Integration von Softwarespezifikationen

Markus Zirbes
233 731

Betreut von Dipl.-Inform. Anne-Thérèse Körtgen

Zusammenfassung

Gemäß des vorherrschenden Ansatzes in der rationalen Systementwicklung, dem Viewpoint-Modell, sollen Entwicklungsprozesse auf Modellen basieren. Abstrahierte Funktionen und Verhaltensweisen in Form von Modellen sollen den kompletten Lebenszyklus eines Systems begleiten und so zur Verminderung des Entwicklungs- und Wartungsaufwandes beitragen. Dabei sollen die Modelle nicht das System als ganzes umfassen, sondern sich auf spezifische Aspekte konzentrieren. Bedingt durch die Betrachtung und Modellierung aus verschiedenen Blickwinkeln entstehen verschiedene Teilmodelle, die sich in Syntax und Semantik unterscheiden, aber wieder zu einem Systemmodell integriert werden sollen. Die Integration bedarf deshalb einer gemeinsamen semantischen Domäne, welche mächtig genug ist, die Teilmodelle sowohl zu interpretieren, als auch zu integrieren. Diese Domäne findet ihre Realisierung in Form eines Referenzmodells. Transformationssysteme können hierbei die formal-mathematische Grundlage darstellen und so zur Beweisbarkeit und Konsistenz der Richtigkeit der Modelle beitragen.

Inhaltsverzeichnis

1 Einleitung

Softwareentwicklungsprozesse bedürfen, bedingt durch die Komplexität der zu entwickelnden Systeme, handlicher Modelle. Diese Modelle können immer nur Teilmodelle einzelner Aspekte des Systems sein und ihre Entwicklung erfolgt zunächst blickwinkelabhängig. Im Laufe eines Softwareentwicklungsprozesses entsteht so eine Vielzahl heterogener Teilmodelle, deren Funktionen und Eigenschaften Überlappungen und Redundanzen aufweisen können. Dies wirft die Frage nach der Integration dieser Teilmodelle, sowie ihrer Überprüfbarkeit im Rahmen des Systems auf. Die Integration in Form eines formal-mathematischen Referenzmodells soll die thematische Grundlage dieser Ausarbeitung bilden.

Im Kontext des Seminars stellt dieses Thema den theoretischen Überbau dar, der für alle übrigen Themen von Relevanz ist. Gegenüber den übrigen Seminarthemen, die sich in sehr konkreter Weise auf verschiedene Ausgestaltung modellgetriebener Entwicklungsprozesse und deren Werkzeuge beziehen, befasst sich diese Ausarbeitung mit den theoretischen Grundlagen, die die Integration dieser Teilergebnisse ermöglichen und die Teilmodelle auf konsistente Art zusammenführen soll.

1.1 Einführung

Die Spezifikation eines nicht-trivialen verteilten Softwaresystems umfasst eine sehr große Menge an Informationen, die es unmöglich macht, alle Informationen und Aspekte des Systems in einer einzigen Beschreibung zu erfassen. Das *Reference Model of Open Distributed Processing* (RM-ODP) [1] soll hier in Form einer ISO-Norm Abhilfe schaffen, welche Funktionen und Konzepte definiert, bei deren Beachtung in der Planungs- und Umsetzungsphase ein qualitativ hochwertiges, verteiltes System entstehen soll. Zur Reduzierung der Komplexität sieht das RM-OPD vor, ein System aus verschiedenen Viewpoints (Sichtwinkeln) zu betrachten und zu modellieren, macht aber diesbezüglich keine Vorschriften über die dazu zu verwendenden Technologien. Die sich daraus ergebenden Produkte der verschiedenen Blickwinkel und ihrer jeweiligen Akteure sind Modelle.

Ein Modell ist im Wortlaut Große-Rhodes eine abstrakte Repräsentation von etwas, das von jemandem für einen bestimmten Zweck entwickelt wurde. In diesem Sinne repräsentiert oder beschreibt ein Modell etwas, eine Struktur, ein Verhalten, etc., welches sich vom Modell selbst unterscheidet. Ein Klassendiagramm wird z.B. nicht zur Umschreibung eines anderen Klassendiagramms benutzt, aber möglicherweise zur Beschreibung der Zustände eines Objektes [4].

1.2 Aufbau der Arbeit

Zunächst sollen in Kapitel 2 das RM-ODP und die darin definierten Qualitätskriterien für ODP-Systeme eingeführt werden. Das Viewpoint Modell, ein maßgeblicher Bestandteil des RM-ODP, welcher die Systementwicklung aus verschiedenen Sichten propagiert, wird den Anknüpfungspunkt in Kapitel 2.2 liefern. Nach der Vorstellung der einzelnen Viewpoints wird die sich daraus ergebene Integrationsproblematik in Kapitel 3 das Thema sein, wie die Vielzahl heterogener Teilspezifikationen zu einem gemeinsamen Systemmodell zusammengeführt werden können. Es werden Anforderungen an eine Referenzmodell vorgestellt, welches die konsistente Zusammenführung der einzelnen Teilmodelle ermöglichen soll. In Kapitel 4 werden Transformationssysteme als formale Grundlage zur Ausgestaltung dieses Referenzmodells eingeführt sowie deren Komposition vorgestellt. Zum Abschluss wird diese Arbeit in Kapitel 5 zusammengefasst und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Das RM-ODP

Das RM-ODP ist eine ITU-Empfehlung[1] sowie ein ISO/IEC[2]-Standard, welcher ein Referenzmodell für Spezifikation und Entwurf von offenen verteilten Systemen definiert. Das RM-ODP gibt einen allgemeinen Rahmen für die Modellierung und Architektur von ODP-Systemen [3] vor und wendet sich damit sowohl an die Entwickler von ODP-Systemen als auch an die Entwickler anderer Standards, welche Teile von ODP-Systemen sind, z.B. zur Festlegung von Daten- und Dateiformaten, Protokollen etc. Dieser Rahmen bleibt vollständig abstrakt, da er als Referenzmodell auch für zukünftige Anwendungen dienen soll, deren Gestalt noch nicht vorhersehbar ist und nicht eingeschränkt werden soll.

Es soll zugleich Qualitätssicherung und -management dienen, indem es Standards festlegt, wie Produktionsprozesse modelliert, organisiert und dokumentiert werden. Es legt einen Rahmen zur Beschreibung der ODP-Komponenten sowie deren Interrelationen fest und konstatiert einen Bedarf an möglichst formalen Spezifikationen. Das RM-ODP ist weder Integrationsplattform noch Rahmenwerk sondern ein abstraktes Modell, auf das bei der Entwicklung und Beschreibung eines ODP-Systems referenziert (*Referenz*modell [3]) werden soll. Abstrakt bedeutet in diesem Zusammenhang, dass Modellierungskonzepte und eine allgemeine Architektur festgelegt werden, aber keine konkreten Spezifikations- und Programmier-

[1] ITU: International Telecommunication Unit; X.901 bis X.904
[2] IEC: International Electrotechnical Commission; 10746-1 bis 10746-4

sprachen vorgeschrieben werden. In Kapitel 2.1 sollen nun die wünschenswerten Eigenschaften von ODP Systemen aufgezeigt werden, deren Fülle einer praktikablen Herangehensweise bedarf, welche in Form des Viewpoint-Modells in Kapitel 2.2 ihre Umsetzung findet.

