasen im Überblick betrachtet.

2.1 Beschreibungstechniken in SOMT

SOMT kombiniert eine objektorientierte Analyse mit einem formalen Entwurf in SDL. Hierbei kommen im wesentlichen OMT (Object Modeling Technique) [RuBP91] bzw. UML (Unified Modeling Language) [BRuJ97], MSC (Message Sequence Chart) [ITUT96a] und SDL (Specification and Description Language) [ITUT96b] [ITUT97] zum Einsatz. Diese drei grundlegenden Techniken werden im folgenden kurz vorgestellt. Weitere optionale Techniken, wie ASN.1 (Abstract Syntax Notation One), CORBA IDL (Common Object Request Broker Architecture - Interface Description Language) oder TTCN (Tree and Tabular Combined Notation) werden in diesem Kapitel nicht behandelt und sind auch für das weitere Verständnis von SOMT nicht unbedingt notwendig. Der interessierte Leser sei auf [SDTM96], [SDTM97] und [ITUT95] verwiesen.

2.1.1 OMT/UML

OMT wird ursprünglich sowohl für die objektorientierte Analyse als auch für den objektorientierten Entwurf verwendet. Ein System wird dabei durch drei Modelle repräsentiert: das Objektmodell zur Festlegung der statischen Struktur der Objekte und ihrer Beziehun-

gen, das dynamische Modell zur Beschreibung des Systemverhaltens und das funktionale Modell zur Spezifikation von Datentransformationen. OMT wurde mit anderen objektorientierten Analysemethoden kombiniert und von der Rational Software Corporation zu UML weiterentwickelt. Die Unterschiede zwischen OMT und UML sind in SOMT jedoch nicht relevant, da nur gemeinsame Teile der beiden Techniken zum Einsatz kommen, weswegen im weiteren auch abkürzend von OMT/UML gesprochen wird.

Da in SOMT für den Entwurf die formale Beschreibungstechnik SDL verwendet wird, kommt OMT/UML lediglich für die objektorientierte Analyse zum Einsatz. Hierbei findet im wesentlichen das Objektmodell Verwendung[1].

Ein OMT/UML Objektmodell besteht aus einer Menge von Klassen, die durch Angabe ihres Namens, ihrer Attribute und ihrer Operationen beschrieben werden, sowie den Beziehungen zwischen diesen Klassen, wie z.B. Ableitungsbeziehung, Assoziation oder auch speziell Aggregation. Neben den Klassendefinitionen kann ein Objektmodell auch eine Beschreibung der vorkommenden Instanzen und ihrer Beziehungen zueinander beinhalten. Abbildung 2 zeigt eine Darstellung eines Objektmodells für ein einfaches Pingpong-Protokoll, das mit dem SDT OM-Editor erstellt wurde.

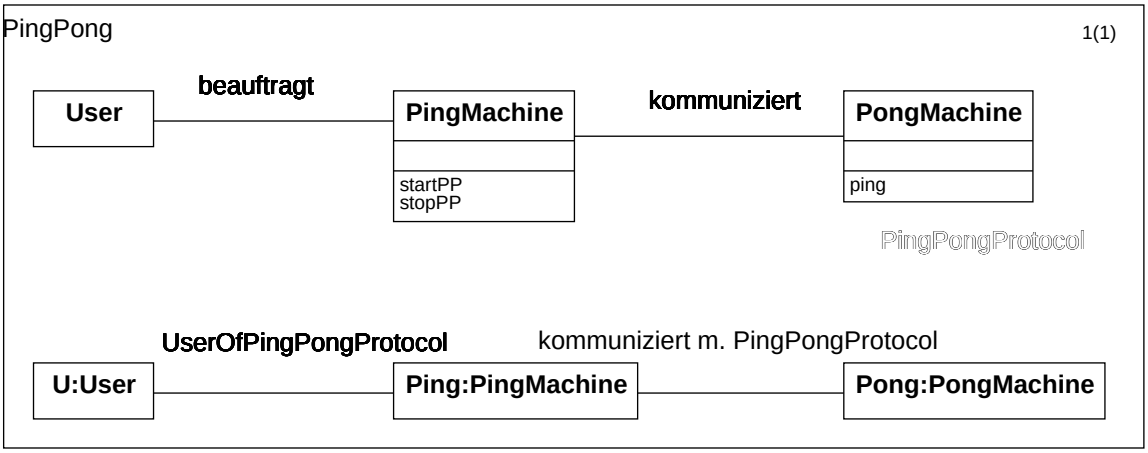

Abbildung 2: Objektmodell für das PingPong-Protokoll

2.1.2 MSC

MSC ist eine formale Technik zur abstrakten Beschreibung des Interaktionsverhaltens zwischen Systemkomponenten und ihrer Umgebung. Durch ihre graphische Notation sind MSC Diagramme intuitiv verständlich. Syntax und Semantik von MSC sind von der ITU (International Telecommunication Union) standardisiert. Die augenblicklich aktuelle Version ist MSC 96.

Ein MSC Diagramm beschreibt ein mögliches Szenario, d.h. einen möglichen Nachrichtenaustausch zwischen einem Teil der im System und seiner Umgebung vorkommenden

1. In der aktuellen Version SDT 3.2 [SDTM97] kommt zusätzlich noch die State Chart Notation von David Harel zur Beschreibung des Grobverhaltens der identifizierten Objekte bereits in der Analysephase hinzu

Instanzen. Abbildung 3 zeigt ein Beispiel für das Pingpong-Protokoll. Beschrieben wird der Nachrichtenaustausch zwischen dem Benutzer, der zur Umgebung des Systems gezählt wird, und den Systemkomponenten *Ping* und *Pong*: nach Beauftragung durch den Benutzer (Empfang des Signals *startPP*), wechselt der Ping-Prozeß von Zustand *idle* in Zustand *aktiv* und tauscht solange mit dem Pong-Prozeß Nachrichten aus, bis der Benutzer

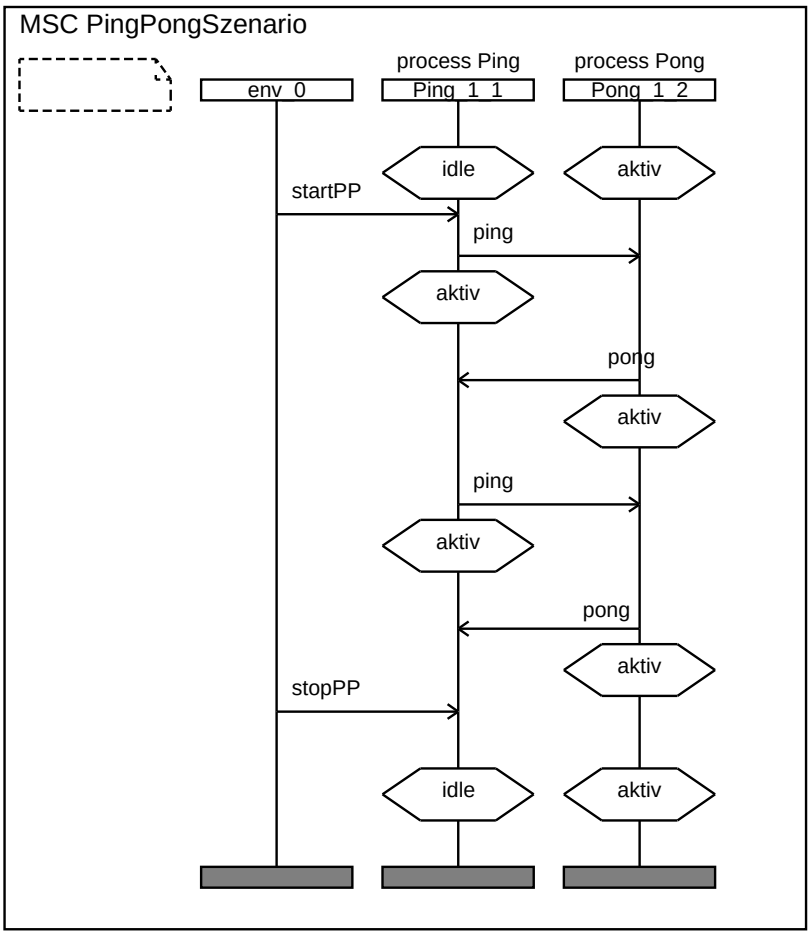

Abbildung 3: Beschreibung eines PingPong Szenarios in MSC

den Vorgang durch die Nachricht *stopPP* beendet. Die Systemumgebung, die in diesem Beispiel aus dem Benutzer besteht, wird in SDT durch die ausgezeichnete Instanz *env_0* im MSC Diagramm repräsentiert.

2.1.3 SDL

SDL ist die SOMT zugrundeliegende Basistechnik. Syntax und Semantik dieser formalen Beschreibungstechnik unterliegen ebenso wie bei MSC der Standardisierung durch die ITU. Die augenblicklich aktuelle Version ist SDL-96. SDL besitzt eine textuelle und auch graphische Darstellung. Wie bereits erwähnt kommt SDL in SOMT während der Entwurfsphase zum Einsatz. SDL eignet sich zur Beschreibung der Architektur, des Verhaltens und auch der Daten eines Systems.

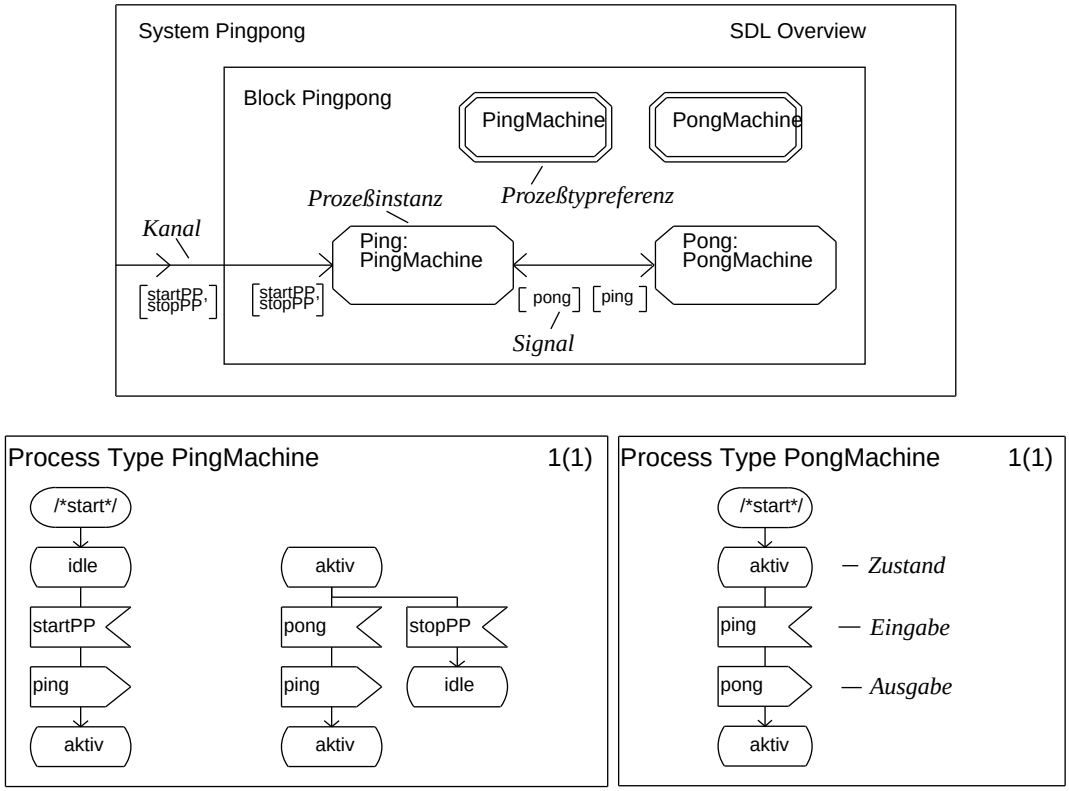

Abbildung 4: SDL Spezifikation des Pingpong Protokolls

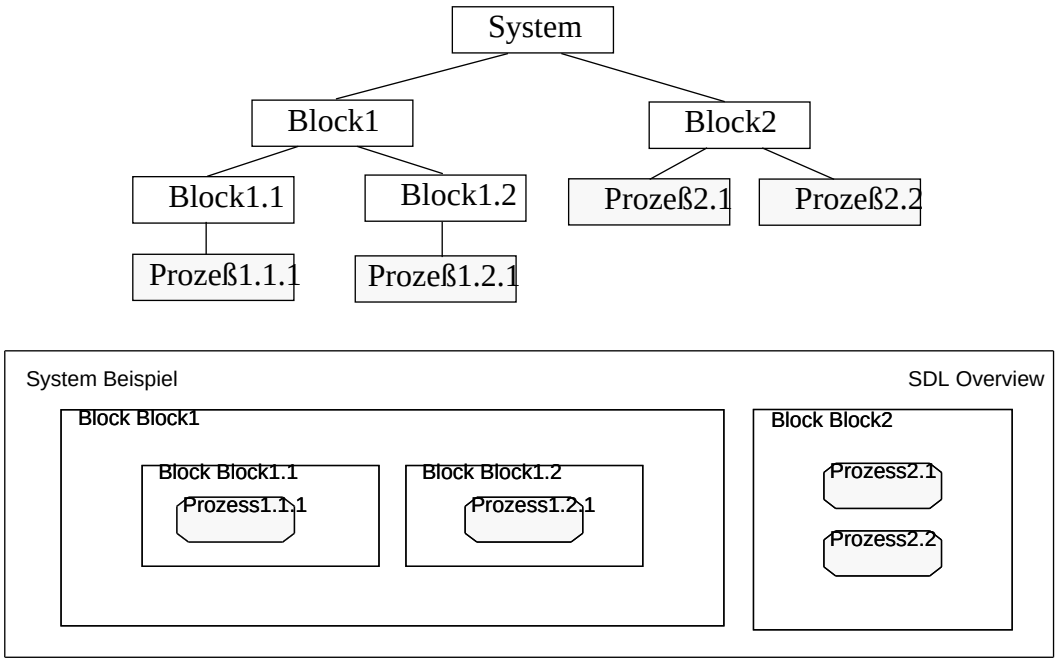

Abbildung 5: Beispiel einer SDL Blockstruktur

Das SDL zugrundeliegende mathematische Modell sind kommunizierende, erweiterte endliche Automaten. Das Verhalten eines Automaten wird dabei durch einen sog. SDL Prozeß (s. Abbildung 4) mit unendlicher FIFO-Eingangswarteschlange zum Speichern ankommender Signale dargestellt, wobei die Prozesse asynchron durch Nachrichtenaustausch kommunizieren. Die Verbindungswege sind dabei zuverlässig und reihenfolgetreu, können jedoch mit einer beliebigen aber endlichen Verzögerung behaftet sein. Prozesse können andere Prozesse dynamisch erzeugen, ein Prozeß kann sich jedoch nur selbst terminieren.

Die Daten eines SDL Prozesses, der ja einen *erweiterten* endlichen Automaten darstellt, werden als abstrakte Datentypen modelliert. Das Verhalten der Operatoren kann direkt in SDL spezifiziert werden. Neben der Möglichkeit, eigene Datentypen zu definieren, werden jedoch auch vordefinierte Datentypen wie z.B. Integer oder CharString angeboten.

Zur Strukturierung eines Systems können Prozesse zu sog. SDL Blöcken gruppiert werden, die wiederum zu übergeordneten Blöcken zusammengefaßt werden können. Abbildung 5 illustriert eine mögliche Strukturierung eines Systems, sowie die zugehörige Darstellung als SDL-Blockstruktur, wobei mögliche Verbindungswege und zugehörige Signale zwischen den einzelnen Systemkomponenten nicht eingezeichnet sind.

Bei SDL handelt es sich um eine objektorientierte Technik, die die Typisierung und Spezialisierung von SDL Systemen, Prozessen, Blöcken und Signalen erlaubt (s. Abbildung 4 für ein Beispiel von Prozeßtypen *PingMachine* und *PongMachine* mit zugehörigen Instanzen *Ping* und *Pong*). Abstrakte Datentypen können ebenfalls spezialisiert werden. Es besteht außerdem die Möglichkeiten, Bibliotheken von Typdefinitionen anzulegen und diese in verschiedenen Projekten wiederzuverwenden. Eine solche SDL-Klassenbibliothek wird als Package bezeichnet.

2.2 Entwicklungsphasen in SOMT

Die SOMT Methode basiert auf einer Kombination von objektorientierter Analyse mit SDL-basiertem Entwurf. Zusätzlich bietet die Verwendung von SDL auch noch die Möglichkeit zu einem werkzeuggestützten Testen sowie einer automatischen Prototypengenerierung.

Die im folgenden beschriebenen Aktivitäten werden alle von dem Entwicklungswerkzeug SDT unterstützt. Zwischen den erzeugten Modellen der einzelnen Aktivitäten (z.B. Analyse-Objektmodell und SDL Entwurfsmodell) können dabei Verweise angelegt werden, die eine Konsistenzüberprüfung ermöglichen und eine Verfolgbarkeit zwischen den Modellen erleichtern. So kann z.B. von einer Klasse im Objektmodell ein Verweis auf den SDL Prozeßtyp, der dieses Objekt implementiert, angelegt werden. Wird dies konsequent verfolgt, so kann z.B. überprüft werden, ob alle in der Analyse identifizierten Objekte auch im SDL Entwurf repräsentiert werden.

2.2.1 Analyse

Zu Beginn wird eine objektorientierte Analyse der Anwendungsdomäne, der Problembeschreibung und des zu entwickelnden Systems durchgeführt. SOMT unterscheidet hierbei zwischen Anforderungsanalyse, in der das System noch als Black-Box gesehen wird, und

der Systemanalyse, die die interne logische Architektur des Systems betrachtet. SDL kommt hierbei noch nicht zum Einsatz. Bei den erstellten Modellen handelt es sich vielmehr um OMT/UML Objektmodelle und Use Case-Modelle.