2.1 Eigenschaften von ODP-Systemen

RM-ODP zielt auf die Verbesserung offener verteilter Systeme ab. Daher seien zunächst bestehende und erwünschte Eigenschaften solcher Systeme aufgelistet.

Die **Inhärenten Eigenschaften** verteilter Systeme, sind jene, die sich aus der Verteiltheit ergeben:

- **Entfernung**: Komponenten eines verteilten Systems können räumlich verteilt sein, Interaktionen können lokal oder über Entfernungen stattfinden.
- **Nebenläufigkeit**: Jede Komponente eines Systems kann parallel mit anderen Komponenten ablaufen.
- **Fehlen eines globalen Zustands**: Ein globaler Zustand eines verteilten Systems kann nicht genau ermittelt werden.
- **Partielles Fehlverhalten**: Jede einzelne Komponente eines verteilten Systems kann unabhängig von anderen fehlerhaftes Verhalten zeigen bzw. ausfallen.
- **Asynchronität**: Kommunikation und Synchronisation werden nicht durch eine globale Systemzeit gesteuert.

Weitere Eigenschaften, die charakteristisch für große ODP-Systeme sind, wie z.B. Systeme, die sich über mehrere Organisationen bzw. Unternehmen erstrecken, lauten:

- **Heterogenität**: Die Komponenten eines ODP-Systems basieren üblicherweise nicht auf der gleichen Technologie, darüber hinaus kann sich die Technologie einer Komponente ändern. Heterogenität betrifft Hardware, Betriebssysteme, Kommunikationsnetze und Protokolle, Programmiersprachen etc.
- **Autonomie**: Es gibt in einem ODP-System i.A. kein zentrales Management, d.h. Gruppen von Komponenten agieren autonom.

- **Evolution**: Während des Lebenszyklus eines ODP-Systems können sich Komponenten aufgrund von technischen Verbesserungen, neuen Anwendungen etc. verändern.

- **Mobilität**: Ressourcen können physisch mobil sein.

Die **erwünschten Eigenschaften**, die durch die Entwicklung und Beschreibung gemäß RM-ODP zu erzielen sind, lauten:

- **Offenheit**: Ein System ist offen, wenn es möglich ist, Komponenten ein- und auszubauen. Insbesondere unterstützt es Portablilität (d.h. die Möglichkeit, verschiedenen Komponenten von verschiedenen Prozessoren ohne Modifikation ausführen zu können) und *interworking* (d.h. sinnvolle Interaktionen zwischen Komponenten verschiedener Systeme).

- **Integration**: Die Zusammensetzung von verschiedenen Systemen und Ressourcen zu einer Applikation soll den Umgang mit Heterogenität unterstützen.

- **Flexibilität**: Sie soll der Evolution eines Systems dienen und Laufzeitänderungen sowie dynamische Rekonfigurationen zulassen.

- **Modularität**: Das System besteht aus autonomen aber verbundenen Komponenten, welche die Basis für die Flexibilität darstellen.

- **Föderation**: Die Kombination von Systemen verschiedener Herkunft (verschiedener Organisationen) für eine Applikation soll gewährleistet sein.

- **Handhabbarkeit**: Monitoring, Kontrolle und Management der Ressourcen eines Systems zur Unterstützung von Konfiguration, Dienstqualität und Zugriffsnormen sollen gegeben sein.

- **Dienstqualität**: Sie soll durch zeitliche, Verfügbarkeits- und Zuverlässigkeitsqualität, sowie Fehlertoleranz gewährleistet werden.

- **Sicherheit**: Die Sicherheit soll beispielsweise durch Authentifizierung und Zugriffskontrolle gewahrt werden.

- **Transparenz**: Das Verstecken von Details und Mechanismen, die durch die Verteiltheit des Systems bedingt sind, wie z.B. Heterogenität der Soft- und Hardware, Ort und Mobilität von Komponenten, Erreichen der Dienstqualitäten, Replikation, Migration etc., soll für Transparenz sorgen.

Diese Fülle an Eigenschaften gibt bereits einen Eindruck von der zu erwartenden Größe und Komplexität eines ODP-Systems, das praktisch nicht durch *eine* Spezifikation beschrieben werden kann [3]. Das RM-ODP führt deshalb das Konzept der *viewpoints* ein, also die Unterscheidung verschiedener Systemaspekte in Form von fünf verschiedenen Aspektspezifikationen, welche das Thema des folgenden Kapitels sein sollen.

2.2 Das Viewpoint-Modell

Eines der Grundprinzipien der Systementwicklung, das bevorzugt durch die UML [6] (*Unified Modeling Language*, eine standardisierte Sprache für die Modellierung von Software und anderen Systemen) realisiert wird, stellt das Viewpoint-Modell dar. Das Viewpoint-Modell umfasst zwei Hauptaspekte, zum einen, dass Softwareentwicklung auf Modellen basieren soll, zum anderen, dass das System nicht in einem einzigen Modell erfasst wird, sondern dass sich verschiedene Modelle auf unterschiedliche Teilaspekte des Systems konzentrieren sollen.

Der erste Aspekt besagt, dass Modelle eine abstrakte Repräsentation des Systems, seiner Strukturen, Funktionen und Eigenschaften liefern sollen und das, ohne die Notwendigkeit zur detaillierten Ausprogrammierung, denn die eigentliche Umsetzung soll hierbei noch offen bleiben. Modelle, die hierzu in Betracht kommen, sind beispielsweise Anwendungsfall- und Klassendiagramme, aber auch (mathematisch) formalere Methoden. Unter anderem werden CSP (Communicating Sequential Processes, eine Prozessalgebra) und Petrinetze genannt. Die entstehenden Modelle sollen aber nicht nur als Hilfsmittel für die frühe Entwicklungsphase angesehen werden, sondern sollen den kompletten Lebenszyklus einer Software unterstützend begleiten. Wenn jede Eigenschaft des Systems und jede Entwurfsentscheidung in Form von Modellen dokumentiert ist, kann sich jede Weiterentwicklung zunächst an diesen abstrakten Repräsentationen orientieren, ohne sich mit Programmcode auseinander setzen zu müssen, in welchem die zugrunde liegenden Strukturen nur schwer nachzuvollziehen wären. Entwicklungskomplexität und Wartungsaufwand werden also durch Abstrahierung in Form von Modellen reduziert.