Ein Use Case-Modell in SOMT umfaßt eine kurze Charakterisierung der beteiligten Instanzen und beschriebenen Kommunikationsszenarien sowie eine detaillierte Beschreibung des Interaktionsverhaltens. Dies geschieht formal unter Verwendung von MSC Diagrammen oder auch informell unter Verwendung von strukturiertem Text (s. [SDTM96], Kapitel 3).

Gleichzeitig zu allen restlichen Aktivitäten wird außerdem ein sog. Wörterbuch (Data Dictionary) erstellt, das eine Definition der identifizierten Konzepte des Anwendungsgebietes darstellt, und somit für ein gemeinsames Vokabular zwischen Entwicklern, Kunden und auch Anwendern sorgt. Dieses Data Dictionary wird während nachfolgenden Aktivitäten ständig aktualisiert

Anforderungsanalyse

Ausgangspunkt für die Anforderungsanalyse ist eine informelle Anforderungsbeschreibung. Das Ziel dieser Aktivität besteht darin, die Anwendungsdomäne selbst sowie die gestellten Anforderungen zu untersuchen. Das Ergebnis wird mittels verschiedener Notationen dargestellt. Hierzu zählt ein Anforderungs-Objektmodell zur Beschreibung der Objekte des Anwendungsgebietes sowie ein Kontextdiagramm, welches das System gegenüber seiner Umgebung abgrenzt. Kommunikationsszenarien zwischen den Objekten der Umgebung und dem System werden durch ein Use Case Modell beschrieben. Hierbei berücksichtigen die Szenarien des Modells jedoch nur die Kommunikation an der Systemschnittstelle, da das System selbst noch als Black-Box betrachtet wird.

Systemanalyse

Im Gegensatz zur Anforderungsanalyse, in der das Problem selbst erfaßt werden sollte, ist das Ziel dieser Aktivität die Analyse des zu entwickelnden System selbst, das nun nicht mehr länger als Black-Box betrachtet wird. In der Systemanalyse wird die logische Architektur des Systems sowie die enthaltenen Objekte mittels objektorientierter Analyse identifiziert. Das Ergebnis wird durch ein Analyse-Objektmodell repräsentiert. Die Interaktionen zwischen den identifizierten Objekten des Systems untereinander und mit der Systemumgebung werden mittels eines Analyse-Use Case Modells in Form von MSC Diagrammen und evtl. strukturiertem Text dargestellt.

2.2.2 Entwurf

Während des Entwurfs wird sowohl die Architektur als auch das Verhalten des zu entwikkelnden Systems in SDL spezifiziert. SOMT gliedert hierzu den Entwurfsschritt weiter in System-Entwurf, in dem die Systemarchitektur festgelegt wird, und Objekt-Entwurf, in dem das Verhalten der Systemkomponenten spezifiziert wird, sowie einem anschließenden MSC basierten Test. Das Resultat ist eine korrekte, ausführbare SDL Entwurfsspezifikation.

System-Entwurf

Im System-Entwurf wird zunächst die Architektur des Systems in SDL beschrieben. Hierzu wird untersucht, in welche Komponenten das Gesamtsystem partitioniert werden kann. Richtlinien hierfür sind zum einen die Dekomposition der Systemfunktionalitäten in überschaubare, beherrschbare Teile, die Einteilung des Systems in Arbeitspakete, die von einzelnen Entwicklungsteams weiter bearbeitet werden können, aber auch die Berücksichtigung von Verteilungsaspekten und sonstigen basis-technologischen Randbedingungen. Als Eingabe hierfür dienen die Objekt- und Use Case-Modelle aus der Analysephase, die bereits eine logische Architektur festlegen. Das Resultat ist eine SDL Blockstruktur sowie eine Zuteilung der SDL Typdefinitionen zu einzelnen SDL Packages, die entweder bereits von vorangegangenen Projekten vorliegen oder neu von einem Entwicklungsteam zu erstellen sind. Weiterhin müssen noch die Schnittstellen zwischen den einzelnen identifizierten Systemkomponenten festgelegt werden. Die statische Schnittstelle wird hierbei durch Angabe von SDL Kanälen und Signalen definiert. Optional kann hierfür auch CORBA-IDL zum Einsatz kommen. Die dynamische Schnittstelle wird durch Angabe von Szenarien beschrieben. Hierfür kann entweder MSC oder TTCN verwendet werden.

Objekt-Entwurf

Ausgangspunkt für den Objekt-Entwurf ist das Objektmodell der Analysephase, die SDL Architektur des System-Entwurfs sowie die bislang erstellten Szenarien der Use Case-Modelle. Die Aufgabe besteht nun in der Zuteilung der Objekte des Objektmodells zu den Architekturkomponenten des System-Entwurfs und die Ausarbeitung ihrer Verhaltensbeschreibung in SDL. Aktive Objekte, d.h. Objekte mit eigenem Kontrollfluß, werden dabei in SDL Prozesse bzw. Prozeßtypen abgebildet, passive Objekte in abstrakte Datentypen. Es werden zunächst nur die wichtigsten Szenarien betrachtet und das Verhalten der Objekte soweit spezifiziert, daß sie diese ausgewählten Szenarien realisieren. Im Anschluß daran werden die restlichen Szenarien eingebaut, wobei SOMT jedoch keine Richtlinien vorgibt, wie die Anpassung und Erweiterung bestehender SDL Typen vorgenommen werden sollte, um sie um die neuen Funktionalitäten zu ergänzen. Nach jeder Iteration sollte durch einen (Entwurfs-) Test überprüft werden, ob die bislang betrachteten Szenarien auch wirklich vom aktuellen Entwurf erfüllt werden.

(Entwurfs-) Test

Einer der Hauptvorteile eines formalen Entwurfs in SDL ist, daß bereits vor der Erstellung der eigentlichen Implementierung ein werkzeugunterstütztes Testen durchgeführt werden kann. Dies liegt darin begründet, daß eine formale Entwurfsspezikation in SDL bereits eine ausführbare Beschreibung des zu implementierenden Systems darstellt und somit eine Simulation leicht durchgeführt werden kann. Die Erkennung von Entwurfsfehlern bereits in frühen Stadien der Systementwicklung wird dadurch ermöglicht.

Die Testfälle, gegen die der Entwurf validiert werden soll, stammen hauptsächlich aus Szenarien, die in den vorangegangenen Entwicklungsschritten erstellt wurden. Es besteht aber auch die Möglichkeit, daß wie teilweise im Telekommunikationsbereich üblich standardisierte Test-Suites vorgegeben sind. Weiterhin gibt es gerade im SDL Bereich auch Forschungsansätze, die sich mit der automatischen Testfallgenerierung beschäftigen. Zur Beschreibung der Testfälle kann außer MSC auch TTCN verwendet werden. Es gibt ein Werkzeug ITEX der Firma Telelogic, welches die Erstellung und das Ausführen von

TTCN Test-Suites unterstützt und direkt mit SDT zusammenarbeitet. ITEX ist jedoch im SE-Labor nicht installiert, so daß im weiteren nur MSC basiertes Testen betrachtet wird.

Die einfachste Art, einen SDL Entwurf gegen einen MSC Testfall zu validieren, ist die Simulation des Entwurfs interaktiv mit einem Simulationswerkzeug durchzuführen, dabei jeweils die vorgeschriebenen Eingaben manuell zu tätigen und zu überprüfen, ob sich das System wie erwartet verhält. SDT bietet hierzu eine komfortable, graphische Simulationsoberfläche. Komfortabler dagegen ist das automatische Testen auf Basis der Zustandsraumexploration. Hierbei wird dem Testwerkzeug die SDL Entwurfsspezifikation und ein MSC Testfall als Eingabe übergeben. Durch Zustandsraumexploration wird nun automatisch überprüft, ob das vorgegebene MSC mit der SDL Spezifikation konsistent ist.

2.2.3 Implementierung

Ziel dieser Aktivität ist die Erstellung ausführbarer Software, wobei dieser Vorgang im besonderen von der Zielplattform beeinflußt wird. In einem ersten Schritt wird aus der formalen Entwurfsspezifikation automatisch Kode generiert. Hierzu muß der SDL Entwurf zunächst partitioniert werden, um festzulegen, welche Teile als separate Laufzeitmodule angelegt werden sollen. Die eigentliche Kodegenerierung geschieht unter Verwendung des Cadvanced-Compilers des SDT Tools.

Um eine Einbettung des generierten Kodes mit der Umgebung zu erreichen, müssen sog. Umgebungsfunktionen definiert werden. Diese legen fest, wie SDL Signale, die vom System an die Umgebung gesendet werden, in entsprechende Betriebssytemdirektive umgesetzt werden und umgekehrt. Diese Funktionen sorgen also dafür, daß das generierte System überhaupt mit seiner Umgebung kommunizieren kann.

Nach der Kodegenerierung und Einbettung der Implementierung in die Umgebung mittels Umgebungsfunktionen ist abschließend noch zu testen, ob das generierte System sich wie gewünscht in der einbettenden Umgebung verhält. Da dies abhängig ist von der Testunterstützung der Zielumgebung, wird dieser Vorgang von SOMT nicht weiter festgelegt und auch von SDT nicht unterstützt.

Kapitel 3

Die Modellierung

Ziel dieses Kapitels ist es, das SOMT-Prozeßmodell zu charakterisieren, den Modellierungsvorgang und hierbei gemachte Erfahrungen zu beschreiben, sowie einzelne Bestandteile des Prozeßmodells im Detail zu erläutern.

3.1 Einleitung

Die Modellierung wird anhand folgender Kriterien dargestellt, die vorab kurz erläutert werden:

- *Quelle*
  Es wird die Literaturangabe der Vorlage von SOMT und der Erfahrungen angegeben, die zur Beschreibung der Methode und zur Modellierung der MVP-L-Modelle herangezogen wurden.
- *Kontext*
  Der Kontext der Software-Entwicklungsmethode SOMT wird mittels des modellorientierten Charakterisierungsschemas von Basili und Rombach [BaRo91] beschrieben. Das Schema wird deshalb vorab kurz erklärt:

  In [BaRo91] werden vier Annahmen über die Software-Wiederverwendung gemacht:

  (a) Jegliche Erfahrung kann wiederverwendet werden.

  (b) Wiederverwendung erfordert typischerweise einige Änderungen am wiederzuverwendenen Objekt.

  (c) Es bedarf einer Analyse, um zu entscheiden, ob und wann Wiederverwendung sinnvoll ist.

Wiederverwendung muß in den Software-Entwicklungsprozeß integriert sein.

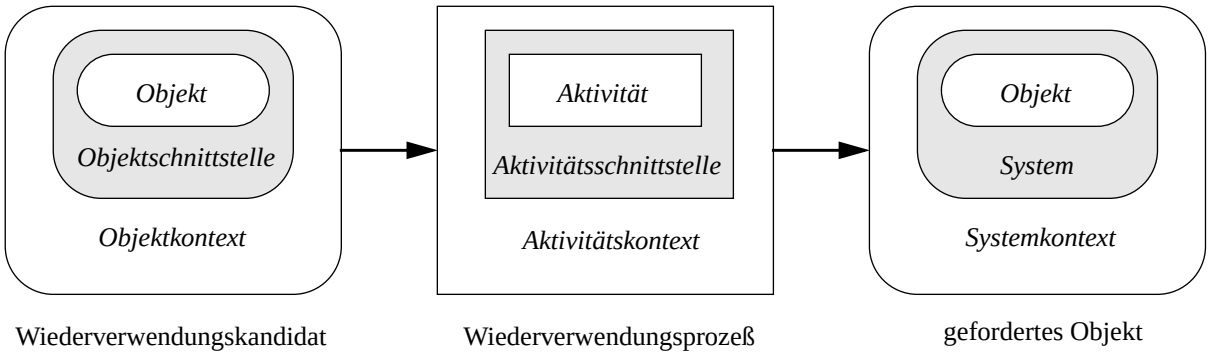

Abbildung 6: Modell-orientiertes Schema [BaRo91]

Basierend auf diesen vier Annahmen stellen Basili und Rombach folgende vier Anforderungen an ein Charakterisierungsschema:

(a) Das Schema muß auf alle Arten von Wiederverwendungsobjekten anwendbar sein.

(b) Es muß sich für die Modellierung sowohl des Wiederverwendungskandidaten als auch der Wiederverwendungsanforderungen eignen.

(c) Das Charakterisierungsschema muß sich für die Modellierung des Wiederverwenungsprozesses selbst eignen.

(d) Das Schema muß so gestaltet und beschaffen sein, daß es einfach an spezifische Projektanforderungen und -eigenschaften angepaßt werden kann.

Um ein Charakterisierungsschema zu bekommen, daß diese Anforderungen erfüllt, wurde der modell-orientierte Schematyp entwickelt. Wie in Abbildung 6 (eine Übersetzung von Abbildung 4 in [BaRo91]) zu sehen ist, gliedert es sich in drei Teile: Wiederverwendungskandidaten, Wiederverwendungsprozeß und das geforderte Objekt.

Der Wiederverwendungskandidat stellt Erfahrung dar, die im Laufe von Software-Entwicklungen gewonnen, als Wiederverwendungspotential eingestuft und in einer Erfahrungsdatenbank - zur Wiederverwendung aufbereitet - abgelegt wurde. Die Wiederverwendungsanforderungen spezifizieren die geforderten Objekte für die Wiederverwendung in einem Projekt. Soll es zu einer Wiederverwendung eines Kandidaten aus der Erfahrungsdatenbank kommen, muß eine angepaßte Form des Wiederverwendungskandidaten den Anforderungen an ein Objekt genügen. Der Wiederverwendungsprozeß paßt den Wiederverwendungskandidaten anhand der Wiederverwendungsanforderungen an das geforderte Objekt an.

Alle drei Teile des Schemas sind jeweils dreifach verfeinert: Ein Teil der Beschreibung des Wiederverwendungskandidaten bildet das Objekt selbst. Da der Wiederverwendungskandidat meist nur eine Komponente eines größeren Objektes ist, er aber als einzelne Einheit wiederverwendet werden soll, wird in der Objektschnitt-

stelle beschrieben, wie der Wiederverwendungskandidat mit anderen Objekten in Beziehung steht. Der Objektkontext dient der Beschreibung der Umgebung, für den der Wiederverwendungskandidat ehemals entwickelt wurde.

Beim Wiederverwendungsprozeß wird die Aktivität selbst, ihre Schnittstelle und ihr Kontext beschrieben. Die Wiederverwendungsaktivität muß in den Software-Entwicklungsprozeß integriert werden; die benötigten Mittel sind in der Aktivitätsschnittstelle zusammengestellt. Im Aktivitätskontext wird dokumentiert, wie die Unterstützung zum Austausch der Erfahrungen über Projektgrenzen hinweg gehandhabt wird.

Die Beschreibung des geforderten Objektes umfaßt die Attribute des Objektes, das System, in welches das zu transformierende Objekt eingebunden werden soll, und den Systemkontext, in welchem das System entwickelt werden soll.

Für alle drei Teile des Schema werden im folgenden die in [BaRo91] vorgestellten Kriterien zur Charakterisierung dargestellt. Sie sind in Abbildung 7 - eine Zusammenfügung der Abbildungen 5a und 5b aus [BaRo91] - zu sehen. Für den Wiederverwendungskandidaten und das geforderte Objekt selbst gibt es sechs Kriterien, um ihre Eigenschaften näher zu beschreiben. Sie sind [BaRo91] entnommen und werden im folgenden der Reihe nach erläutert:

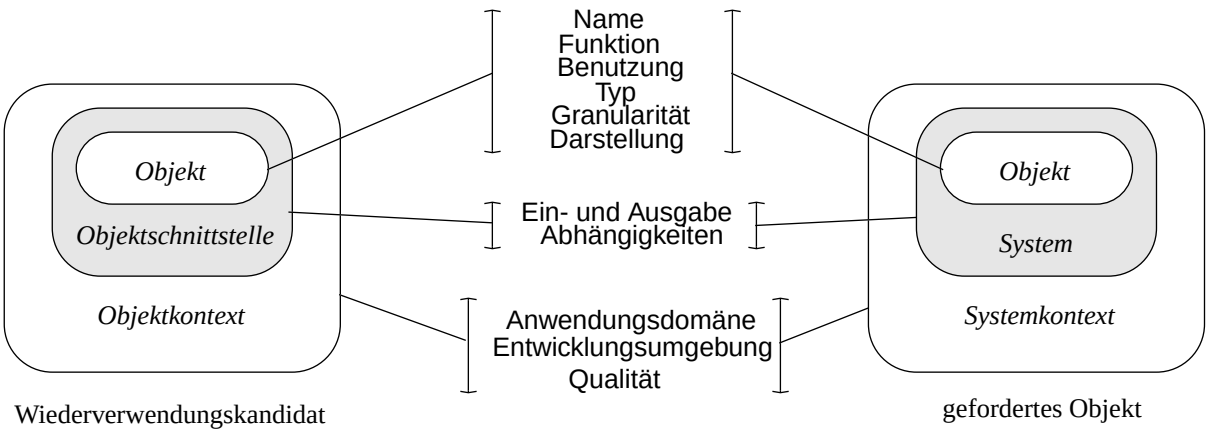

Abbildung 7: Kriterien für Wiederverwendungskandidaten und -anforderungen

(a) Name:
Der Name des Objektes.

(b) Funktion:
Die formale Spezifikation oder der Zweck des Objektes (z. B.: Druckerwarteschlange, geschätzte Produktionskosten, Sensor-Überwachungssystem).