Auch der zweite Aspekt, dass sich jedes Modell auf Teilaspekte des Systems konzentrieren soll, trägt letztendlich dazu bei, die Komplexität zu reduzieren. Diese Teilaspekte unterliegen verschiedenen Domänen, also Akteuren mit unterschiedlichen Arbeitsschwerpunkten und Interessen, so z.B. Softwareentwickler, Systemingenieure und Prozessmanager, die jeweils anderer Informationen und Abstraktionen bedürfen. Das RM-ODP führt diesbezüglich eine Klassifikation ein, die für alle Systementwicklungen Gültigkeit haben soll. Nach [5] sollen die folgenden

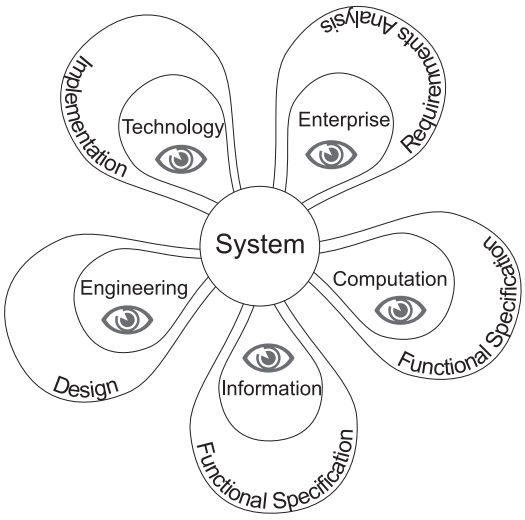

Abbildung 1: Die fünf verschiedenen Sichten (*views*) des Viewpointmodells.

fünf Viewpoints (vgl. Abbildung 1) bei der Betrachtung eines Systems unterschieden werden:

1. Im Enterprise Viewpoint werden das Gesamtvorhaben, der Umfang und die Verhaltensnormen (*policies*) des Systems spezifiziert. Dabei werden das System und seine Komponenten als Gemeinschaft interagierender Komponenten modelliert. Agenten sind hierbei Objekte, die Aktionen initiieren können, also nicht nur passiv sind. Die Verhaltensnormen machen Vorgaben in Bezug auf ein bestimmtes Vorhaben, welche als Verpflichtung, Erlaubnis oder Verbot formuliert sind. Verhaltensnormen können beispielsweise den Gebrauch von Ressourcen oder die Sicherheit betreffen.

2. Der Informational Viewpoint spezifiziert die Semantik der Information und Informationsverarbeitung in Form eines Informationsmodells. Eine Spezifikationssprache für den information viewpoint muss Konzepte für die Spezifikation von statischen, invarianten und dynamischen Schemen bereitstellen. Statische Schemen beschreiben feste Zustände von Informationsobjekten, so z.B. initiale Zustände. Invariante Schemen beschreiben Zustandsinvarianten, also solche Eigenschaften, die immer gelten. Dynamische Schemen beschreiben Zustandsübergänge, insbesondere das Erzeugen und Löschen von Objekten. Die im information viewpoint spezifizierten Interaktionen der Informationsobjekte ergeben Konformanzanforderungen an die Implementierungen dieser Objekte durch Konfigurationen von Objekten im computational und engineering viewpoint.

3. Der Computational Viewpoint beschreibt die funktionale Dekomposition des Systems in Objekte, die über Schnittstellen interagieren. Die Schnittstellen beinhalten drei Komponenten: (1) Die Signatur, (2) das Verhalten und (3) den Vertrag mit der Umgebung. Die Ausprägung einer Schnittstelle hängt von ihrem Typ ab: So gibt es Operations-, Fluß- und Signalschnitstellen. Im computational viewpoint wird außerdem die Bindung zwischen Objekten spezifiziert, welche explizit oder implizit erfolgen kann. Implizit geschieht dies in Form eines Binder-Objekts, dass die Bindung herstellt und kontrolliert, explizit durch einen Methodenaufruf, der z.B. durch einen Trader vermitteln kann.

4. Der Engineering Viewpoint spezifiziert die Mechanismen und Funktionen, die die verteilte Interaktion zwischen den Objekten ermöglicht und somit die geforderten Transparenzen realisiert und kann als eine abstrakte Maschine angesehen werden. Eine Spezifikationssprache für den engineering viewpoint muss Konzepte zur Verfügung stellen, welche die Strukturierung von Kommunikationskanälen zwischen Objekten und die Strukturierung der Ressourcen bzw. die Konfiguration von Objekten erlaubt.

5. Der Technology Viewpoint spezifiziert die Implementierung des Systems als Konfiguration von Objekten, die die Soft- und Hardwarekomponenten der Implementierung repräsentieren.

Jeder Viewpoint abstrahiert nur einen Teil des Systems, indem er einige Aspekte hervorhebt und andere aus lässt. Die Sichtweisen sind hierbei nicht unabhängig voneinander zu betrachten, da sie zur Beschreibung eines gemeinsamen Systems der Konsistenz zueinander bedürfen [1]. Der ODP-Standard sieht weiterhin vor, dass die Modellierung der einzelnen Viewpoints durch eine adäquate (domänenspezifische) Sprache erfolgen soll. Diese Sprachen sollen Repräsentation der jeweils relevanten Eigenschaften so direkt und formal wie möglich ausdrücken, um eine formale Überprüfbarkeit zu gewährleisten und damit auch den Einsatz von Überprüfungswerkzeugen zu ermöglichen.

Aus der Entwicklung mit Hilfe des Viewpoint Modells ergeben sich jedoch auch Probleme, die Große-Rhode mit der folgenden Aussage zusammenfasst: „*Many* people develop *many* models in *many* languages of *many* views of **one** system.“ Viele Menschen entwickeln viele Modelle in vielen Sprachen aus vielen Blickwinkeln für ein System [5].

3 Integrationsproblematik

Um zu gewährleisten, dass die Vielzahl heterogener Modelle **ein** und das selbe System beschreiben, sind die beiden folgenden Teilprobleme zu unterscheiden: Das erste Teilproblem betrifft die (abstrakte) Integration der Modelle, also wie die verschiedene Teilmodelle zu einem gemeinsamen Systemmodell zusammengeführt werden können. Beispielsweise kann ein Modell ein anderes Modell ergänzen, indem es es bisher unbetrachtete Aspekte dem Modell hinzufügt und es somit verfeinert. Die Sequenzdiagramme der UML ermöglichen so z.B. die Verfeinerung der Informationen, die Anwendungsfalldiagramme bereitstellen. Die Objekte, deren Funktionalitäten in Anwendungsfällen spezifiziert wurden, können somit durch Sequenzdiagramme in eine Interaktionsbeziehung zueinander gebracht werden. Weiterhin können, in Folge der Betrachtung aus verschiedenen Blickwinkeln, Modelle Informationen liefern, die sich mit denen anderer Modelle überlappen. Auf der einen Seite können beispielsweise Sequenzdiagramme die Interaktionen von Objekten in bestimmten Szenarios modellieren, wogegen sich andere Modelle in Form von Zustandsdiagrammen auf das (innere) Objektverhalten eines dieser Objekte und dessen Möglichkeiten konzentrieren. Die daraus entstehende semantische Anforderung ist es zu prüfen, ob die Objekte gemäß ihres in Form von Zustandsdiagrammen modellierten Potentials die Anforderungen erfüllen können, die sich aus den Sequenzdiagrammen ergeben.

Das zweite Teilproblem bezieht sich auf die Konsistenz der gegebenen Modelle, so z.B., ob die verschiedenen Modelle untereinander frei von Widersprüchen, also konsistent sind. Im logischen Sinne besagt die Konsistenz, dass es ein gemeinsames Modell für eine gemeinsame semantische Interpretation der Teilmodelle gibt. Im Kontext der Modelle, die sich heterogenen Spezifikationssprachen bedienen, sind dies Modellsprachen aber oft schon per Definition disjunkt, was sie im streng logischen Sinne inkonsistent macht. Der von Große-Rhode favorisierte Integrationsansatz nimmt diese Hürde, indem er sich allein auf die Semantik der Spezifikationen konzentriert und eine gemeinsame semantische Domäne entwickelt, die es ermöglichen soll, alle Sprachen zu interpretieren, die zur Spezifikation verwendet wurden. Damit wird jede Sprache Teil einer gemeinsamen, globalen Semantik. Große-Rhode nennt dieses übergeordnete Konstrukt Referenzmodell [5] und es erfüllt das Konsistenzkriterium, denn eine Gruppe von Spezifikationen gilt als konsistent, wenn es ein Modell für alle Spezifikationen dieser Gruppe gibt.

3.1 Konsistenzanalyse

Im Kontext der Modellintegration definiert Große-Rhode die Konsistenz wie folgt: Eine Menge von Spezifikationen gilt als konsistent, wenn es ein gemeinsames Modell aller Spezifikationen dieser Menge gibt [5]. Die konsistente Integration soll im Folgenden mit Hilfe eines Beispiels veranschaulicht werden. Abbildung 2 zeigt einen kleinen funktionellen Ausschnitt eines Systems, welches durch vier verschiedene Ansätze dargestellt wird: In Form einer Programmiersprache, einer Klasse, eines algebraischen Petrinetzes und eines Zustandsgraphen.

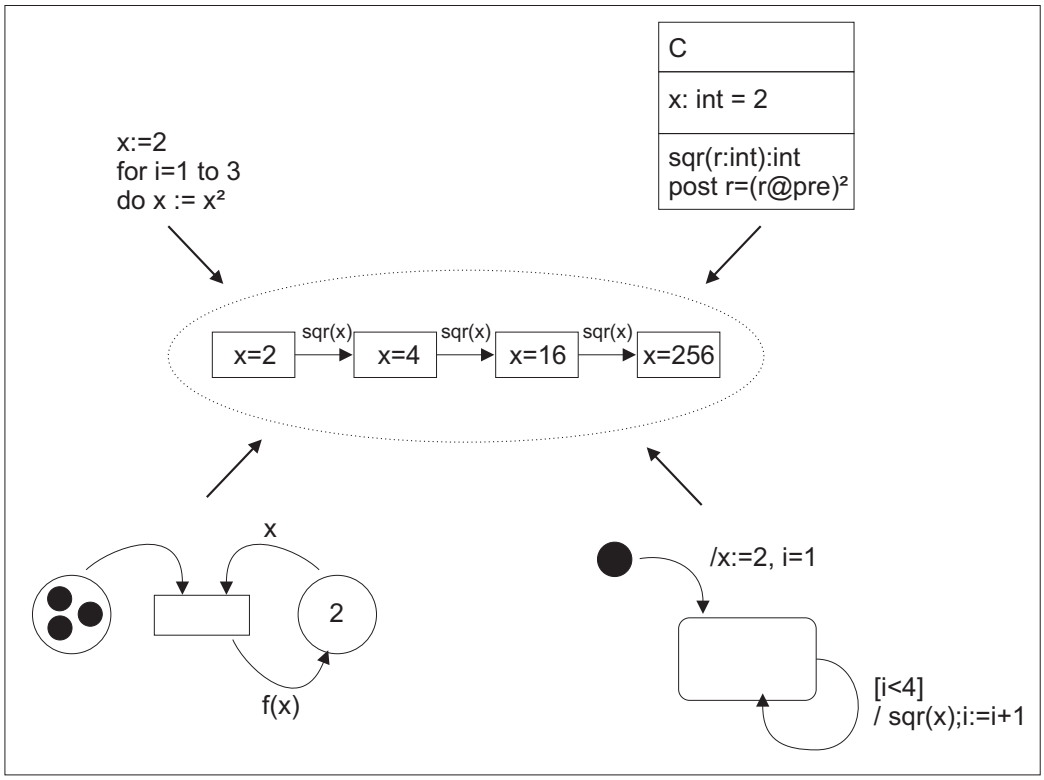

Abbildung 2: Die Abbildung zeigt die Interpretation von verschiedenen Beschreibungsformen (von oben links nach unten rechts: Programmcode, Klassendiagramm, algebraisches Petrinetz und Zustandsgraph) in Form eines gemeinsamen formalen Modells im Zentrum.

Die Klasse definiert die Struktur durch ein Attribut `x` vom Typ Integer und die Methode `sqr`, ebenfalls vom Typ Integer. Weiterhin spezifiziert es den Startwert 2 für `x` und die Methode `r = (r@pre)`^{2}, welche in der OCL (*object constraint language of the UML*) formuliert wurde. Der Ausdruck `r@pre` bezeichnet hierbei den Wert des Parameters `r`, bevor die Methode `sqr` angewendet wurde. Dadurch wird

nicht nur die reine Struktur, sondern auch ein Teil des Verhaltens durch die Klasse spezifiziert. Sie macht aber keine Aussage darüber, wann und wie die Methode angewendet werden soll.

Der Programmcode oben links im Bild und der Zustandsgraph unten rechts, spezifizieren das Verhalten in sehr ähnlicher Weise. Sowohl die Attribute (*data states*) als auch der Kontrollfluss werden durch Variablen modelliert, deren Werte manipuliert werden. Zusätzlich zu der impliziten Strukturinformation wird hier auch dynamisches Verhalten spezifiziert: Die sqr-Methode kommt dreimal in Folge zur Ausführung. Der Hauptunterschied der beiden Beschreibungsformen liegt darin, dass das Programm rein textuell, wogegen der Zustandsgraph gemischt, also graphisch und textuell ist. Auffällig ist, dass nicht nur die Typisierung von x sowohl im Programm als auch im Zustandsübergraphen mit den Informationen überlappen, die die Klasse bereitstellt, sondern auch ihr Initialisierungswert. Andererseits wird keine Information über den Effekt der Methode sqr auf die Variable x gegeben, allein der Name gibt Aufschluss auf die beabsichtigte Funktion.

Das Petrinetz modelliert das selbe dynamische Verhalten auf eine etwas andere Weise. Statt eine neue Zählvariable einzuführen, wie i im Programmcode, wird der Kontrollfluss durch Einführung einer neuen Stelle (place) realisiert, welche drei Tokens beinhaltet. Eines der Tokens wird jeweils dann konsumiert, wenn die Funktion f angewendet wird. Der Effekt von f wird hier ebenfalls nicht spezifiziert. Setzt man das Petrinetz jedoch in Beziehung zu den sqr-Methoden der anderen Spezifikationen, wird diese Information durch die Bedingungen in der Klassendefinition ergänzt. Die Integration der partiellen Spezifikationen kann jede von ihnen bereichern.

Durch die Entwicklung eines Refernzmodells kann die Konsistenz der vier Spezifikationen anhand eines gemeinsamen Modells gezeigt werden. Dieses Modell wird im Zentrum der Abbildung 2 dargestellt. Es besitzt vier Zustände, welche die jeweils aktuellen Werte von x beinhalten und durch Transitionen verbunden sind, welche mit der Methodenanwendung sqr(x) beschriftet sind. Dieses Modell ist eine mögliche Interpretation der operationalen Semantik des Programmcodes sowie des Zustandsgraphen, indem es von der Kontrollflussvariablen i abstrahiert. Analog repräsentiert es auch die operationale Semantik des Petrinetzes, indem die Funktion f als sqr(x), also x^2, interpretiert wird.