(c) Benutzung:
Die Art und Weise, wie das Objekt verwendet werden kann (z. B.: Produkt, Prozeß, Erfahrung).

(d) Typ:
Die Art des Objektes (z. B.: Anforderungsdokument, Programmkode, Prozeßmodell).

(e) Granularität:
Der Gültigkeitsbereich des Objektes (z. B.: Systemebene, Prozedur, Lebenszyklus).

(f) Darstellung:
Die Repräsentation des Objektes (z. B.: formales mathematisches Modell, formale Sprache, informelle Richtlinien).

Für die Beschreibung der Schnittstelle des Wiederverwendungskandidaten und des Systems vom geforderten Objekte werden zwei Kriterien benutzt:

(a) Ein- und Ausgabe:
Die externen Ein- und Ausgabe-Abhängigkeiten, die es als selbständiges alleinstehendes Objekt besitzt (z. B.: Globale Daten, formale und aktuelle Parameter einer Prozedur, Variablen eines Kostenmodells).

(b) Abhängigkeiten:
Die benötigten Annahmen und Abhängigkeiten, um das Objekt zu verstehen (z. B.: Annahmen über die Benutzerqualifikation, wie z. B. die Fähigkeit, das Spezifikationsdokument zu lesen).

Die drei folgenden Kriterien dienen dem Charakterisieren des Kontextes des Wiederverwendungskandidaten und des Systemkontextes des geforderten Objektes:

(a) Anwendungsdomäne:
Die Klasse, für die das Objekt entwickelt wurde bzw. werden soll (z. B.: Gebäudeautomationssystem, kaufmännische Software für mittelständische Betriebe, Betriebssystem).

(b) Entwicklungsumgebung:
Die Umgebung in der das Projekt verwirklicht wurde bzw. verwirklicht werden soll.

(c) Qualität:
Die Qualität, die das Objekt vorweisen kann bzw. soll (z. B.: Grad der Fehlertoleranz, Grad der Sicherheit, Benutzerfreundlichkeit, Zuverlässigkeit).

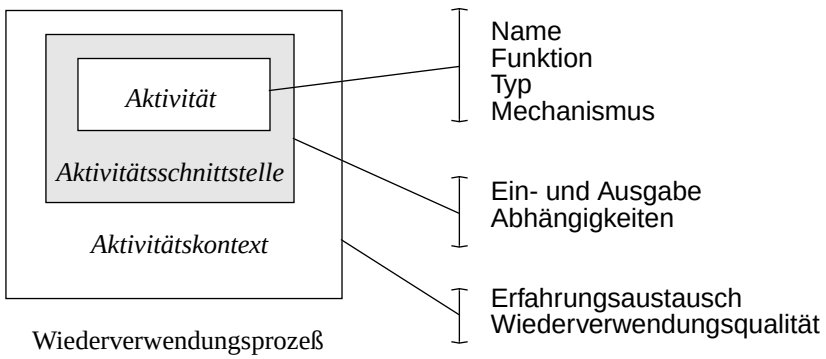

Abbildung 8: Kriterien für den Wiederverwendungsprozeß [BaRo91]

Auch die Beschreibung des Wiederverwendungsprozesses wird mit Kriterien verfeinert, wie in Abbildung 8 (eine Anlehnung an Abbildung 5c in [BaRo91]) zu sehen ist. Für die Aktivität sind vier Kriterien vorgesehen:

(a) Name:
Name der Aktivität.

(b) Funktion:
Die Funktion, die von der Aktivität durchgeführt wird (z. B.: Wähle Kandidaten, die bestimmte Eigenschaften der Wiederverwendungsanforderungen erfüllen).

(c) Typ:
Die Art der Aktivität (z. B.: Erkennen, Ändern).

(d) Mechanismus:
Die Verwirklichung der Aktivität (z. B.: Wonach wird identifiziert, nach dem Namen, nach der Funktion?).

Zwei Kriterien dienen der Verfeinerung der Schnittstelle der Wiederverwendungsaktivität:

(a) Ein- und Ausgabe:
Die Schnittstelle zwischen den Wiederverwendungsaktivitäten und der Software-Entwicklungsumgebung (z. B.: Im Falle der Erkennung: Beschreibung der Anforderungen für die Wiederverwendung / Menge von Wiederverwendungskandidaten).

(b) Abhängigkeiten:
Weitere implizite Annahmen und Abhängigkeiten an Daten und Informationen der Software-Entwicklungsumgebung (z. B.: Ausführungszeitpunkt der Wiederverwendungsaktivität relativ zum Beginn des Entwicklungsprozesses).

Die folgenden beiden Kriterien sollen den Kontext der Aktivität genauer beschreiben:

(a) Erfahrungsaustausch:
Die unterstützten Mechanismen zum Austausch von Erfahrungen (z. B.: Erfahrungsdatenbank, Personen).

(b) Wiederverwendungsqualität:
Die Qualität jeder Wiederverwendungstätigkeit (z. B.: hohe Zuverlässigkeit, Korrektheit, durchschnittlicher Durchsatz).

- *Beschreibung der Produkte*
Es werden alle Produkte dargestellt, die in der MVP-L-Modellierung der Methode SOMT produziert, konsumiert oder modifiziert werden. Alle Produkte werden in einer Tabelle aufgelistet, aus der auch Aggregationen ersichtlich sind. Der Grad der Abstraktion nimmt in der Tabelle nach links hin zu. Als Beispiel ist in Tabelle 1 das Produkt ‘Models’ eine Aggregation von den beiden Produkten ‘Object_model’ und ‘Dynamic_model’. Ist für ein Modell keine Verfeinerung vorhanden, wird dies durch ‘./.’ gekennzeichnet. In Tabelle 1 besitzt als Beispiel das Produkt ‘Problem’ keine Verfeinerungen.

Problem	./.
Models	Object_model
	Dynamic_model

Tabelle 1: Beispiel einer Produkttabelle

- *Beschreibung der Prozeßmodelle*
  Alle Prozeßmodelle der MVP-L-Modellierung von SOMT werden kurz beschrieben und die durch die Prozeßmodelle repräsentierten Aktivitäten werden erklärt. Nähere Angaben zu den Prozessen sind in der referenzierten Literatur und unter dem Stichpunkt ‘Quelle’ zu finden. In einer Tabelle werden die in MVP-L modellierten Prozeßaggregationen übersichtlich dargestellt, wobei der Abstraktionsgrad nach links zunimmt. Sollte für ein Prozeßmodell keine Verfeinerung vorgesehen sein, steht statt einem Modellnamen ‘./.’ in der Tabelle. Die grafischen Darstellungen der MVP-L-Prozeßmodelle befinden sich im Anhang A.1.
- *Erfahrungen mit der Vorlage*[1]
  Unter dem Stichpunkt Erfahrungen mit der Vorlage wird beschrieben, welche und in welcher Form die für die Modellierung in MVP-L nötigen Informationen aus der Beschreibung von SOMT hervorgeht. Außerdem werden Unklarheiten und Fehler der Vorlage angegeben.
- *Erfahrungen mit MVP-L*
  Es wird berichtet, inwieweit die Konzepte von MVP-L zur Darstellung des Entwicklungsprozesses von SOMT geeignet ist.
- *Bisherige Erfahrungen im SFB 501*
  Hier wird beschrieben, welche Erfahrungen bereits im Sonderforschungsbereich 501 (SFB 501) mit SOMT gesammelt wurde.
- *Hinweise zur Anpassung an verschiedene Kontexte*
  Es wird dargestellt, wie die Methode an bestimmte Kontexte anzupassen ist. Dafür werden die Einflußfaktoren und Variationsparameter für die nötigen Anpassungen dargelegt, wenn sie aus der Vorlage hervorgehen. Zusätzlich wird beschrieben, wie die Anpassungen vorzunehmen sind.
- *Modifikation des Originals*
  Es wird beschrieben, welche Änderungen aus welchem Grund bei der Modellierung in MVP-L gegenüber der Vorlage gemacht wurden.

3.2 Quelle

Die Vorlage des Modells wurde dem ‘Online Manual SDT 3.1 Methodology Guidlines’ [SDTM96] entnommen. In ‘Combining Object-Oriented Analysis and SDL Design’ [EkAn96] werden bezüglich der für die Formalisierung der Methode relevanten Aspekte die gleichen Aussagen getroffen; allerdings anders dargestellt. ‘Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL’ [GeRö96] und Erfahrungen in

1. Unter Vorlage der Methode wird das Dokument [SDTM96] verstanden, das die Methode beschreibt und Ausgabgspunkt für die Modellierung in MVP-L war.

der Anwendung der Methode in der AG Rechnernetze des Fachbereichs Informatik der Universität Kaiserslautern führten zu Modifikationen am MVP-L-Modell.

3.3 Kontext

Nach dem Handbuch zu SDT (SDL Design Technique) [SDTM96] und Bræk und Haugen [BrHa93] eignen sich SOMT und SDL zur Entwicklung von verteilten, reaktiven und Echtzeitsystemen.

SDL ist nur bedingt zum Entwerfen von Echtzeitsystemen im Sinne dieser Arbeit geeignet, da die Kommunikation im System mittels Signalen modelliert wird, deren Weg vom Sender zum Empfänger nach Definition eine unbestimmte Zeit in Anspruch nimmt. Damit also keinen zeitlichen Bedingungen unterworfen werden kann.

Zur Charakterisierung wurde das modell-orientiere Schema von Basili und Rombach [BaRo91] verwendet. Der Wiederverwendungsprozeß wird nicht betrachtet, da keine Angaben vorlagen.

Im folgenden wird jeweils das Objekt selbst mit folgenden Kriterien beschrieben: Name, Funktion, Benutzung, Typ, Granularität und Darstellung. Für die Objektschnittstelle wird die Ein- und Ausgabe und die Abhängigkeiten betrachtet. Der Kontext für das Objekt wird durch die Kriterien Anwendungsdomäne, Entwicklungsumgebung und Qualität betrachtet. Alle Kriterien sind in Abschnitt 3.1 unter Punkt ‘Kontext’ kurz erläutert.

(1) Objekt
   (a) Name: SOMT
   (b) Funktion: Unterstützung beim Entwickeln von Software nach objektorientierten Grundsätzen und einem SDL-basiertem Entwurf
   (c) Benutzung: formale Software-Entwicklungsmethode
   (d) Typ: Projektplan mit Prozeß- und Produktmodellen
   (e) Granularität: integrierte Methode, insbesondere Anforderungsanalyse, Entwurf und Implementierung
   (f) Darstellung: Prozeßmodellierungssprache MVP-L

(2) Schnittstelle
   (a) Eingabe: Textuelle Beschreibung der Anforderungen
   (b) Ausgabe: verschiedene Dokumente, Programmkode in C
   (c) Abhängigkeiten: setzt Wissen über MSC, SDL, Implinks, Kontext-Diagramme, TTCN und eventuell C voraus

(3) Kontext
   (a) Anwendungsdomäne: verteilte und reaktive Systeme
   (b) Entwicklungsumgebung: SDT
   (c) Qualität: keine Angabe

3.4 Beschreibung der Produktmodelle

Die im folgenden kurz beschriebenen Produkte von SOMT sind in der Tabelle 2 dargestellt. Die Produktaggregationen sind nicht der Vorlage entnommen worden, sondern lediglich zur Übersichtlichkeit des MVP-L-Modells vorgenommen worden.

External_textual_requirements	./.
Requirements_models	Requirements_object_model
	Requirements_use_case_model
	System_operations
	Textual_requirements_model
Data_dictionary	./.
System_analysis_models	Analysis_object_model
	Analysis_use_case_model
	Textual_analysis_documentation
Design_models	Textual_design_documentation
	Architecture_definitions
	Static_interface_definitions
	Design_use_case_model
	Test_model
	SDL_design
	Design_module_structure
Application	Partitioned_sdl_model
	Adapted_sdl_model
	Code
	Test_reports

Tabelle 2: Produkte von SOMT mit Verfeinerungen

3.4.1 External_textual_requirements

Dieses Dokument enthält die vom Kunden aufgestellten Anforderungen und Vorstellungen vom gewünschten System. Die Beschreibungen in diesem Dokument sind oftmals informell und unvollständig.

3.4.2 Requirements_models

Das Produkt ‘Requirements_models’ faßt alle in der Anforderungsanalyse erstellten Produkte zu einem zusammen.

(1) Requirements_object_model
Das ‘Requirements_object_model’ dient der Dokumentation von Objekten, Beziehungen, Attributen und Operationen in der Anwendungsdomäne, die zur Entwicklung des Systems von Bedeutung sind. Mittels Diagrammen stellt es die logische Struktur der Daten und Informationen, die Umgebung und den Kontext des Systems dar. Damit dient es als gemeinsame Kommunikationsgrundlage zwischen den Benutzern und den Entwicklern.

(2) Requirements_use_case_model
Mit Hilfe von Use Cases wird versucht die Anforderungen an das System aus Sicht des Benutzers zu erfassen. Wobei es dem Benutzer selber möglich sein sollte, die Use Cases zu verstehen und sie auf ihre Richtigkeit hin zu prüfen. Hauptzweck der Use Cases ist es, zu gewährleisten, daß die Vorstellungen der Benutzer richtig erfaßt werden.
In einem Use Case wird ein Benutzungsszenario dargestellt. Es wird beschrieben, wie ein möglicher abstrakter Nutzer das zukünftige System verwendet. Nutzer kann dabei zum Beispiel eine Person, ein Computer, ein Sensor oder ähnliches sein. Bei einem Gebäudeautomationssystem wäre ein mögliches Szenario, daß eine zugangsberechtigte Person das Haus betritt. Ein weiter Use Case wäre, daß eine nichtzugangsberechtigte Person versucht, sich Zugang zum Gebäude zu verschaffen.
In SOMT besteht ein Use Case Modell aus drei Teilen: eine Liste von Akteuren, eine Liste von Use Cases und der Beschreibung der Use Cases. Beschrieben werden die Use Cases entweder textuell oder mittels MSC.

(3) System_operations
In einer ‘System_operation’ wird genau definiert, wie das System auf Ereignisse reagieren soll. Es wird demnach beschrieben, wie das System auf Grund von eintretenden Ereignissen Zustände verändern oder Ausgaben erzeugen soll. ‘System_operations’ werden vorwiegend in Form von strukturiertem Text erfaßt.

(4) Textual_requirements_model
Dieses Modell besteht aus Textdokumenten, in denen Anforderungen festgehalten werden, die nur schwer mit den anderen Modellen der ‘Requirements_analysis’ beschrieben werden können. Ein typisches Beispiel sind nichtfunktionale Anforderungen, wie Zuverlässigkeit oder Leistungsaspekte.

3.4.3 Data_dictionary

In einer textuellen Liste werden alle im Entwurfsprozeß definierten Konzepte (Klassen, Objekte, Beziehungen usw.) festgehalten. Es definiert somit ein Vokabular für die Entwickler und Benutzer des Systems. Als Beispiel werden alle relevanten Objekte der Domäne in dem Data Dictionary festgehalten. Zu jedem Eintrag im Data Dictionary gehört jedesmal ein Name und eine kurze Erläuterung zum Eintrag.

3.4.4 System_analysis_models

Dieses aggregierte Produkt dient lediglich dem Zusammenfassen der in dem Prozeß ‘System_analysis’ produzierten Produkte.

(1) Analysis_object_model
Mit diesem Modell soll die Architektur des Systems beschrieben werden. Im Gegensatz zum ‘Requirements_object_model’ wird die interne Objektstruktur betrachtet. Außerdem wird dargelegt, wie das System auf unterschiedlichen Abstraktionsniveaus in Teilsysteme aufgeteilt werden kann.

(2) Analysis_use_case_model
Im ‘Analysis_use_case_model’ werden die dynamischen Aspekte der Systemzerlegung festgehalten. Der Hauptaugenmerk wird dabei auf die interne Kommunikation der einzelnen Teilsysteme untereinander gelegt.

(3) Textual_analysis_documentation
Als Gegensatz zu den sehr formalen Beschreibungen der Use Cases und des Objektmodels sind textuelle Beschreibungen wichtig, in denen nichtfunktionale Anforderungen, wie Leistungsaspekte der Architektur, erfaßt werden können.

3.4.5 Design_models

Durch das Produkt ‘Design_models’ werden alle in dem ‘Design’-Prozeß produzierten Produkte zu einem aggregiert.

(1) Textual_design_documentation
Da nicht alle statischen und dynamischen Aspekte des Entwurfs mit SDL und Use Cases beschrieben werden können, bedarf es Dokumente in denen solche Aspekte textuell erfaßt werden können.

(2) Architecture_definitions
In der ‘Architecture_definitions’ werden die Blockdiagramme festgehalten, die beim SDL-Entwurf ein System in eine Blockhierarchie einteilt.

(3) Static_interface_definitions
Die in der ‘Architecture_definitions’ vorgegebenen SDL-Blöcke kommunizieren laut SDL-Spezifikation lediglich mittels Signalen. Dabei wird bei der Beschreibung der ‘Static_interface_definitions’ nur die zu übertragenden Daten bezeichnet. Anforderungen an den Ablauf werden hier nicht bestimmt.

(4) Design_use_cases
Die ‘Design_use_cases’ beschreiben die dynamischen Aspekte der Block-Schnittstellen. Dabei werden die Use Cases der Systemanalyse soweit verfeinert, daß sie zu den Beschreibungen der ‘Static_interface_definitions’ passen.

(5) Design_module_structure
Die ‘Design_module_structure’ spiegelt die Programmkode-Module wieder, aus der die Anwendung bestehen wird. Dabei sollten Aspekte wie zum Beispiel Wiederverwendung und Kodegenerierung eine Rolle bei der Aufteilung in Module spielen.