Da die Klasse kein dynamisches Verhalten spezifiziert, lässt sie eine Reihe möglicher Modellierungen im Referenzmodell zu, wogegen die anderen Spezifikationen normalerweise so interpretiert werden, dass sie nur ein einziges Verhalten modellieren. Doch auch diese können auf verschiedenen Weisen interpretiert werden, was z.B. darin zum Ausdruck gekommen ist, dass die Behandlung der Funktion f im Petrinetz zunächst im unklaren blieb. Erst das in Bezug setzen zur Klasse hat

die Interpretation der Funktion beeinflusst, da es nahe lag, dass der Methodenrumpf der Klasse mit der Funktion im Petrinetz korrespondiert. Ähnlich verhält es sich mit den Variablen x und i, die im Referenzmodell unterschiedlich behandelt werden: x als statische Entität, wogegen i in dynamisches Verhalten überführt wurde. Auch diese Unterscheidung wurde durch das Vergleichen mit der Klasse getroffen, welche nur das Attribut x beinhaltet. Diese lokalen Entscheidungen auf Grundlage möglicher Interpretationen müssen berücksichtigt werden, wenn verschiedene Gruppen von Spezifikationen verglichen werden. Hier wird deutlich, warum Konsistenzchecks nicht gänzlich automatisiert werden können aber kontextabhängigen (Korrespondenz-)Informationen bedürfen.

Die Voraussetzung zur Anwendung eines Referenzmodells ist, dass die Semantiken aller relevanten Sepzifikationssprachen in der Sprache des Referenzmodells wiedergegeben werden können. Diese Sprache muss entsprechend vollständig und ausdrucksstark sein [4].

Die Fragestellung nach der Konsistenz einzelner Viewpoint-Spezifikationen war Gegenstand zahlreicher Forschungsprojekte an der University of Kent at Canterbury. Ausgehend von der Feststellung, dass traditionelle Definitionen der Konsistenz nicht einfach auf diesen Kontext angewendet werden können, wurde ein Rahmenwerk zur Unterstützung von Konsistenzchecks der ODP-Viewpoints entwickelt. Dabei wurden die folgenden zusätzlichen Anforderungen formuliert, die ein solches Rahmenwerk erfüllen muss:

- „The consistency definition is applicable *intra language* for different formal description techniques“, etwa: Die Konsistenzdefinition ist beispielsweise für zwei verschiedene Modelle der selben formalen Beschreibungssprachen anwendbar.
- „It is also applicable *inter language* between different formal description techniques“, Die Konsistenzdefinition ist auf verschiedene formale Beschreibungssprachen anwendbar.
- „It supports different classes of consistency checks, depending on the viewpoints considered and the relation between these viewpoints“, etwa: Es unterstützt verschiedene Klassen von Konsistenzchecks abhängig von den zu betrachtenden Viewpoints und der Beziehung zwischen diesen Viewpoints.
- „It supports global consistency, i.e. checks arbitrary numbers of specifications instead of only binary checks.“, etwa: Es unterstützt globale Konsistenz, z.B. indem es eine beliebige Zahl von Spezifikationen prüft, statt nur binäre Überprüfungen auszuführen.

- „It allows viewpoints to relate to the target system in different ways“, etwa: Es erlaubt Viewpoints, sich auf verschiedene Weise auf das Zielsystem zu beziehen.

Der Schlüssel dieser Integration liegt in der Betrachtung der verschiedenen Entwicklungsbeziehungen, wie Verfeinerung, Implementierung und Äquivalenzen, um daraus eine Vereinigung der Spezifikationen zu finden. Die Existenz einer gemeinsamen Vereinigung dieser Ansammlung verschiedener Spezifikationen genügt der Konsistenz und zeigt zugleich, welche Merkmale dieser verschiedenen Spezifikationen mit denen anderer korrespondieren. Der Konsistenzansatz beginnt damit, Spezifikationssprachen in Hinblick auf ihre Semantik und die Entwicklungsbeziehungen zu untersuchen. In diesem Sinne ist die Betrachtung dual, im Vergleich zum Ansatz der Transformationssysteme, die sich zunächst nur auf die semantische Analysen konzentrieren, die sich nicht auf bestimmte Sprachen beschränken. Erst aus dem Ergebnis der semantischen Vergleiche sollen syntaktische Repräsentationen innerhalb konkreter Sprachen gegeben werden. Die Idee des Konsistenzansatzes hat jedoch die Entwicklung von Transformationssystemen maßgeblich beeinflusst und da die Ansätze komplementär sind, sieht Große-Rhode gute Chancen, die beiden Ansätze endgültig zu vereinigen [5].

3.2 Anforderungen an ein Referenzmodell

Für die interne Struktur eine Referenzmodells zur Integration heterogener Spezifikationen können zunächst einige allgemeine Anforderungen definiert werden: Zum einen muss es Elemente liefern, die die Funktion formaler semantischer Modelle der Elemente des Systems einnehmen. Das bedeutet auf der einen Seite, das Struktur und Verhalten dieser Systemelemente repräsentiert werden können, auf der anderen Seite aber auch, dass die Granularität gemäß des Grades der gewünschten Abstrahierung angepasst werden kann.

Weiterhin muss ein Referenzmodell auch die Entwicklung solcher Elemente ausdrücken können, um verbessernde Modellierungsentscheidungen nachvollziehen zu können. In diesem Zusammenhang bedarf es auch Operationen, die Extension und Reduktion der Elemente ermöglichen, das heißt, dass gegebene Elemente sowohl verfeinert (konkretisiert) als auch abstrahiert werden können.

Zuletzt bedarf es noch Operationen zur Komposition, die es ermöglichen, aus gegebenen Elementen größere, komplexere Elemente zu konstruieren. Dazu muss zunächst definiert werden, wie Elemente verbunden werden können. So muss beispielsweise die Definition von Beziehungen der Strukturen und Verhaltensweisen unterstützt werden. Diese Elementbeziehungen repräsentieren die Architektur des

Systems. Darüber hinaus muss das Ergebnis einer Komposition definiert werden können, also ein Element, das das System aus miteinander verbundenen Elementen repräsentiert. Durch solch ein übergeordnetes Element kann die innere Struktur, also das Systems aus kleineren Elementen, versteckt werden.

Die Sprache eines solchen Referenzmodells muss allgemeingültig genug sein, um alle anderen Modellsprachen, sowie deren eventuell schon vorhandene formale Definition, abbilden zu können. Aus dieser Abbildung in Form einer Redefinition sollen aber immer noch Rückschlüsse auf die zugrunde liegende Semantik des Ursprungselements möglich sein. Es ist nicht Sinn und Zweck eines Referenzmodelles, neue, verbesserte Semantiken zu entwickeln, aber es kann vorkommen, dass durch den Prozess der Integration Verbesserungsvorschläge für die Ausgangselemente hervorgebracht werden. Elemente, die bisher nur informell definiert wurden, können jedoch durch das Referenzmodell ihre formale Ausgestaltung erhalten [5].

4 Transformationssysteme

Im Rahmen des Forschungsprojektes IOSIP[3] (Integration objektorientierter Softwarespezifikationstechniken und deren anwendungsspezifische Erweiterung für industrielle Produktionssysteme) wurde ein Referenzmodell zur Integration heterogener Softwarespezifikationen entwickelt, das einen formal-semantischen Ansatz unterstützt. Dazu wurde eine globale semantische Domäne entwickelt, die die direkte Interpretation und den Vergleich verschiedener Spezifikationen zulässt. Die Elemente dieser semantischen Domäne sind Transformationssysteme [2], die im Folgenden eingeführt werden sollen.