(6) Test_model
Die meisten Testfälle kommen von den Use Cases der Systemanalyse. Es wird versucht, die Testfälle so zu gestalten, daß das Testen automatisch ausgeführt werden kann.

(7) SDL_design
Das ‘SDL_design’ enthält die Beschreibung des gesamten Systementwurfs in SDL.

3.4.6 Application

Im Produkt ‘Application’ sind alle bei der Implementierung erstellten Produkte zusammengefaßt.

(1) Partionted_sdl_design
Diese Beschreibung besteht aus Programmteilen, die für sich alleine ausgeführt werden können.

(2) Adapted_sdl_design
Der an seine Umgebung angepaßte SDL-Entwurf wird durch das ‘Adapted_sdl_design’ beschrieben.

(3) Code
Der ‘Code’ enthält schließlich das gesamte System in Form von Programmkode.

(4) Test_reports
In den ‘Test_reports’ sind die gesamten Ergebnisse der Tests aufgezeichnet.

3.5 Beschreibung der Prozeßmodelle

In Tabelle 3 ist die Hierarchie der SOMT-Prozeßmodelle aufgezeigt, die im folgenden vorgestellt werden.

3.5.1 Requirements_analysis

Im Prozeß ‘Requirements_analysis’ (dt. Anforderungsanalyse) werden die Anwendungsdomäne erfaßt und analysiert, sowie die Benutzeranforderungen an das System erstellt. Dabei wird das zu betrachtende System als Black-Box angesehen und lediglich die Objekte und Konzepte außerhalb des Systems oder an seinen Grenzen berücksichtigt.

Begonnen wird die Anforderungsanalyse mit dem Prozeß ‘Create_data_dictionary_and_structure_requirements’. Anschließend werden die Prozesse ‘Create_requirements_object_model’ und ‘Create_requirements_use_cases’ solange iterativ durchlaufen, bis die Anforderungen erfaßt sind. Zum Schluß wird noch ein Konsistenztest durchgeführt. Damit ist diese Phase aber nur vorübergehend abgeschlossen, da in einer der späteren Prozesse festgestellt werden kann, daß weitere Anforderungen erfaßt werden müssen. In Anhang A.1 ist der MVP-L-Prozeß grafisch dargestellt.

(1) Create_data_dictionary_and_structure_requirements
Während dieses Prozesses werden alle Anforderungen gesammelt und weitere verfügbare Informationen zum Beispiel über die Anwendungsdomäne oder die potentiellen Nutzer des Systems eingeholt. Dies beinhaltet auch die Strukturierung und Vervollständigung der externen informellen Anforderungen (‘External_textual_requirements’).
Gleichzeitig wird in dieser Phase ein ‘Data Dictionary’ angelegt, das unter ande-

rem eine Liste mit Akteuren (z. B. Rollen von Benutzern), Use Cases und wichtigen Konzepten der Anwendungsdomäne enthält.

(2) Create_requirements_object_model
In diesem Prozeß werden die relevanten Objekte, Beziehungen der Objekte untereinander, ihre Attribute und die benötigten Operationen bestimmt und diese in einem Objektmodell grafisch dargestellt.

Requirements_analysis	Create_data_dictionary_and_structure_requirements
	Create_requirements_object_model
	Create_requirements_use_cases
	Requirements_consistency_checks
System_analysis	Create_analysis_object_model
	Create_analysis_use_cases
	Analysis_consistency_checks
Design	Create_architecture
	Create_design_module_structure
	Create_static_interfaces
	Create_design_use_cases
	Object_design
	Design_testing
	Design_consistency_checks
Implementation_	Partition
	Adaption
	Generation_of_code
	Testing

Tabelle 3: Prozesse von SOMT

(3) Create_requirements_use_cases
Es werden Use Cases (Benutzerszenarien) ermittelt und optimiert. Beschrieben werden Use Cases in Form von Text oder als MSC.

(4) Requirements_consistency_checks
Als Abschluß der ‘Requirements_analysis’ werden die Anforderungen, die erstellten Modelle und die Use Cases auf ihre Konsistenz hin überprüft.

3.5.2 System_analysis

Dieser Prozeß dient der Analyse des zu erstellenden Systems. Der Aufbau des Systems und die in der Anwendungsdomäne gefundenen Objekte werden beschrieben, wenn diese zur Modellierung der geforderten Funktionalität benötigt werden.
Die ersten beiden Prozesse werden mehrmals iterativ durchlaufen bis die gewünschten

Produkte komplett erstellt sind. Eine grafische Repräsentation der ‘System_analysis’ befindet sich in Anhang A.1.3.

(1) Create_analysis_object_model
Es wird ein Objektmodell erstellt. Zu diesem Zweck werden relevante Objekte, ihre Attribute und Operationen und Verbindungen von Objekten über Klassen hinweg ermittelt. Vor allem wird das Objektmodell der ‘Requirements_analysis’ verfeinert.

(2) Create_analysis_use_cases
Die Use Cases aus der ‘Requirements_analysis’ werden verfeinert, um das Objektmodell aus ‘Create_analysis_object_model’ testen zu können. Es werden aber auch zusätzliche Use Cases erzeugt, um das Testen bestimmter Systemeigenschaften zu unterstützen.

(3) Analysis_consistency_checks
Am Ende der ‘System_analysis’ werden alle erstellten Modelle und Use Cases auf ihre Konsistenz hin getestet.

3.5.3 Design

Im Prozeß ‘Design’ wird der Systementwurf erstellt. Er gliedert das Gesamtsystem in Teile, die getrennt bearbeitet werden können. Bei der Systemaufteilung sind Wiederverwendungsaspekte zu berücksichtigen. Dieser Prozeß liefert außerdem eine vollständige Beschreibung des Verhaltens des Systems, das mittels SDL-Diagrammen dargestellt wird. Zum Kontrollfluß wird in der Vorlage keine Aussage gemacht. Grafisch dargestellt ist das MVP-L-Prozeßmodell in Anhang A.1.4.

(1) Create_architecture
In diesem Prozeß wird die Architektur des System festgelegt. Diese wird in SDL mittels Block-Diagrammen beschrieben. Die so aufgebaute Block-Hierarchie stellt eine formale Definition des Systems da.

(2) Create_design_module_structure
Es wird ein Diagramm erstellt, welches die Struktur des Systems darstellt. Dabei spiegelt die Modulstruktur die Programmkodemodule wieder, aus der die Anwendung bestehen wird. Der Wiederverwendungsaspekt spielt eine wichtige Rolle bei der Erstellung der Struktur.

(3) Create_static_interfaces
Die Schnittstellen zwischen den SDL-Blöcken werden in diesem Prozeß erstellt. Dabei werden auch die benötigten Datentypen festgelegt.

(4) Create_design_use_cases
Die Use Cases aus dem ‘System_analysis’-Prozeß werden so verfeinert, daß sie zu den Definitionen der statischen Schnittstellen passen. So werden durch die Use Cases die dynamischen Aspekte der Schnittstellen dargestellt.

(5) Object_design
In diesem Prozeß wird das Verhalten des Systems formal beschrieben.

(6) Design_testing
In diesem Prozeß wird der SDL-Entwurf gegen seine Anforderungen getestet.

Dabei kann nicht nur das Gesamtsystem, sondern auch einzelne Teile getestet werden.

(7) Design_consistency_checks
Abschließend werden die in dem ‘Design’-Prozeß erstellten Produkte auf ihre Konsistenz hin überprüft.

3.5.4 Implementation_[2]

Bei der Implementierung wird aus den zuvor entwickelten Modellen die gewünschte Anwendung in einer bestimmten Programmiersprache oder in Hardware umgesetzt. Dabei müssen die Besonderheiten einer Implementierung in Hardware oder in den verschiedenen Programmiersprachen berücksichtigt werden. Die vier Prozesse werden dabei sequentiell durchlaufen, außer in der Testphase werden Fehler festgestellt, so daß ein vorhergehender Prozeß wiederholt durchlaufen werden muß. Im Anhang A.1.5 ist eine grafische Darstellung vom MVP-L-Modell des Prozesses zu finden.

(1) Partition
In diesem Prozeß wird das SDL-System in eigenständige Teilsysteme zerlegt, die für sich alleine lauffähig sind.

(2) Adaption
Bei der Adaption wird das SDL-System in seine Arbeitsumgebung eingepaßt.

(3) Generation_of_code
Der Programmkode für das SDL-System wird weitgehend automatisch erzeugt, der anschließend noch angepaßt werden kann.

(4) Testing
Es wird getestet, ob das entwickelte System seinen Anforderungen gerecht wird.

3.6 Erfahrung mit der Vorlage

Im folgenden wird die gemachte Erfahrung entsprechend der für die Modellierung mit MVP-L wichtigen Aspekte erläutert. Anschließend folgen Anmerkungen zu speziellen Punkten.

- *Beschreibung der Prozesse*
Die einzelnen Aktivitäten sind auf dem obersten Abstraktionsniveau klar von einander getrennt und ausführlich beschrieben. Auf der Ebene von Prozeßverfeinerungen sind die Aktivitäten nur skizzenhaft dargestellt.
- *Verfeinerungen der Prozesse*
Für jeden Prozeß auf dem obersten Abstraktionsstufe ist genau eine Verfeinerungsstufe in der Vorlage skizziert.
- *Kontrollfluß*
Die Vorlage gibt den Kontrollfluß für die Prozesse auf dem obersten Abstraktionsniveau fest vor. Für Prozesse auf der ersten Verfeinerungsstufe wird lediglich ein

2.Der Tiefstrich ist nötig, da der Begriff ‘Implementation’ in MVP-L reserviert ist.

Vorschlag gemacht. Inden Prozessen ‘System Design’ und ‘Object Design’, die in dem modellierten MVP-L-Modell zum Prozeß ‘Design’ zusammengefaßt wurden, wird kein Vorschlag unterbreitet.

- *Entry-/Exit-Kriterien*
  Hierüber wird keine Aussage gemacht.
- *Produzierte, konsumierte und modifizierte Produkte von Prozessen*
  Es wird klar ausgeführt, welcher Prozeß welche Produkte produziert. Allerdings wird höchstens impliziert dargelegt, welche Produkte von welchen Prozessen konsumiert oder modifiziert werden.
- *Verfeinerungen der Produkte*
  Es wird in der Vorlage keine Vorgabe für die Verfeinerung von Produkten gemacht. (Die im MVP-L-Modell vorhandenen Produktaggregationen dienen überwiegend der Übersichtlichkeit.)
- *Benötigte Ressourcen*
  Welche Ressourcen bei den einzelnen Aktivitäten verwendet werden, wird nicht beschrieben.
- *Fehler in der Vorlage*
  Der Prozeß ‘Design’ soll das Artefakt ‘SDL-Design’ produzieren, im darauf folgenden Prozeß ‘Implementation’ wird laut Vorlage aber stattdessen ein Produkt ‘Object design model’ erwartet, welches aber von keinem Prozeß produziert wird.
  Fehlerauflösung: Die Produkte ‘SDL-Design’ und ‘Object design model’ werden als identisch angesehen und im MVP-L-Modell als ‘SDL_design’ modelliert.

3.7 Erfahrung mit MVP-L

Die Konzepte von MVP-L waren geeignet, um die in der Vorlage gemachten Angaben bezüglich der Prozesse von SOMT vollständig zu modellieren.

Etwas unübersichtlich ist in MVP-L, daß alle Produkte, die produziert werden und kein Bestandteil einer Aggregation sind, auf dem obersten Abstraktionsniveau produziert werden müssen; auch wenn das Produkt lediglich von Prozessen auf niedrigeren Abstraktionsniveaus produziert und konsumiert wird.

In Abbildung 9 - ein Ausschnitt aus der grafischen Repräsentation der ersten Version des MVP-L-Modells ‘Design’ von SOMT - müssen die Produkte ‘Design_module_structure’, ‘Architecture_definitions’ und ‘Textual_design_documentation’ so modelliert werden, daß sie auf dem obersten Abstraktionsniveau produziert werden, obwohl sie ausschließlich auf der ersten Verfeinerungsstufe produziert und konsumiert werden müßten.

Als Abhilfe wurden solche Produkte bei den endgültigen MVP-L-Modellen mit Produkten aggregiert, die auf dem obersten Abstraktionsniveau produziert werden. So sind auf dem obersten Abstraktionsniveau nur noch die aggregierten Produkte sichtbar (d.h. werden produziert).

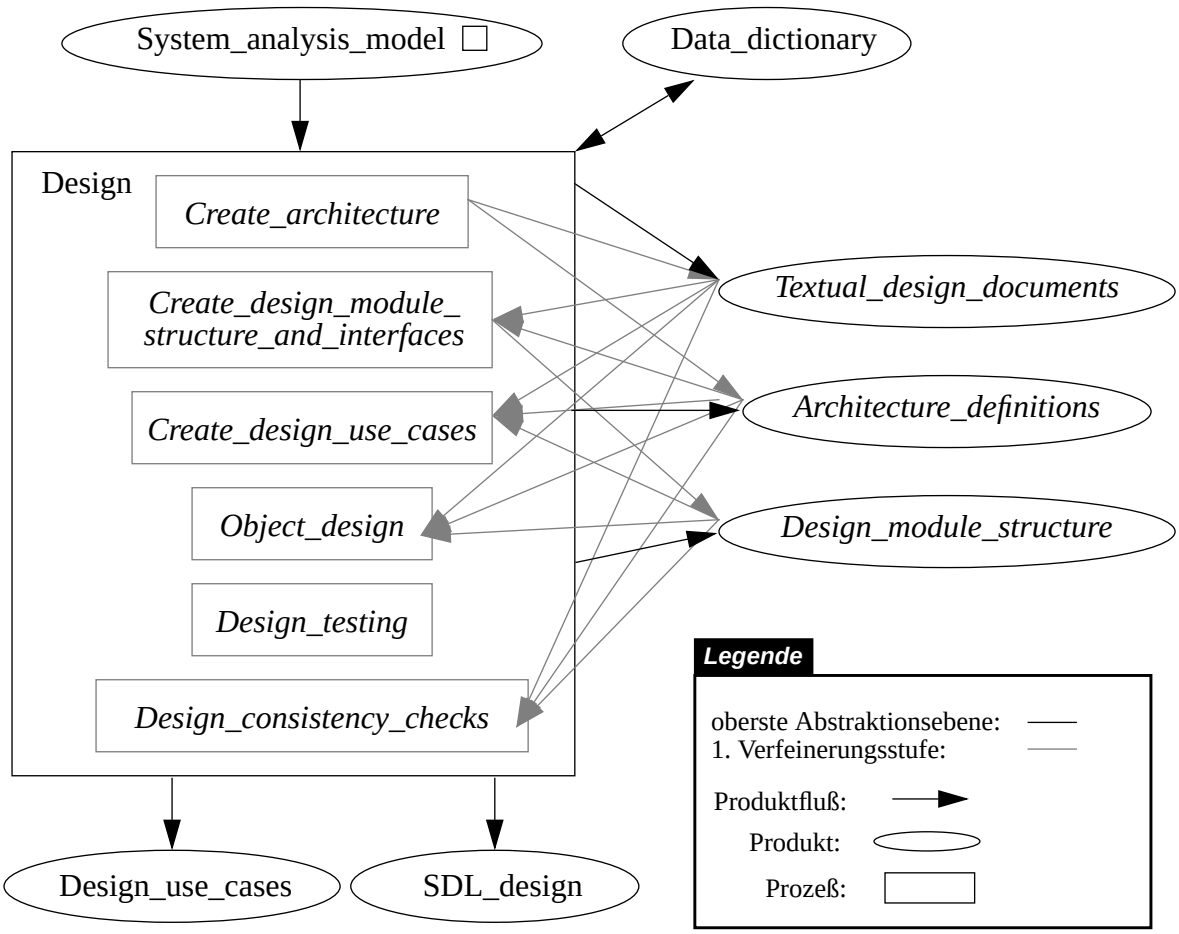

Abbildung 9: Ausschnitt der ersten Modellierung vom Prozeß ‘Design’

3.8 Bisherige Erfahrungen im SFB 501

Eine auf SOMT basierende Methode wurde bereits von der AG Rechnernetze am Fachbereich Informatik der Universität Kaiserslautern eingesetzt. Es wurden bereits Telekommunikationssysteme und ein verteiltes Modelleisenbahnsystem in Kooperation mit Teilprojekt B5 (A6 Systemsoftware) erstellt. Die daraus resultierenden Erfahrungen sind in das MVP-L-Prozeßmodell eingeflossen. Beschrieben sind die Änderungen im Abschnitt 3.10.

3.9 Hinweise zur Anpassung an verschiedene Kontexte

Es gibt in der Vorlage keinen Hinweis bezüglich der Anpassung der Methode an unterschiedliche Entwicklungskontexte. Der Beschreibung von SOMT ist ein Beispiel mitgegeben, welches für mittelgroße Projekte geeignet sein soll - genauere Angaben werden nicht gemacht. Bei dem Beispiel wird vor der ‘Requirement_analysis’ noch ein ‘Prestudy/con-

ceptualization'-Prozeß durchlaufen. Bis auf den neu hinzugekommene Prozeß 'Prestudy/ conceptualization' werden die übrigen Prozesse iterativ durchlaufen.

3.10 Modifikation des Originals

(1) Bei der auf SOMT basierenden Methode, die in der AG Rechnernetze eingesetzt wird, gibt es keine Trennung des Entwurfprozesses in System- und Objekt-Design. Auf der obersten Abstraktionsebene gibt es nur den Prozeß 'Design', der beide Prozesse in sich vereint.

(2) Mehrere Produkte wurden zu übergeordneten Produkten zusammengefaßt (siehe Tabelle 2), um die Übersichtlichkeit auf der obersten Prozeßebene zu erhöhen. In der Vorlage war keine Produktaggregation vorgesehen.

(3) Da keine Entry-/Exit-Kriterien vorgegeben waren, wurden diese Kriterien nach eigenem Ermessen modelliert. Im folgenden wird grob dargestellt, nach welcher Maßgabe die Kriterien modelliert wurden:
Prozesse dürfen nur ausgeführt werden, wenn alle zu konsumierenden Produkte vorhanden sind und die zu produzierenden Produkte noch nicht alle bereits komplett fertig gestellt sind. Der Konsistenztest am Ende einiger Phasen wird nur mit komplett fertigen Produkten aufgerufen. In den nächsten Prozeß wird erst dann übergegangen, wenn alle Produkte des aktuellen Prozesses konsistent sind.

(4) Da für den Prozeß 'Design' keine Vorgabe bezüglich des Kontrollflusses gemacht wurde, ist dies folgendermaßen realisiert worden: Die Prozesse 'Create_architecture', 'Create_design_module_structure', 'Create_static_interfaces' und 'Create_ design_use_cases' werden solange iterativ durchlaufen, bis die produzierten Produkte der Prozesse vorläufig fertig sind (product.status = 'complete'). Es wird der Prozeß 'Object_design' ausgeführt und anschließend 'Design_testing'. Sollten Schwächen im Entwurf festgestellt worden sein, wird wieder beim Prozeß 'Create_architecture' begonnen, andernfalls werden die Konsistenztests durchgeführt.

(5) Die Inkonsistenz bezüglich den von dem Prozeß 'Design' produzierten Produkt 'SDL_design' und von dem Prozeß 'Implementation_' geforderten Produkt 'Object_design_model' wurde aufgelöst, indem angenommen wurde, daß das gleiche Produkt gemeint, aber unterschiedlich bezeichnet wurde. Im MVP-L-Modell konsumiert daher der Prozeß 'Implementation_' das Produkt 'SDL_design'.

Anhang

A.1 Grafische Darstellung der MVP-L-Modelle

In diesem Abschnitt sind die MVP-L-Modelle grafisch dargestellt. Es wird zuerst das Modell auf dem obersten Abstraktionsniveau gezeigt. Es folgen die einzelnen Prozesse mit ihren Verfeinerungen. Zur Darstellung der Modelle wurde die grafische Notation aus [BDHK95] verwendet.

Jede Darstellung eines Prozesses besteht aus vier Bausteinen: Oben befindet sich das grafische MVP-L-Prozeßmodell, das den Datenfluß aufzeigt. In der Mitte befindet sich rechts die dazugehörende Legende für die grafische Repräsentation des MVP-L-Modells. Links davon ist das Gesamtmodell der Methode in verkleinerter Form als Übersicht (overview) abgebildet. Die Namen der Produkte und Prozesse wurden dabei durch entsprechende Abkürzungen ersetzt: zum Beispiel wurde ‘External_textual_requirements’ durch ‘etr’ abgekürzt. In der Übersicht sind nur die Produkte und Prozesse weiß hinterlegt, die im obigen Prozeßmodell auch sichtbar sind; alle anderen sind wie der Hintergrund unterlegt. Ganz unten ist der Kontrollfluß (control flow) des MVP-L-Modells aufgezeigt. Die Namen der Prozesse sind dabei durch Zahlen ersetzt, die sich bei den Prozessen im oben dargestellten MVP-L-Prozeßmodell wiederfinden. Der Kontrollfluß ist nur schematisch dargestellt und kann im Detail abweichen. Die exakte Kontrollflußbeschreibung ist im textuellen MVP-L-Modell enthalten (d.h. über Entry-/Exit-Kriterien formuliert); einige Beispiele finden sich in Anhang A.2.