4.1 Einführung in Transformationssysteme

Ein Transformationssystem ist ein erweitertes *Transitionssystem*, z.B. ein Graph, in welchem nicht nur die Transitionen[4] sondern auch die Zustände beschriftet werden. Transitionssysteme kommen insbesondere bei der Definition operationaler Semantiken verschiedener Sprachen und Systeme zum Einsatz, z.B. für imperative oder funktionale Programmiersprachen, aber auch als Spezifikationssprachen für Prozesse und Datentypen. Transformationssysteme werden in diesem Zusammenhang als formale Modelle gewählt, da sie den kleinsten gemeinsamen Nenner

[3] http://tfs.cs.tu-berlin.de/SPP/iosip.html
[4] Transition (lat. Übergang)

für semantische Modelle darstellen. Die geschickte Definition der Transitions- und Zustandsbeschriftungen erlaubt so eine flexible Anpassung an die verschiedenen Anwendungsformen.

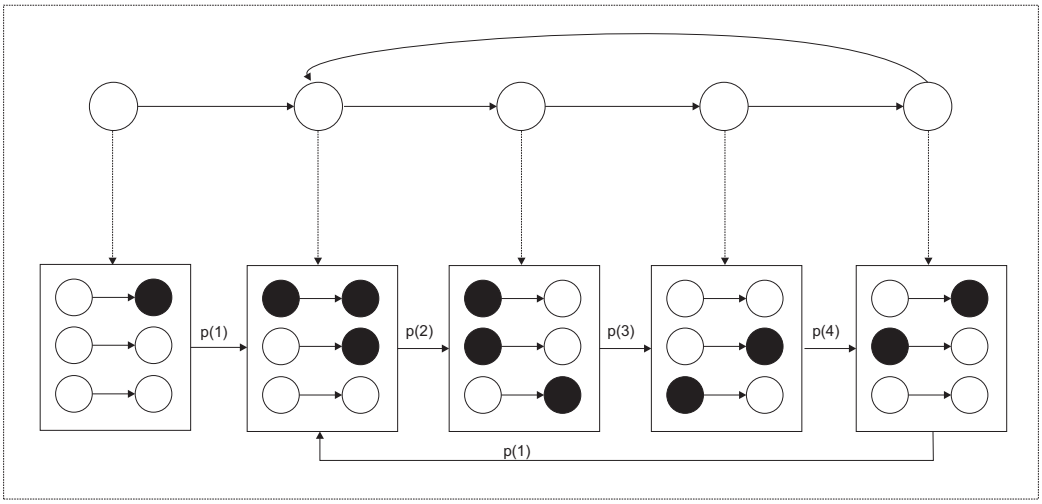

Abbildung 3: Die Abbildung zeigt ein einfaches Transformationssystem das eine Ampelschaltung modelliert.

In ihrer formalen Struktur repräsentieren Transformationssysteme die allgemeine Dualität von dynamischem Verhalten und statischer Struktur. Dieser Dualität entsprechend beinhalten Transformationssysteme den Transformationsgraphen, der das Gerüst des dynamischen Verhaltens darstellt, sowie die Datenräume (*data spaces*), welche wiederum Datenzustände (*data states*) und Ablaufstrukturen (*action structures*) beinhalten und die statische Struktur abbilden.

Der Transitionsgraph beinhaltet eine Menge von abstrakten Kontrollzutänden, repräsentiert durch Knoten und den möglichen Transitionen zwischen diesen Zuständen in Form von Kanten und ermöglicht so die Modellierung der zeitlichen Abfolge der auszuführenden Einzelschritte des Systems.

Die Abbildung 3 zeigt ein einfaches Transformationssystem, welches eine Ampelschaltung modelliert. Das Transformationsystem besteht aus dem Transformationsgraphen oben und den Datenräumen darunter. Die Datenräume wiederum bestehen aus Datenzuständen in Form der Knoten und aus Ablaufstrukturen in Form von Transitionen und Methodenaufrufen, wie z.B. `p(1)`, welcher soviel bedeutet wie: Schalte Ampelphase 1 (rot geht über in rot-gelb). Die Knoten und Kanten innerhalb der Datenräume repräsentieren die Variablen für die Zustände der Ampelphasen. Jede Zeile steht für eine Farbe der Ampel, also rot, gelb und grün. Ein ausgefüllter schwarzer Knoten bedeutet, dass diese Farbe eingeschaltet ist.

Der Datenraum stellt alle möglichen Datenzustände und Transformationen zur Verfügung, unabhängig davon, ob sie vom System erreicht oder ausgeführt werden (wie z.B. rot nach rot im zweiten Datenraum von Abbildung 3). Die Auswahl der Datenzustände und Transformationen aus dem Datenraum erfolgt mit Hilfe des Transitionsgraphen, der zugleich ihre sequentielle Abfolge repräsentiert. Eine Beschriftungsfunktion (gestrichelte Assoziationspfeile) soll dafür sorgen, dass mit jedem Kontrollzustand ein Datenzustand und mit jeder Transition eine Datentransformation assoziiert werden kann.

4.2 Komposition von Transformationssystemen

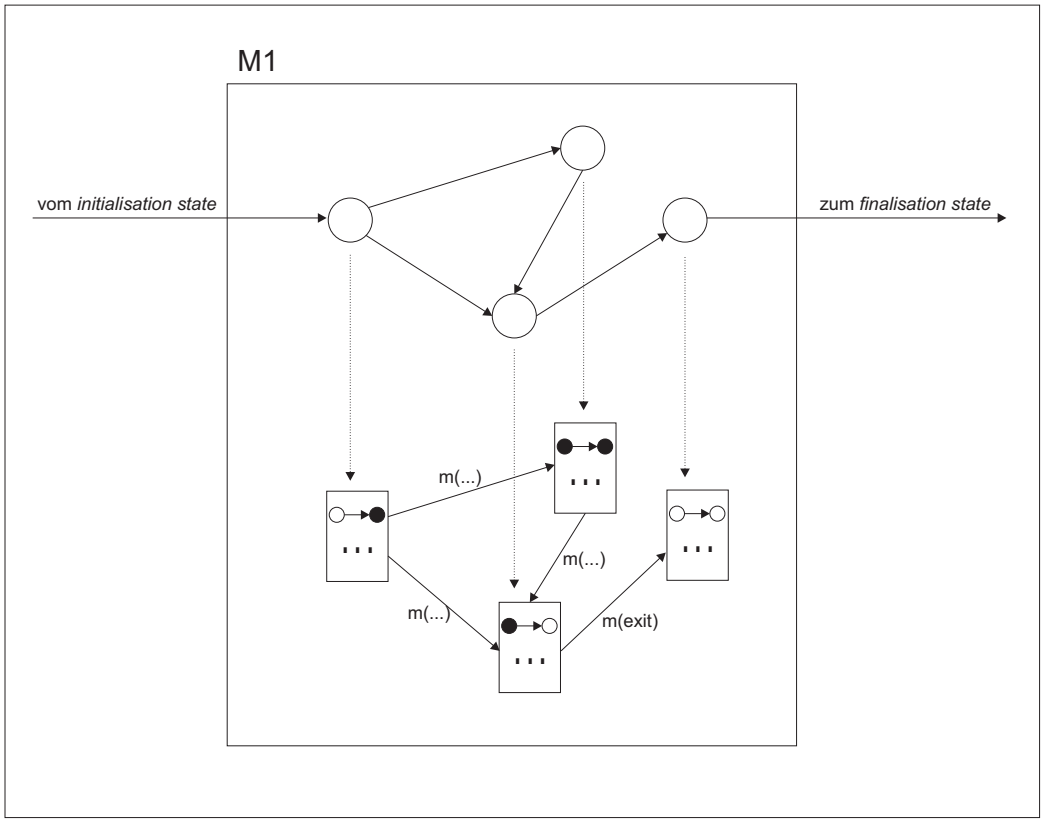

Abbildung 4: Die Abbildung zeigt ein Transformationsystem sowie zwei zusätzliche Kanten, die zum Betreten und Verlassen des Systems verwendet werden.