A.1.1 Gesamtmodell ohne Verfeinerungen

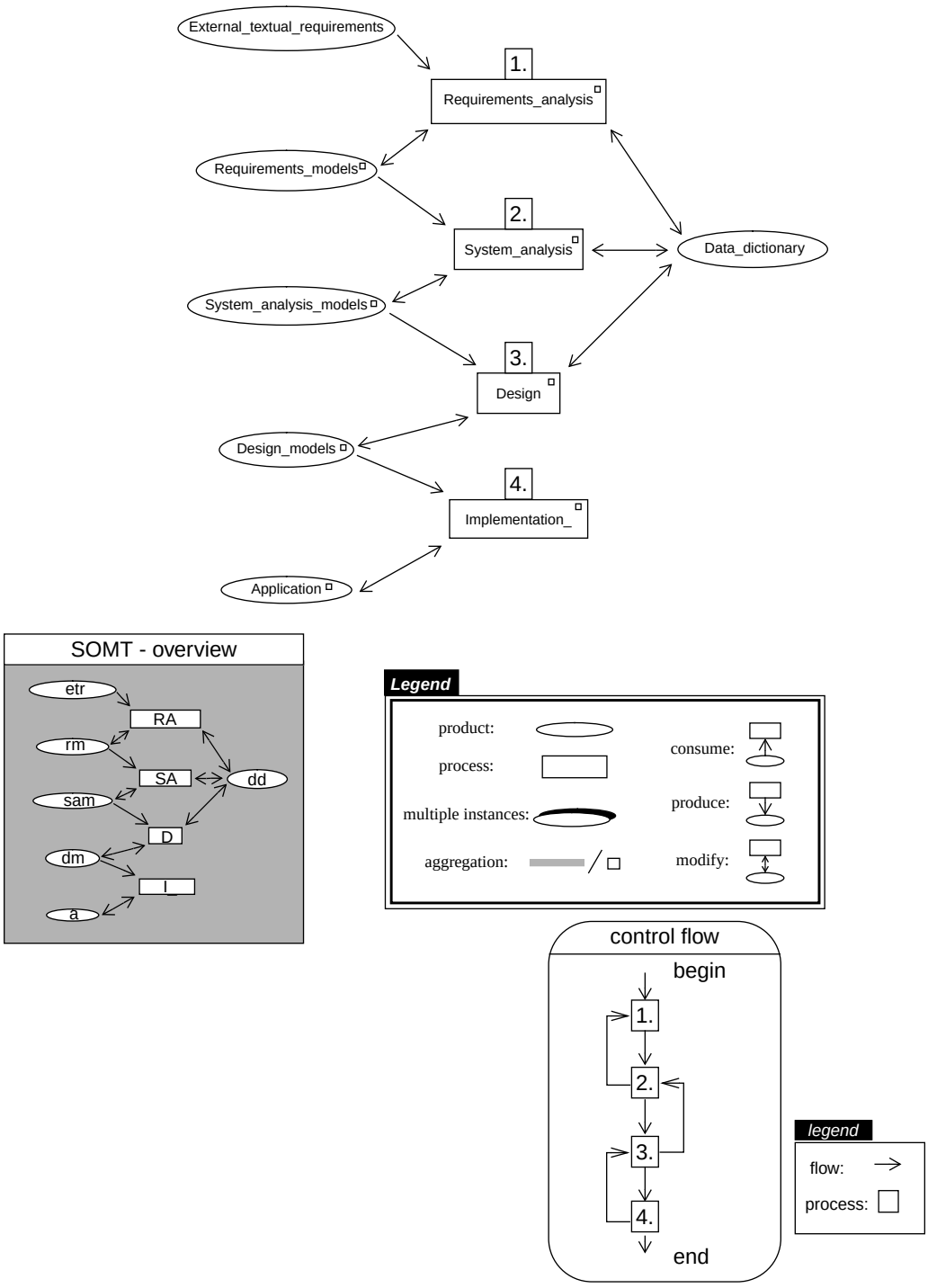

Abbildung 10: SOMT-Gesamtprozeß

A.1.2 Requirements_analysis

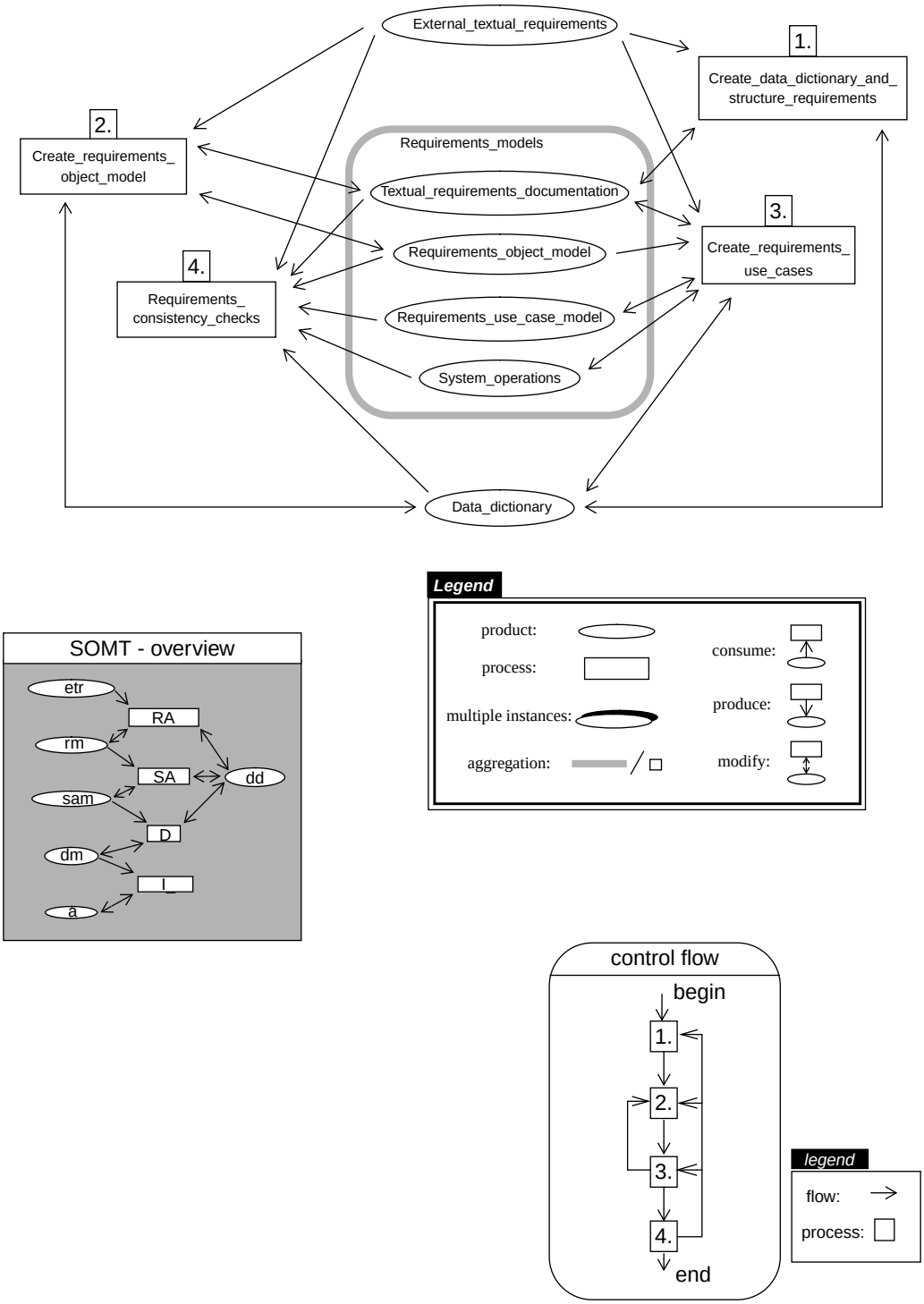

Abbildung 11: SOMT-Requirements_analysis-Prozeß

A.1.3 System_analysis

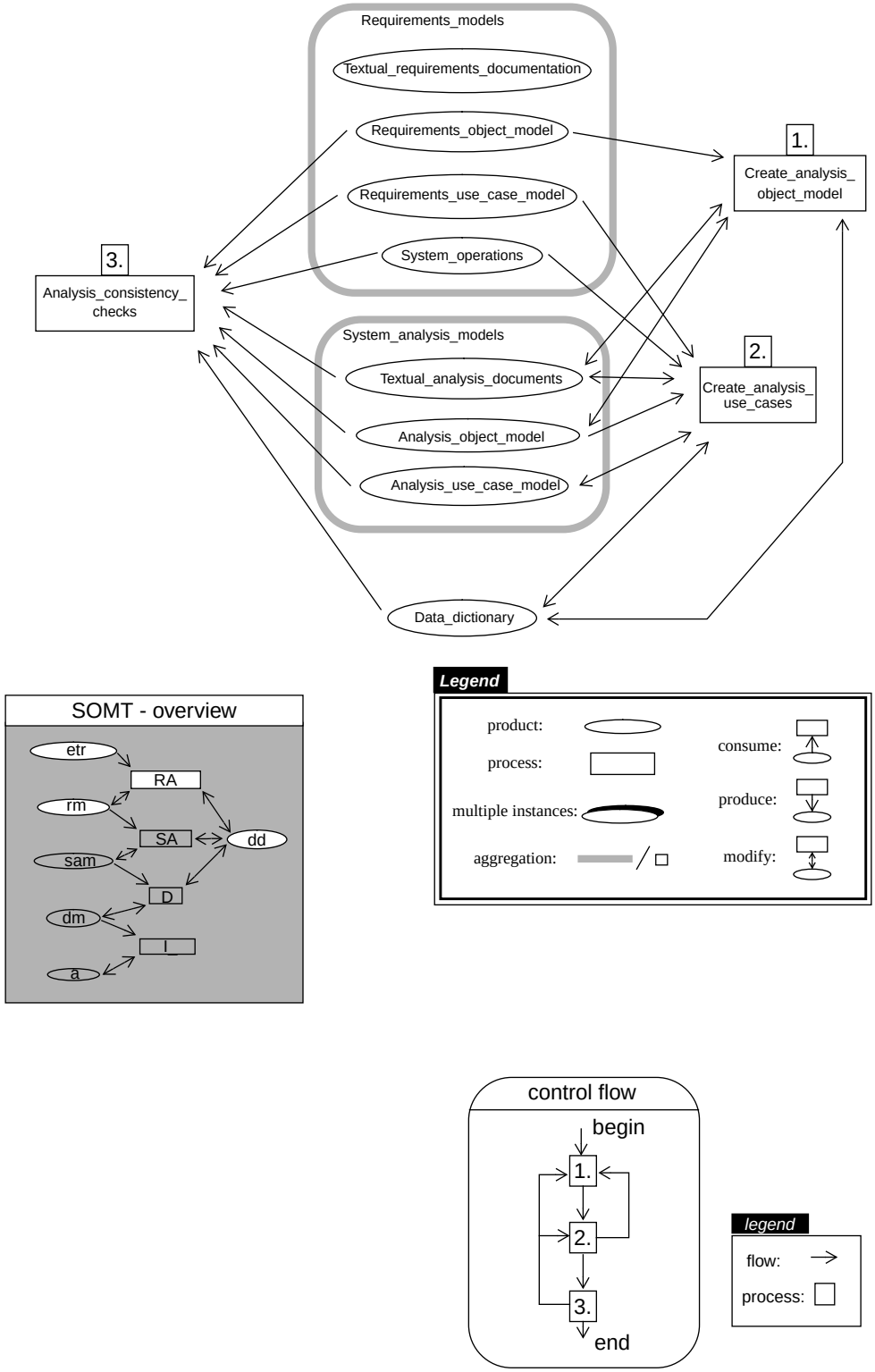

Abbildung 12: SOMT-System_analysis-Prozeß

A.1.4 Design

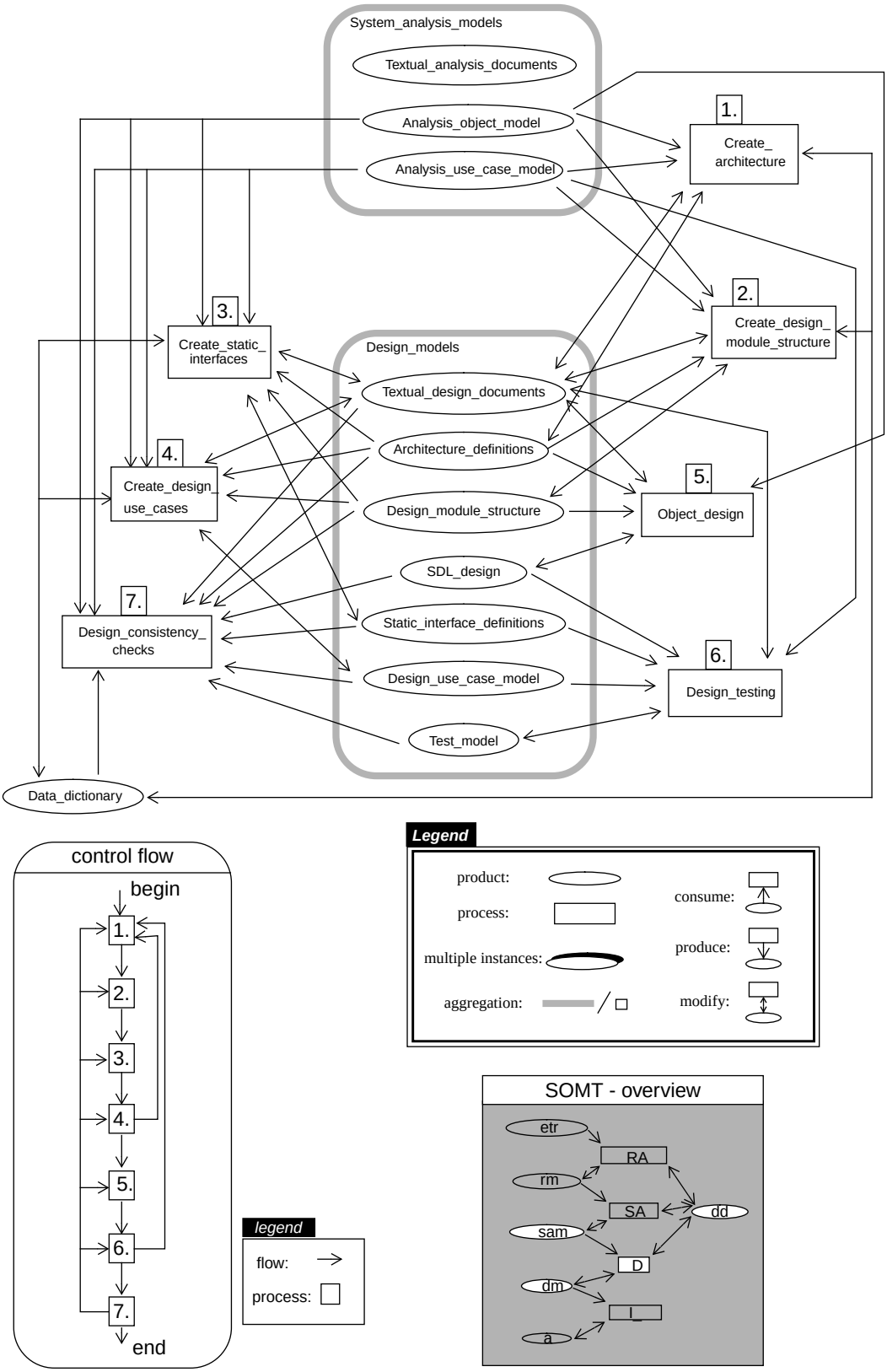

Abbildung 13: SOMT-Design-Prozeß

A.1.5 Implementation_

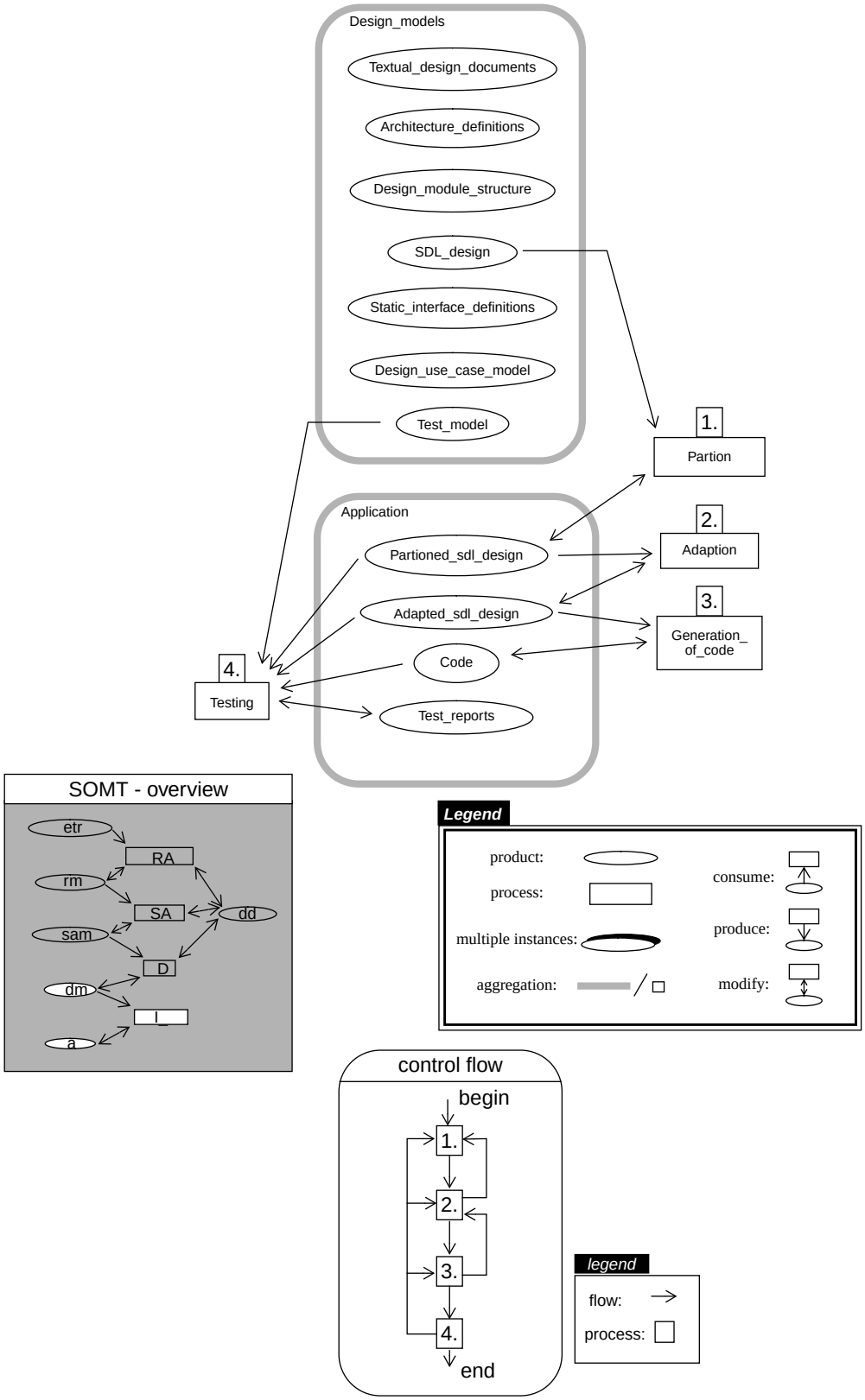

Abbildung 14: SOMT-Implementation_-Prozeß

A.2 Textuelle Beschreibung ausgewählter MVP-L-Modelle

Im folgenden wird für jeden bei der Modellierung der drei Software-Entwicklungsmethoden verwendeten MVP-L-Modelltyp ein Beispiel angegeben: Projektplan, Prozeßmodell, Produktmodell und Produktattributmodell. Für die Produkt- und die Prozeßmodelle wird jeweils eine Aggregation und eine Verfeinerung angegeben. Die reservierten Wörter von MVP-L wurden zur Erhöhung der Übersichtlichkeit fett hervorgehoben.

A.2.1 Projektplan (Beispiel)

Im folgenden ist der Projektplan von SOMT dargestellt. Er bildet die übergeordnete Einheit für alle MVP-L-Modelle, die zur Beschreibung der SOMT-Methode nötig sind.

In der <imports>-Klausel sind alle Modelle aufgeführt, die zur Deklaration der Prozeß-, Produkt- und Ressourcenobjekte benötigt werden. Deklariert werden die Objekte in der <objects>-Klausel. Dabei werden hier Objekte bestimmt, die keiner Aggregation angehören (demnach keine Verfeinerungen sind). In der <object_relations> werden die Verbindungen der Modelle angegeben. Dabei werden den Prozeßmodellen die Produktmodelle übergeben, die sie produzieren, konsumieren oder modifizieren (konsumieren und produzieren).

In Abbildung 15 ist ein Ausschnitt des SOMT-Projektplan dargestellt, wie er in der Erfahrungsdatenbank der AG Software Engineering zu finden ist.

```
project_plan SOMT is

 imports
  process_model Requirements_analysis, System_analysis,Design, Implementation_;
  product_model External_textual_requirements, Data_dictionary,
    Requirements_models, System_analysis_models, Design_models, Application;
 objects
  requirements_analysis: Requirements_analysis;
  system_analysis: System_analysis;
  design: Design;
  implementation_: Implementation_;
  external_textual_requirements: External_textual_requirements;
  data_dictionary: Data_dictionary;
  requirements_models: Requirements_models;
  system_analysis_models: System_analysis_models;
  design_models: Design_models;
  application: Application;
 object_relations
  requirements_analysis(external_textual_requirements => external_textual_requirements,
     requirements_models => requirements_models,
     data_dictionary => data_dictionary);
  system_analysis(requirements_models => requirements_models,
     system_analysis_models => system_analysis_models,
     data_dictionary => data_dictionary);
  design(system_analysis_models => system_analysis_models,
     design_models => design_models,
     data_dictionary => data_dictionary);
  implementation_(design_models => design_models,
     application => application);

end project_plan SOMT.
```

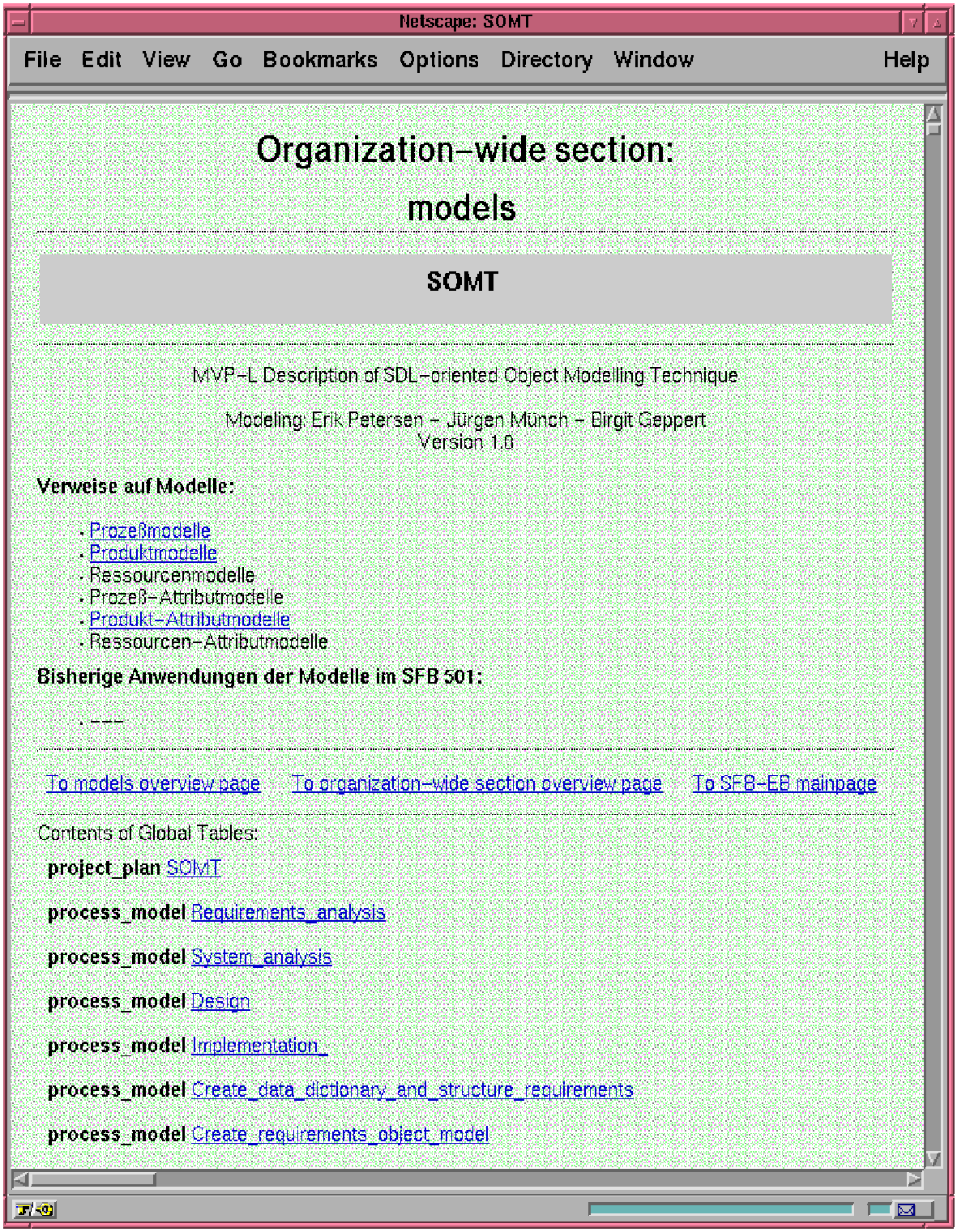

Abbildung 15: Ausschnitt der Modellierung von SOMT in der Erfahrungsdatenbank

A.2.2 Prozeßmodellaggregation (Beispiel)

Eine Prozeßbeschreibung gliedert sich grob in die drei Teile Schnittstelle <process_interface>, Körper <process_body> und benötigte Resourcen <process_resources>. Die Prozeßverfeinerungen, die die Prozeßaggregation bilden, sind im <process_body> in der <refinement>-Klausel angegeben.

In der Schnittstellenbeschreibung werden die für die Deklaration benötigten Modelle aufgeführt. Der Produktfluß wird bestimmt, in dem das Produkt entweder als zu konsumieren, zu produzieren oder zu konsumieren und zu produzieren (zu modifizieren) angegeben wird. Gleichzeitig werden die Objekte in der <product_flow>-Klausel deklariert. Im Beispiel soll das Produkt ‘external_textual_requirements’ vom Typ ‘External_textual_requirements’ vom Prozeß ‘Requirements_analysis’ konsumiert werden.

Mittels Entry- und Exit-Kriterien wird der Kontrollfluß vorgegeben: Es wird bestimmt, welche Bedingungen erfüllt sein müssen, damit der Prozeß gestartet werden, und unter welchen Umständen ein Prozeß beendet werden kann. Zusätzlich können Bedingungen mittels einer Invariante angegeben werden, die während der gesamten Prozeßausführung erfüllt sein muß. Der angegebene Beispielprozeß soll erst gestartet werden, wenn das Produkt ‘external_textual_requirements’ vollständig fertig (‘complete’) ist. Außerdem dürfen die Produkte ‘requirements_models’ und ‘data_dictionary’ noch nicht abgeschlossen sein: Sie müssen also entweder den Status ‘non_existent’ oder ‘incomplete’ haben.

Im <process_body> ist die Verfeinerung definiert. Die <imports>-Klausel beinhaltet alle Modelle, die zur Deklaration der Objekte in der <objects>-Klausel benötigt werden. In der <object_relations> wird angegeben, wie die Aggregation aussieht. In diesem Beispiel besteht der aggregierte Prozeß aus der Summe aller Prozeßverfeinerungen, da die Verfeinerungen mittels ‘&’ verbunden sind. Wird die Prozeßaggregation ausgeführt, bedeutet dies, daß alle Verfeinerungen des Prozesses ausgeführt werden müssen.

In den <interface_relation> werden die Prozeßmodellverfeinerungen mit den Produktmodellen verbunden.

In diesem Beispiel sind keine Resourcen - weder für das Personal noch für Werkzeuge - angegeben.

```
process_model Requirements_analysis() is

 process_interface
  imports
   product_model External_textual_requirements, Requirements_models, Data_dictionary;
  exports
  product_flow
   consume
    external_textual_requirements: External_textual_requirements;
   produce
   consume_produce
    requirements_models: Requirements_models;
    data_dictionary: Data_dictionary;
  context
  entry_exit_criteria
   local_entry_criteria
    (external_textual_requirements.status = 'complete' and ((requirements_models.status = 'non_existent'
       or requirements_models.status = 'incomplete') or (data_dictionary.status ='non_existent' or
       data_dictionary.status = 'incomplete')));
   global_entry_criteria
```

```
    local_invariant
    global_invariant
    local_exit_criteria
      (requirements_models.status = 'consistence' and data_dictionary.status = 'consistence');
  global_exit_criteria
end process_interface

process_body
  refinement
    imports
      process_model Create_data_dictionary_and_structure_requirements, Create_requirements_object_model,
        Create_requirements_use_cases, Requirements_concistency_checks;
    objects
      create_data_dictionary_and_structure_requirements: Create_data_dictionary_and_structure_requirements;
      create_requirements_object_model: Create_requirements_object_model;
      create_requirements_use_cases: Create_requirements_use_cases;
      requirements_concistency_checks: Requirements_concistency_checks;
    object_relations
      (create_data_dictionary_and_structure_requirements & create_requirements_object_model &
          create_requirements_use_cases & requirements_concistency_checks);
    interface_refinement
    interface_relations
      create_data_dictionary_and_structure_requirements(external_textual_requirements =>
          external_textual_requirements,data_dictionary => data_dictionary,
          textual_requirements_model => requirements_models.textual_requirements_model);
      create_requirements_object_model(external_textual_requirements => external_textual_requirements,
          data_dictionary => data_dictionary,
          textual_requirements_model => requirements_models.textual_requirements_model,
          requirements_object_model => requirements_models.requirements_object_model);
      create_requirements_use_cases(external_textual_requirements => external_textual_requirements,
          data_dictionary => data_dictionary,
          textual_requirements_model => requirements_models.textual_requirements_model,
          requirements_object_model => requirements_models.requirements_object_model,
          requirements_use_case_model => requirements_models.requirements_use_case_model,
          system_operations => requirements_models.system_operations);
    requirements_concistency_checks(external_textual_requirements => external_textual_requirements,
          data_dictionary => data_dictionary,
          textual_requirements_model => requirements_models.textual_requirements_model,
          requirements_object_model => requirements_models.requirements_object_model,
          requirements_use_case_model => requirements_models.requirements_use_case_model,
          system_operations => requirements_models.system_operations);
    attribute_mappings
end process_body

process_resources
  personnel_assignment
  tool_assignment
end process_resources

end process_model Requirements_analysis
```

A.2.3 Prozeßmodellverfeinerung (Beispiel)

Der hier dargestellt SOMT-Prozeß ist ein Teil der Verfeinerung der unter Anhang A.2.2 beschriebenen Prozeßaggregation. Der Aufbau des MVP-L-Modells weist gegenüber der Aggregation vor allem den Unterschied auf, daß <process_body> keine Verfeinerung enthält.

```
process_model Create_requirements_object_model() is

 process_interface
  imports
   product_model External_textual_requirements, Requirements_object_model,
    Data_dictionary, Textual_requirements_model;
  exports
  product_flow
   consume
    external_textual_requirements: External_textual_requirements;
   produce
   consume_produce
    requirements_object_model: Requirements_object_model;
    data_dictionary: Data_dictionary;
    textual_requirements_model: Textual_requirements_model;
  context
  entry_exit_criteria
   local_entry_criteria
    (external_textual_requirements.status = 'complete' and (requirements_object_model.status = 'non_existent'
       or requirements_object_model.status = 'incomplete') and textual_requirements_model.status != 'non_existent'
       and data_dictionary.status != 'non_existent');
   global_entry_criteria
   local_invariant
   global_invariant
   local_exit_criteria
    (requirements_object_model.status = 'incomplete' or requirements_object_model.status = 'complete');
   global_exit_criteria
 end process_interface

 process_body
  implementation
 end process_body

 process_resources
  personnel_assignment
  tool_assignment
 end process_resources

end process_model Create_requirements_object_model
```

A.2.4 Produktmodellaggregation (Beispiel)

Ein Produktmodell gliedert sich in MVP-L in die Teile <product_interface> und <product_body>. Im <product_interface> werden die Im- und Exporte deklariert. Im dargestellten Beispiel, wird dem Prozeß der Parameter ‘product_status’ übergeben. Das Objekt ‘status’ wird mit dem Wert des Parameters initialisiert und als Export definiert. Auf dieses Objekt kann von außen zugegriffen werden.