.

Zur Unterstützung der Komposition [5] von Komponenten bedarf es zusätzlicher Strukturen. Zum einen bedarf es Start- und Endzuständen, welche dazu benutzt

werden, das Modell zu betreten oder zu verlassen. Große-Rhode nennt sie *initialisation state* und *finalisation state*, da sie Zustände ausserhalb des Systems repräsentieren, die zu den Start- und Endzuständen innherhalb des Systems jeweils durch nur eine Kante verbunden sind.

Die Abbildung 4 zeigt ein einfaches Transformationssystem, welches um zwei zusätzliche Kanten erweitert wurde. Die erste beginnt ausserhalb des Systems im *initialisation state* und endet im Startzustand innerhalb des Systems. Die zweite Kante beginnt im Endzustand des Systems und endet im *finalisation state* ausserhalb des Systems. Mit Hilfe dieser Kanten können verschiedene Transformationssysteme verknüpft und so in eine zeitliche Abfolge gebracht werden (vgl. Abbildung 5)

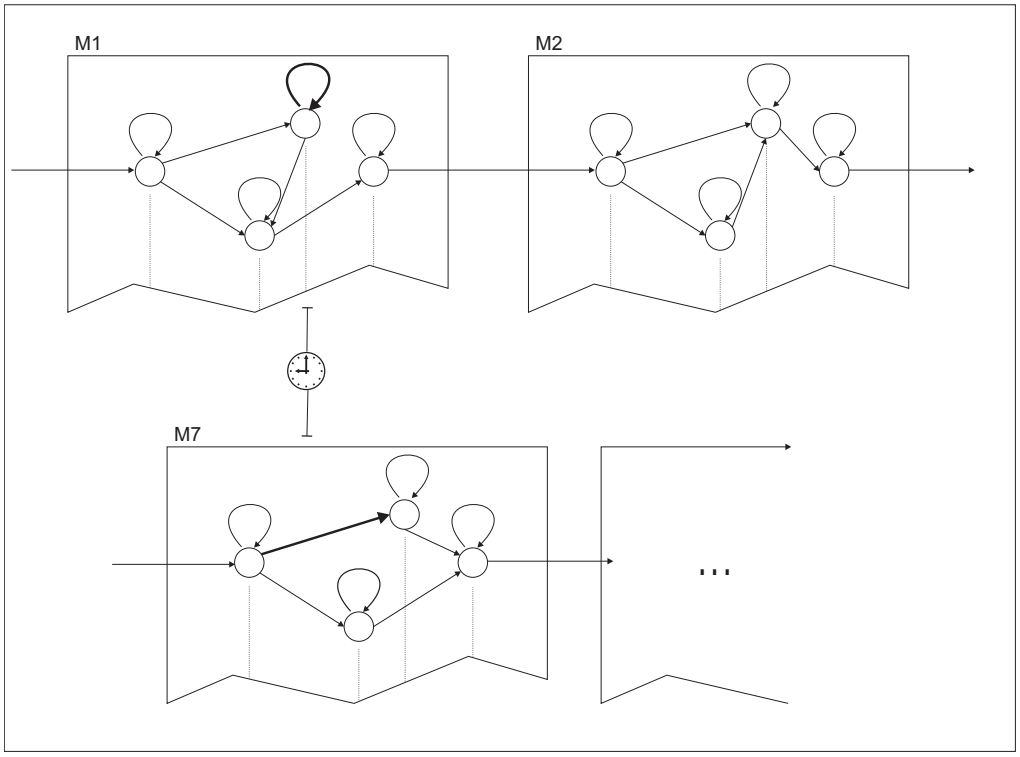

Abbildung 5: Die Abbildung zeigt einen Ausschnitt aus einem Netzwerk von Transformationsystemen.

Weiterhin erhalten alle Zustände *idle transitions* (etwa: Leerlauftransitionen), die in der parallelen Komposition zur Synchronisation der einzelnen Komponenten benötigt werden. Synchronisation mit Hilfe einer *idle transition* bedeutet, dass nur eine Komponente eine Aktion ausführt, während die anderen Komponenten im Leerlauf verharren.

Die Abbildung 5 zeigt einen Ausschnitt aus einem Netzwerk von Transformationssystemen. Die Kontrollzustände wurden hier um *idle transitions* in Form von Schleifen ergänzt. Zu einem bestimmten Zeitpunkt (symbolisiert durch die Zeitachse oberhalb und unterhalb der Uhr) wird im oberen Transformationssystem eine Schleife (fett dargestellt) durchlaufen, während im unteren Transformationsystem eine Transition ausgeführt wird.

Die Komposition von Transformationsystem als formale Modelle lokaler Systemkomponenten umfasst zwei Definitionsaufgaben: Zum einen die Verbindung der einzelnen Komponenten untereinander, zum anderen das globale Ergebnis dieser Komposition als Ganzes. Dazu muss ein Transformationssystem gegeben werden, das einen globalen Überblick über das System repräsentiert.