Der <product_body> ist ähnlich dem <process_body> aufgebaut. Mit den <attribute_mappings> wird in diesem Beispiel der Wert des Exportobjekt bestimmt, das von den Exportobjekten der Produktverfeinerungen abhängt.

```
product_model Requirements_models(product_status: Product_status) is

product_interface
  imports
    product_attribute_model Product_status;
  exports
    status: Product_status := product_status;
 end product_interface

 product_body
  refinement
   imports
     product_model Textual_requirements_model, Requirements_object_model,
       Requirements_use_case_model, System_operations;
   objects
     textual_requirements_model: Textual_requirements_model;
     requirements_object_model: Requirements_object_model;
     requirements_use_case_model: Requirements_use_case_model;
     system_operations: System_operations;
   object_relations
     textual_requirements_model & requirements_object_model & requirements_use_case_model &
        system_operations;
   attribute_mappings
     status:
       'non_existent' <-> requirements_object_model.status = 'non_existent' and
         requirements_use_case_model.status = 'non_existent' and system_operations.status = 'non_existent'
         and textual_requirements_model.status = 'non_existent';
       'complete' <-> requirements_object_model.status = 'complete' and
         requirements_use_case_model.status = 'complete' and system_operations.status = 'complete' and
         textual_requirements_model.status = 'complete';
       'consistence' <-> requirements_object_model.status = 'consistence' and
         requirements_use_case_model.status = 'consistence' and system_operations.status = 'consistence'
         and textual_requirements_model.status = 'consistence';
       'incomplete' <-> others;
 end product_body

end product_model Requirements_models
```

A.2.5 Produktmodellverfeinerung (Beispiel)

Das hier dargestellte Produktmodell ist ein Teil der Verfeinerung von ‘Requirements_ models’. Die Produktmodellverfeinerung unterscheidet sich in dem <product_body> von der Aggregation, da hier keine Verfeinerung angegeben ist.

```
product_model Requirements_object_model(product_status: Product_status) is

 product_interface
  imports
    product_attribute_model Product_status;
  exports
    status: Product_status := product_status;
 end product_interface
```

```
product_body
  implementation
end product_body

end product_model Requirements_object_model
```

A.2.6 Produktattribut-Modell (Beispiel)

Das Modell ‘Product_status’ beinhaltet als <attribute_type> die möglichen Zustände, in denen sich ein Produkt während der Projekterstellung befinden kann.

```
product_attribute_model Product_status() is
  attribute_type
('non_existent', 'incomplete', 'complete', 'tested','consistence');
  attribute_manipulation

end product_attribute_model Product_status
```

A.3 Die Prozeßunterstützungsumgebung MVP-E

Auf der Sprache MVP-L beruht die Prozeßunterstützungsumgebung MVP-E (Multi-View Process Environment) [BHMV97]. Sie umfaßt folgende Werkzeuge, die in der Abbildung 16 zusammenfaßt dargestellt sind:

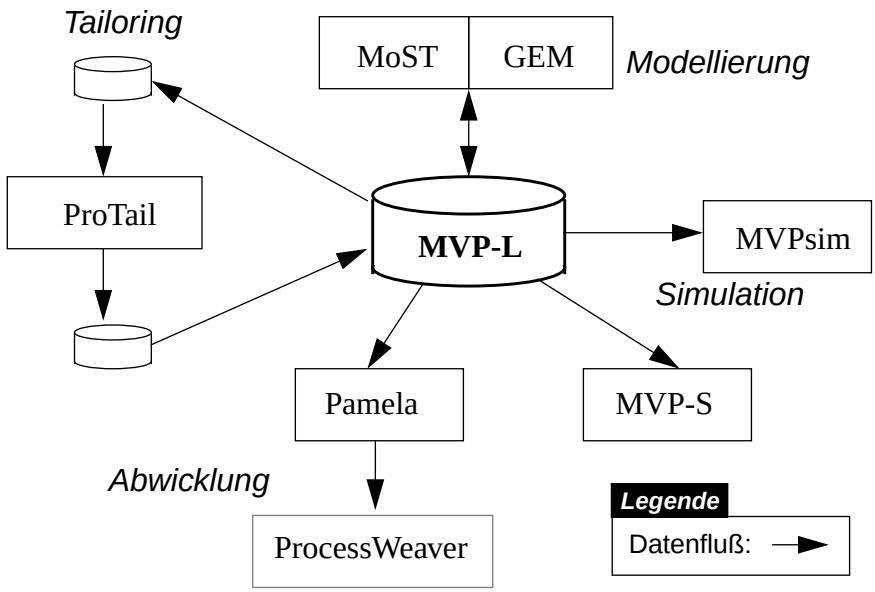

Abbildung 16: Die Bestandteile von MVP-E [BHMV97]

- *GEM (Graphical Editor for MVP-L):*
  Werkzeug zur grafischen Erstellung, Änderung und Analyse von MVP-Modellen. In Abbildung 17 ist zur Anschauung der Projektplan von SOMT im Werkzeug GEM dargestellt.

- *MoST (Modeling Support Tool):*
  Eine grafische Benutzeroberfläche zum Erstellen, Verändern, Kontrollieren und Analysieren von textuellen MVP-Modellen. Abbildung 18 enthält einen Ausschnitt des SOMT-Projektplans wie er in MoST dargestellt wird.

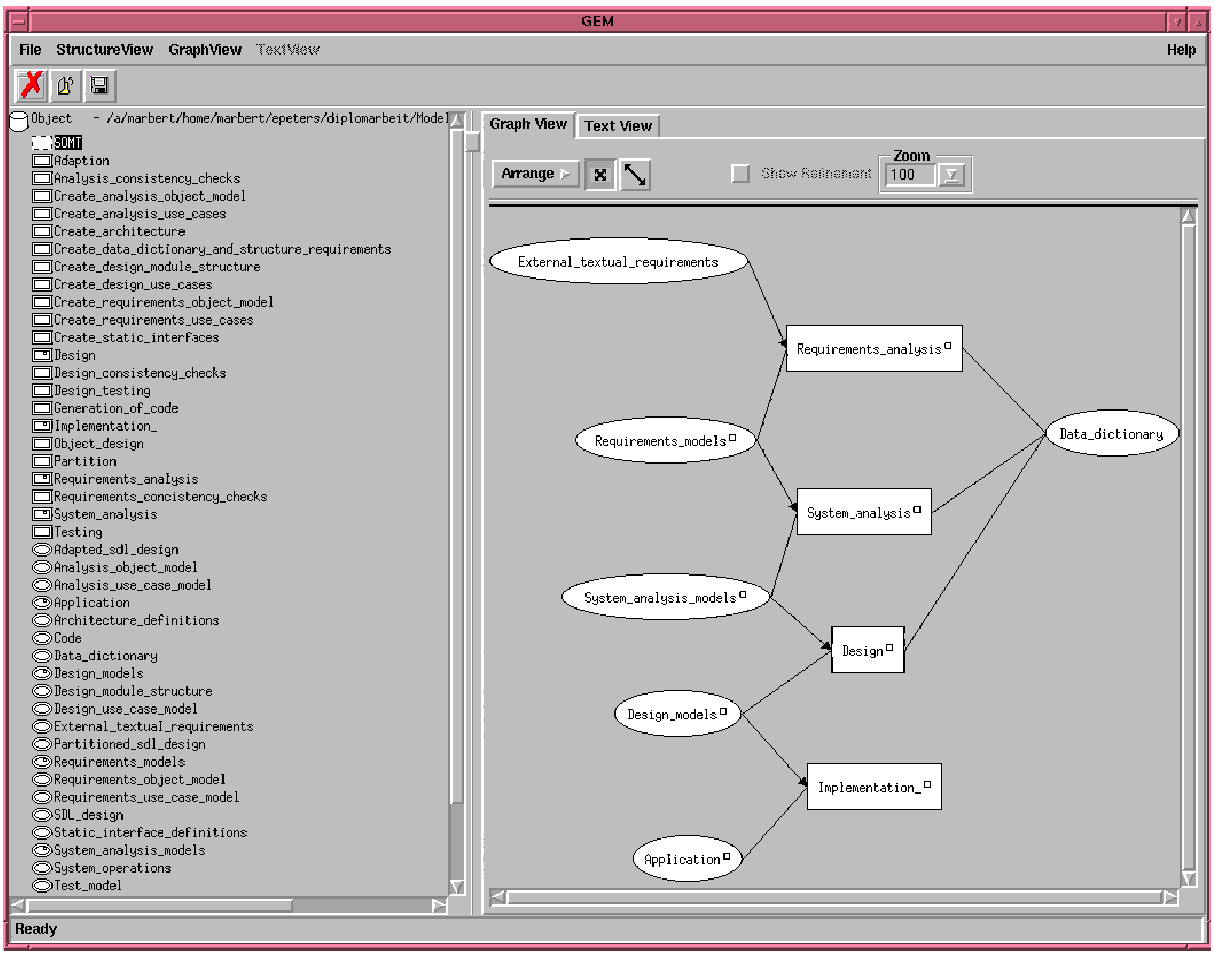

Abbildung 17: Grafisch dargestellter SOMT-Projektplan im Werkzeug GEM

- *MVPsim:*
  Ein Simulator zur Risikoanalyse, der stochastische Modelle zum Aufbau von Qualitätsmodellen benutzt.
- *MVP-S:*
  MVP-S ist eine Ausführungsmaschine, die die operationale Semantik von MVP-L definiert.
- *Pamela:*
  Ein Übersetzer, der MVP-L Prozeßmodelle für das kommerzielle Werkzeug ProcessWAEVER ausführbar macht.

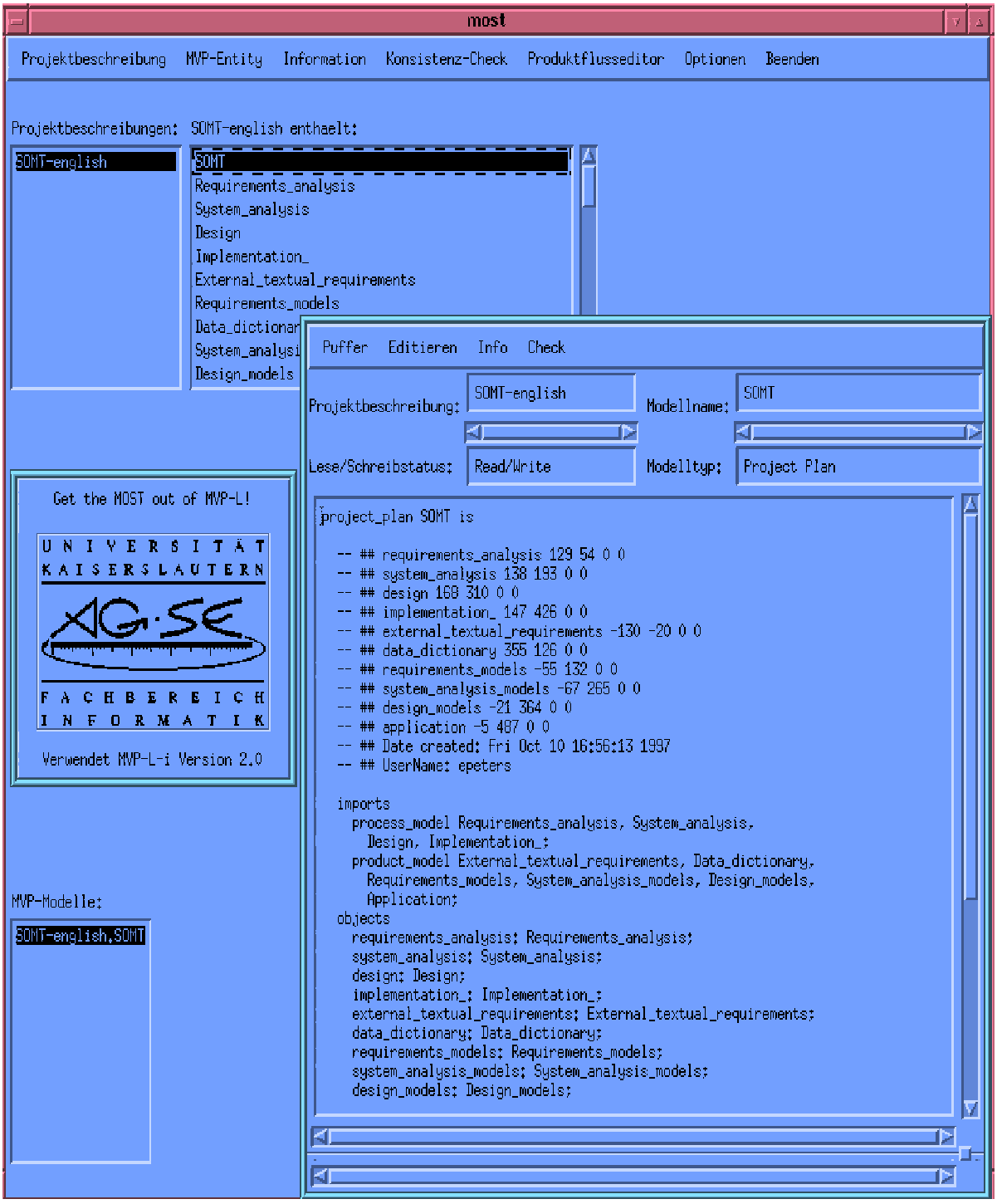

Abbildung 18: Textuell dargestellter SOMT-Projektplanausschnitt in MoST

- *ProTail (Process Tailoring Tool):*
  Werkzeugprototyp zum Anpassen von Prozeßmodellen an Charakteristika und Ziele von Projekten. Um ein Prozeßmodell an ein bestimmtes Projekt anzupassen, werden Regeln angewendet. [SchM97]

Literatur

Zum leichteren Auffinden ist für Quellen, die in der Universitätsbibliothek Kaiserslautern vorhanden sind, am Ende der jeweiligen Literaturangabe die Signatur in runden Klammern angegeben.

[ABGM92] Armenise, P.; Bandinelli, S.; Ghezzi, C.; Morzenti, A.: *Software processes Representation Languages: Survey and Assessment*, **Proceedings of the Conference on Software Engineering and Knowledge Engineering SEKE-92**, Capri (Italien), Juni 1992.

[AlDa91] Alexander, Linda C.; Davis, Alan M.: *Criteria for Selecting Software Process Models*, **Proceedings of the Fifteenth Annual International Computer Software & Applications Conference**, IEEE Computer Society Press, Tokyo, 11.-13. September 1991, Seite 521-528. (INF 151/115-15)

[Alex90] Alexander, Linda C.: ***Selection Criteria for Alternate Software Life Cycle Process Models***, Master of Science Thesis, George Mason University, Fairfax, Virginia (USA), Summer 1990.

[Allw81] Allworth, S. T.: ***Introduction to Real-Time Software Design***, Macmillan Press Ltd., London, 1981. (INF 483/016)

[ArKe94] Armitage, James W.; Kellner, Marc I.: *A Conceptual Schema for Process Definitions and Models*, **Proceedings of the 3th International Conference on the Software Process**, Reston, Virginia (USA), IEEE Computer Society Press, 10.-11. Oktober 1994, Seite 153-165. (INF 410/224-1994)

[BaCR94] Basili, Victor R; Caldiera, Gianluigi; Rombach, H. Dieter: *The Experience Factory,* **Encyclopedia of Software Engineering,** Volume 1, John J. Marciniak (Editor), John Wiley & Sons Inc., 1994. (INF 410/013)

[BalH95] Balzert, Heide: ***Methoden der objektorientierten Systemanalyse***, Angewandte Informatik, Band 14, BI Wissenschaftsverlag, München, 1995.

[Balz97] Balzert, Helmut: *Prozeßmodelle im Quervergleich (Teil 1),* **Objekt Focus**, Nr. 9/10, 1997, Seite 50-58.

[BaMc95] Basili, Victor R.; McGarry, Frank: *The Experience Factory: How to Built and Run One*, Tutorial M1, **17th International Conference on Software Engineering,** Seattle, Washington (USA), April 1995.

[BaRo91] Basili, Victor R.; Rombach, H. Dieter: *Support for comprehensive reuse*, **Software Engineering Journal**, Institution of Electrical Engineers in British Computer Society (IEE), September 1991, Seite 303-316. (INF Z 4861)

[BDHK95] Bröckers, Alfred; Differding, Christiane; Hoisl, Barbara; Kollnischko, Frank; Lott, Christopher M.; Münch, Jürgen; Verlage, Martin; Vorwieger, Stefan: ***A graphical representation schema for the software process modeling language MVP-L***, Interner Bericht 270/95, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, Juni 1995.

[BDKK97] Baum, Lothar; Dellen, Barbara; Kamsties, Erik; von Knethen, Antje; Vorwieger, Stefan: ***Modeling Real-Time Systems with SCR***, SFB-Report 7/97, Sonderforschungsbereich 501, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, 1997.

[BeVe97] Becker, Ulrike; Verlage, Martin: ***MVP-L's Modeling Support Tool MoST***, Frauenhofer Institut für Experimentelles Software Engineering, IESE-Report No. 01.97/E, Version 2.0, Kaiserslautern, 28. Februar 1997.

[BhHe96] Bharadwaj, Ramesh; Heitmeyer, Constance: *Applying the SCR Requirements Specification Method to Practical Systems: A Case Study*, **The 21th Software Engineering Workshop**, NASA Goddard Space Flight Center, Greenbelt MD (USA), Dezember 1996.

[BhHe97] Bharadwaj, Ramesh; Heitmeyer, Constance: *Applying the SCR Requirements Method to a Simple Autopilot*, **Proceedings of the Fourth NASA Langley Formal Methods Workshop**, September 1997.

[BHMV97] Becker, Ulricke; Hamann, Dirk; Münch, Jürgen; Verlage, Martin: *MVP-E: A Process Modeling Environment*, **IEEE TCSE Software Process Newsletter**, Volume 10, 1997.

[Biel95] Bieler, P. (Leitung): ***Veranstaltungsunterlagen Gebäudeautomation***, Nr. E-10-739-103-5, Haus der Technik e. V., 27. September 1995.

[BiNe83] Birch, M. C.; Nevill, D. G.: *The Application of the Software Reuse in Developing Saftey-Critical Systems*, **Second International Conference on Software Engineering for Real Time Systems**, IEE Conference Publication, London, 18.-20. September 1983, Seite 36-39. (ELT 850/021)

[BiPe89] Biggerstaff, Ted J.; Perlis, Alan J. (Editors): ***Software Reusability***, ACM Press Frontier Series, Addison-Wesley Publishing Company, New York, 1989. (INF 410/297-1/2)

[BiRi87] Biggerstaff, Ted; Richter Charles: *Reusability Framework, Assessment, and Directions*, **IEEE Software**, März 1987 (4), Nr. 2, IEEE Computer Society Press, Seite 41-49 oder in [BiPe89]. (INF Z 4879)

[BLRV92] Bröckers, Alfred; Lott, Christopher M.; Rombach, H. Dieter; Verlage, Martin: ***MVP-L Language Report***, Interner Bericht 229/92, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, Dezember 1992.

[BLRV95] Bröckers, Alfred; Lott, Christopher M.; Rombach, H. Dieter; Verlage, Martin: ***MVP-L Language Report Version 2***, Interner Bericht 265/95, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, Februar 1995.

[BoBe90] Boehm, Barry; Belz, Frank: *Experiences with the Spiral Model as a Process Model Generator*, **Proceedings of the Fifth International Software Process Workshop**, IEEE Computer Society Press, Washington DC, Seite 43-45, 1990.

[BoSt92] Bown, Jonathan; Stavridou, Victoria: ***Saftey-Critical Systems, Formal Methods and Standards***, Revised December 1992 to apear in the Software Engineering Journal, 1992.

[BrHa93] Bræk, Rolv; Haugen, Øystein: ***Engineering Real Time Systems: An object-oriented methodology using SDL***, Prentice Hall International, 1993. (INF 411/039)

[BrDr95] Bröhl, Adolf-Peter; Dröschel, Wolfgang (Hrsg.): ***Das V-Modell***, 2. Auflage, R. Oldenbourg Verlag, München 1995. (INF 018/026 2. Aufl.)

[BRuJ96] Booch, G.; Rumbaugh, J.; Jacobson, I.: ***The Unified Modeling Language for Object-Oriented Development***, Documentation Set Version 0.91 Addendum UML Update, Rational Software Corporation, 1996.

[BRuJ97] Booch, G.; Rumbaugh, J.; Jacobson, I.: ***Unified Modeling Language***, Version 1.0, Rational Software Corporation, 1997.

[Brun90] Bunner, Alex: *Integrale Gebäudetechnik*, **Schweizer Ingenieur und Architekt** Nr. 30-31, 30. Juni 1990, Seite 853-856. (ARB Z 2002)

[BSKa95] Ben-Shaul, Israel; Kaiser, Gail E.: ***A Paradigm For Decentralized Process Modeling***, Kluwer Academic Publishers, Boston (USA), 1995. (INF 410/353)

[BuSG96] Budlong, Faye C.; Szulewski, Paul A.; Ganska, Ralph J.: ***Process Tailoring for Software Project Plans***, Software Technology Support Center (STSC), Hill AFB, Utah (USA), Januar 1996.

[Calv93] Calvez, Jean Paul: ***Embedded Real-Time Systems: A Specification and Design Methodology***, Wiley, 1993. (INF 820/055)

[CuKO92] Curtis, Bill; Kellner, Marc I.; Over, Jim: *Process Modeling*, **Communications of the ACM**, Volume 35, No. 9, September 1992, Seite 75-90. (INF Z 1415)

[Derr95] Derr, Kurt W.: ***Applying OMT***, SIGS Books, New York, 1995. (INF 418/074)

[DINV92] Deutsches Institut für Normung e. V., **DIN V 32 734, *Digitale Automation für die Technische Gebäudeausrüstung (Digitale Gebäudeautomation)***, Vornorm, Berlin, April 1992.

[DoFe94] Dowson, M.; Fernström, C.: *Towards Requirements for Enactment Mechanisms*, **Proceedings of the thied European Workshop on Software Process Technology**, (Brian C. Warboys, ed.), **Lecture Notes in Computer Science**, 772, Februar 1994.

[EkAn96] Ek, Anders: ***Combining Object-Oriented Analysis and SDL Design***, Adresse im Internet: http://www.telelogic.se/products/somt.htm, Telelogic AB, Malmo, Schweden, 1996.

[FeHu93] Feiler, Peter H.; Humphrey, Watts S.: ***Software Process Development and Enactment: Concepts and Definitions***, Software Engineering Institute, Carnegie Mellon University (USA), 1993.

[FeVM97] Feldmann, Raimund L.; Vorwieger, Stefan; Münch, Jürgen: ***Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base***, Technischer Bericht Nr. 08/97, Sonderforschungsbereich 501, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, 1997.

[FeMV97] Feldmann, Reimund L.; Münch, Jürgen; Vorwieger, Stefan: ***Experiences with Systematic Reuse: Applying the EF/QIP Approach***, Proceedings of the European Reuse Workshop (ERW'97), Brüssel, Belgien, 26. - 27. November , 1997.

[Fold96] Howe, Denis: ***Foldoc - Free On-Line Dictionary Of Computing***, Adresse im Internet: http://www-igm.univ-mlv.fr/foldoc, Stand: 29. Mai 1996.

[FuGh94] Fugetta, A.; Ghezzi, C.: *State of the Art and Open Issus in Process-centred Software Engineering Environments*, **Journal of Systems and Software**, 26(1), Site 53-60, Juli 1994. (INF Z 4979)

[Gass90] Gasser, Walter: *Energie-Management in Geschäftsbauten*, **Schweizer Ingenieur und Architekt** Nr. 50 (108), 13. Dezember 1990, Seite 1436-1440. (ARB Z 2002)

[GeRö96] Geppert, Birgit; Rößler, Frank: ***Pattern-based Configuring of a Customized Resource Reservation Protocol with SDL***, Sonderforschungsbereich 501, Report 19/96, Fachbereich Informatik, Universität Kaiserslautern, 1996.

[Glas94] Glass, Robert L.: *The Software Research-Crisis*, **IEEE Software**, Seite 42-47, November 1994. (INF Z 4879)

[Halb93] Halbwachs, Nicolas: ***Synchronous Programming of Reactive Systems***, Kluwer Academic Publisher, 1993. (INF 483/034)

[HBGL95] Heitmeyer, Constance; Bull, Alan; Gasarch, Carolyn; Labaw, Bruce: *SCR*: A Toolset for Specifying and Analyzing Requirements*, **Proceedings of the 10th Annual Conference on Computer Assurance (COMPASS '95)**, Gaithersburg, MD (USA), Juni 1995. (INF 425/024-95)

[Heid97] Heide, Marco: ***Systemdokumentation des Heizungssteuerungssystems***, Projektarbeit, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, August 1997.

[HeJL96] Heitmeyer, Constance L.; Jeffords, Ralph D.; Labaw, Bruce G.: *Automated Consistency Checking of Requirements Specifications*, **ACM Transactions on Software Engineering and Methodology**, Volume 5, Nr. 3, Juli 1996. (INF Z 4785)

[Heni80] Heninger, Kathryn L.: *Specifying Software Requirements for Complex Systems: New Techniques and Their Application*, **IEEE Transactions on Software Engineering** Volume 6 Nr. 1, Seite 2-13, Januar 1980. (INF Z 4796)

[Huff96] Huff, Karen E.: *Software Process Modeling*, **Software Process**, Fuggetta und Wolf (Herausgeber), John Wiley & Sons Ltd., Chichester, 1996, Seite 1-24. (INF 410/366)

[Hump89] Humphrey, Watts S.: ***Managing the Software Process***, Addison-Wesley Publishing Company Inc., 1989. (INF 410/299)

[ITUT95] International Telecommunication Union (ITU) - Telecommunication Standardization Sector of ITU **(ITU-T):** ***Recommendation Z.105*** - SDL combined with ASN.1, (SDL/ASN.1), 1995.

[ITUT96a] International Telecommunication Union (ITU) - Telecommunication Standardization Sector of ITU **(ITU-T):** ***Recommendation Z.120*** - Message sequence chart (MSC), 1996.

[ITUT96b] International Telecommunication Union (ITU) - Telecommunication Standardization Sector of ITU **(ITU-T):** ***Recommendation Z.100*** - CCITT specification and description language (SDL), 04/96.

[ITUT97] International Telecommunication Union (ITU) - Telecommunication Standardization Sector of ITU **(ITU-T):** ***Recommendation Z.100 Addendum 1*** (10/96) zu [ITUT96], 1997.

[Jaco94] Jacobson, Ivar: ***Object-oriented software engineering: a use case driven approach***, ACM Press-Wesley, Reading, Massasusettes (USA), Addison-Wesley, 1994. (INF 418/029)

[Jarz97] Jarzabek, Stan: *Modeling Multiple Domains in Software Reuse*, **Software Engineering Notes**, Volume 22, No. 3, Proceedings of the 1997 Symposium on Software Reusability (SSR'97), Boston, Massachusetts (USA), 17-19. Mai 1997, acm press, Seite 65-74.

[KaCa93] Karam, G. M.; Casselman, R. S.: *A Cataloging Framework for Software Development Methods*, **Computer**, 26(2), Seite 34-45, 1990.

[Knab92] Knabe, Gottfried: ***Gebäudeautomation***, 1. Auflage, Verlag für Bauwesen, 1992. (ARB 384/030 und ELT 950/082)

[Kona78] Konakovsky, R.: *Safety Evaluation of Computer Hardware And Software*, **Proceedings of the IEEE Computer Society's Second International Computer Software & Applications Conference, COMPSAC'78**, Chicago, Illinois (USA), 13.-16. November 1978, Seite 559-564. (INF 151/115-2)

[Krue92] Krueger, Charles W.: *Software Reuse*, **ACM Computing Surveys**, Volume 24, No. 2, Juni 1992, Seite 131-183. (INF Z 1450)

[Leve86] Leveson, Nancy G.: *Software Safty: Why, What, and How*, **ACM Computing Surveys**, Volume 18, No. 2, Juni 1986, Seite 125-163. (INF Z 1450)

[Loud94] Louden, Kenneth C.: ***Programmiersprachen: Grundlagen, Konzepte, Entwurf***, International Thomson Publishing, 1. Aufl., 1994. (INF 430/092)

[LuPa90] Lute, P. J.; Van Paassen, D. H. C.: ***Integrated Control System For Low-Energy Buildings***, SL-90-16-3, Seite 889-895.

[McCl88] McClelland, Stephen (Editor): ***Intelligent Buildings*** - An IFS Executive Briefing, IFS Publications, Bedford und Springer Verlag, Berlin, 1988. (ARB 384/007)

[Mill97] Miller, Franz: *Das intelligente Haus*, **Der Frauenhofer** 1/97, Seite 4-9, 1997.

[MüSV97] Münch, Jürgen; Schmitz, Markus; Verlage, Martin: ***Tailoring großer Prozeßmodelle auf der Basis von MVP-L***, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, 1997.

[NASA94] National Aeronautics and Space Administration (NASA): ***Software Engineering Program: Profile of Software at the Goddard Space Flight Center***, Nasa-report NASA-RPT-002-94, Washington DC (USA), April 1994.

[Oste87] Osterweil, Leon: *Software Processes Are Software Too*, **Proceedings of the Ninth International Conference on Software Engineering**, 30. März - 2. April 1987, Monterey CA, Computer Society Press of the IEEE, Washington DC, 1987, Seite 2-13. (INF 157/430-9)

[PaMa95] Parnas, David Lorge; Madey, Jan: *Functional documents for computer systems*, **Science of Computer Programming**, Volume 25, Seite 41-61, 1995. (INF Z 4990)

[Pete98] Petersen, Erik: ***Formalisierung ausgewählter Software-Entwicklungsmethoden für reaktive Systeme in MVP-L***, Diplomarbeit, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, Januar 1998.

[PMvS93] Parnas, David Lorge; Madley, Jan; van Schouwen, A. John: *Documentation of Requirements for Computer Systems*, **Proceedings of the Requirements Engineering Symposium (RE'93)**, San Diego (USA), 1993.

[PriD89] Prieto-Díaz, Rubén: *Classification of Reusable Modules*, Volume II, Application and Experience in [BiPe89], Seite 99-123. (INF 410/297-2)

[PriD90] Prieto-Díaz, Rubén: *Domain Analysis: An Introduction*, **ACM Software Engineering Notes**, ACM SIGSOFT, Volume 15, No. 2, April 1990, Seite 47-54. (INF Z 4884)

[PrFr87] Prieto-Díaz, Rubén; Freeman, Peter: *Classifying Software for Reusability*, **IEEE Software**, IEEE Computer Society, Volume 4, No. 1, Januar 1987, Seite 6-16. (INF Z 4879)

[RBBL94] Rombach, H. Dieter; Birk, Andreas; Bröckers, Alfred; Lott, Christopher M.; Verlage, Martin: ***Qualitätsorientierte, prozeß-sensitive Softwareentwicklungsumgebungen im MVP-Projekt***, Interner Bericht 256/94, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, Oktober 1994.

[RBLV93] Rombach, H. Dieter, Bröckers, Alfred; Lott, Christopher M.; Verlage, Martin: *Entwicklungsumgebung zur Unterstützung qualitätsorientierter Projektpläne*, **Proceedings der Gesellschaft für Informatik (GI) Fachtagung Softwaretechnik 93**, Dortmung, November 1993.

[Romb91] Rombach, H. Dieter: ***MVP-L: A Language For Process Modeling In-The-Large***, Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland (USA), Technical Report UMIACS-TR-91-96, CS-TR-2709, Juni 1991.

[Romb94] Rombach, H. Dieter: ***Software Engineering II***, Skript zur Vorlesung, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, 1994.

[RoVe95] Rombach, H. Dieter; Verlage, Martin: ***Directions in Software Process Research***, Advances in Computers, Volume 41 (M. V. Zelkovitz, ed.), Boston, MA (USA), Academic Press, 1995.

[RuBP91] Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; Lorensen, William: ***Object-Oriented Modeling and Design***, Prentice-Hall Inc., Englewood Cliffs, New Jersey (USA), 1991. (INF 418/028)

[Rumb94] Rumbaugh, James: *The OMT Process*, **Rational Software Corporation**, Technical Support, Internet-Adresse: http://www.rational.com, Mai 1994.

[SchH97] Schneider, Hans-Jochen (Hrsg.): ***Lexikon Informatik und Datenverarbeitung***, R. Oldenbourg Verlag, Version 4, München, 1997. (INF 010/053)

[SchM97] Schmitz, Markus: ***Transformationsbasiertes Zuschneidern von Prozeßmodellen***, Diplomarbeit, AG Software Engineering, Fachbereich Informatik, Universität Kaiserslautern, Januar 1997.

[Schu89] Schulze, Hans Herbert: ***Computer Enzyklopädie***, Rowohlt (rororo), Reinbek, 1989. (INF 010/073)

[SDTM96] ***Online Manual SDT 3.1 Methodology Guidlines***, Telelogic, Schweden, Dezember 1996.

[SDTM97] ***Online Manual SDT 3.2 Methodology Guidlines***, Telelogic, Schweden, 1997.

[Spie95] *Wir stehen im Treibsand*, **Der Spiegel**, Nr. 4, Januar 1995.

[Tele97] Telelogic AB: ***Tau 3.2 User‘s Manual***, 1997.

[VDIR90] Verein Deutscher Ingenieure, ***VDI-Richtlinie 3814 Blatt 1, Gebäudeleittechnik (GLT) Strukturen, Begriffe,U Funktionen***, Düsseldorf, Juni 1990. (ARB 368/055-2 und MAS 898/018-3 und MAS 898/019-2)

[Wull90] Wullschleger, Rudolf: *Mehr Umweltqualität dank Gebäudeleittechnik*, **Schweizer Ingenieur und Architekt** Nr. 50 (108), 13. Dezember 1990, Seite 1469-1470. (ARB Z 2002)

[ZiSz95] Zilahi-Szabó, Miklós Géra (Hrsg.), ***Kleines Lexikon der Informatik***, R. Oldenbourg Verlag, München, 1995. (INF 010/095)