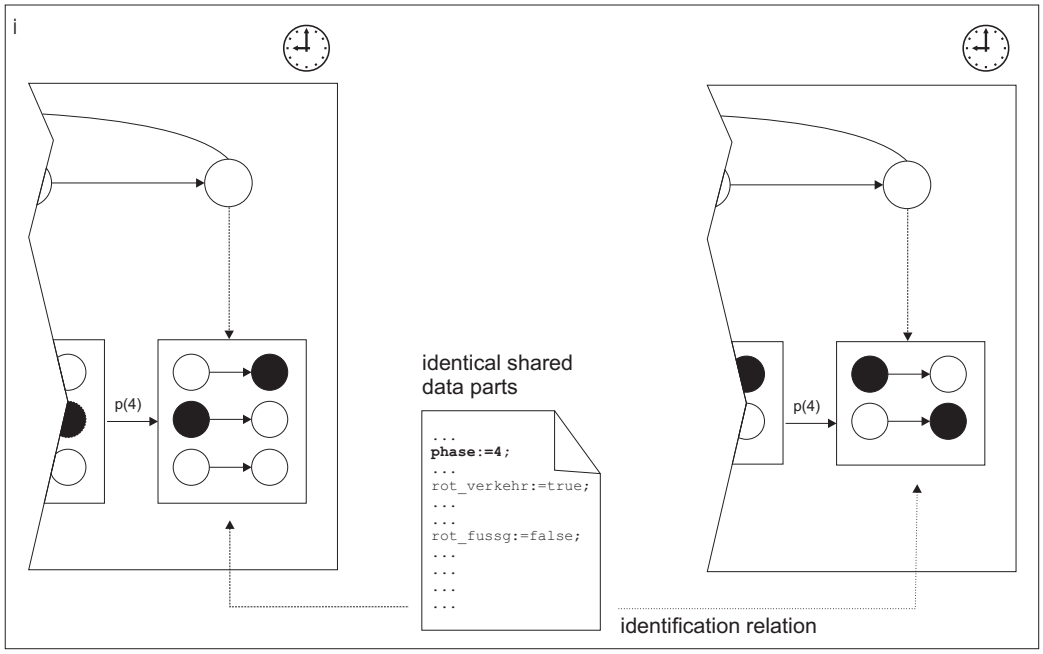

Abbildung 6: Die Abbildung zeigt einen Ausschnitt aus dem Transformationssystem der Ampelschaltung und einen zusätzlichen Ausschnitt aus dem Transformationssystem der entsprechenden Fußgängerampel, die beide auf den selben Datenbestand zugreifen.

Eine Verbindungsrelation für zwei Transformationssysteme wird in Form einer *identification relation* für ihre Struktur und einer *synchronisation relation* für ihr Verhalten definiert. Die *identification relation* besagt, welche Datenzustände und Aktionen von den beiden Systemen geteilt werden. Dies umfasst statische Datentypen, Variablen und gemeinsam ausgeführte Aktionen und Ereignissynchronisationen. Die *synchronisation relation* besagt, welche Kontrollzustände von beiden

Komponenten zur selben Zeit betreten werden können und so einen globalen Zustand formen. Dazu muss die Konsistenzbedingung gewährleistet werden, dass synchrone Kontrollzustände die selben Daten (*identical shared data parts* teilen. Die Synchronisation einzelner Schritte findet ihre Repräsentation in Form von Synchronisationsbeziehungen an den Transitionen.

Die Abbildung 6 zeigt einen Ausschnitt aus dem Transformationssystem der Ampelschaltung und einen zusätzlichen Ausschnitt aus dem Transformationssystem der entsprechenden Fußgängerampel. Beide Teilsysteme greifen letztendlich auf den selben Datenbestand (Mitte) zu, welcher durch den Methodenaufruf (`p(4)`) aktualisiert wird.

Die hier vorgestellte Komposition bedarf, was die zugrunde liegenden Modelle angeht, formaler Vorarbeit. Sollen verschiedene Modellsprachen in eine Kompositionsbeziehung gesetzt werden, muss für jede Sprache zunächst ein formaldeskriptiver Overhead entwickelt werden, also ein Transformationssystem, das auf der Semantik dieser Sprache aufsetzt und den Rahmen zur Vereinigung mit anderen Transformationsystemen bildet, die ihrerseits Sprachen beschreiben. Große-Rhode diskutiert diese formale Interpretation verschiedener Semantiken als Transformationssysteme für die Sprachen der Klassendiagramme, Zustandsmaschinen und Sequenzdiagramme in [4]. Ein Beispiel wurde bereits in Abbildung 2 gegeben.

5 Zusammenfassung und Ausblick

Ziel dieser Ausarbeitung war es, einen Überblick über die Integrationsproblematik im Zusammenhang mit heterogenen Softwaresepzifikationstechniken zu geben. Die Problematik ergab sich daraus, dass verschiedene Spekzifikationssprachen für spezifische Teilbereiche des Softwareentwicklungsprozesses Akzeptanz gefunden und sich durchgesetzt haben. Im RM-ODP wurde dieser multi-paradigmatische Ansatz detailliert ausgearbeitet, der sich z.B. in Form des Viewpoint-Modells manifestiert hat. Die verschiedenen Viewpoints wurden vorgestellt und und auf dieser Grundlage Anforderungen an ein Referenzmodell formuliert, welches mächtig genug ist, die verschiedenen Teilspezifikationen zu einem gemeinsamen Systemmodell zusammenzuführen. Das Referenzmodell sollte dazu keine weitere Sprache einführen, sondern die beteiligten Sprachen auf semantischer Ebene analysieren und zusammenführen. Diese Voraussetzung erfüllen Transformationssysteme, die auf formal-mathematischer Ebene semantische und syntaktische Repräsentationen der einzelnen Modellsprachen darstellen und zu einem System integrieren können.

Die Konsistenz des gemeinsamen Systemmodells ergibt sich aus der Beweisbarkeit des formalen Grundinventars der Transformationssysteme und der Tatsache, dass ein gemeinsames Spezifikationsmodell auf Grundlage verschiedener Spezifikationen entwickelt werden konnte und damit die Konsistenzanforderung erfüllt wurde.

Eine detaillierte formale Ausgestaltung der Transformationssysteme, die auf den Sprachen der Klassendiagramme, Zustandsmaschinen und Sequenzdiagramme aufsetzt, wurde in dieser Ausarbeitung außer Acht gelassen und könnte ebenso den Rahmen für weitere Auseinandersetzungen mit diesem Thema darstellen, wie die bereits vorhandenen Tools, die für Transformationssysteme eingesetzt werden können [7].

Zu Kritisieren bleibt an diese Stelle noch die sehr abstrakte und einseitige Auseinandersetzung mit dem Thema Integration seitens Große-Rhodes, dessen Priorität scheinbar vorranging in der formalen Ausgestaltung eines Transformationssystems mit mathematischen Mitteln lag. Wünschenswerter wäre es gewesen, auch weitere, anwendungsspezifischere Methoden zur Integration heterogener Modellierungskonzepte kennen zu lernen und dies insbesondere in weniger abstrakterer, sondern anwendungsbezogener Weise.

Literatur

[1] COMPUTERGESTÜTZTE INFORMATIONSSYSTEME (CIS), TU-BERLIN: *Viewpoints und Viewpoint-Languages*. `http://cis.cs.tu-berlin.de`.

[2] GROSSE-RHODE, M., S. JOHN und G. SCHROETER: *Transformation Systems for the Integration of Software Specifications*. In: KREOWSKE, H.-J. und P. KNIRSCH (Herausgeber): *Proceedings of APPLIGRAPH Workshop on Applied Graph Transformation (AGT'02), ETAPS 2002 Satellite Event, Grenoble, France*, Seiten 151–160, 2002.

[3] GROSSE-RHODE, MARTIN: *ODP - Ein Standard zur Beschreibung von offenen verteilten Anwendungen*, 1996. Seminartext.

[4] GROSSE-RHODE, MARTIN: *Integrating Semantics for Object-Oriented System Models*. In: OREJAS, SPIRAKIS, LEEUWEN (Herausgeber): *Proceedings International Colloquium on Automata, Languages and Programming (ICALP 2001)*, Seiten 40–60. Springer, 2001.

[5] GROSSE-RHODE, MARTIN: *Semantic Integration of Heterogeneous Formal Specifications via Transformation Systems*. Technischer Bericht, Fakultät IV, TU Berlin, 2001.

[6] GROUP, OBJECT MANAGEMENT: *UML® Resource Page*. `http://www.omg.org/technology/uml/index.htm`.

[7] PROGRAM-TRANSFORMATION.ORG: *Transformation Systems*. `http://www.program-transformation.org`.