**Gottfried Wilhelm**
**Leibniz Universität Hannover**
**Fakultät für Elektrotechnik und Informatik**
**Institut für Praktische Informatik**
**Fachgebiet Software Engineering**

# Integration von Techniken zur Spezifikation und Simulation reaktiver Systeme mit struktureller Dynamik

## Bachelorarbeit

im Studiengang Informatik

von

**Fabian Schmidt**

**Prüfer: Prof. Dr. Joel Greenyer**
**Zweitprüfer: Prof. Dr. Kurt Schneider**
**Betreuer: Prof. Dr. Joel Greenyer**

**Hannover, 29. Mai 2014**

## Zusammenfassung

Reaktive Systeme mit struktureller Dynamik sind aufgrund ihrer vielfältigen Interaktionen, intern und mit ihrer Umwelt, mittels herkömmlichen Modellierungstechniken nur schwierig in vollem Umfang zu modellieren. Aus diesem Grund wird der Versuch unternommen, szenariobasierte Modellierung und Graphtransformation miteinander zu integrieren, um dadurch ein umfangreicheres und detaillierteres Ergebnis in Bezug auf Modellierung und Simulation solcher Systeme zu erzielen.

## Abstract

Reactive systems with structural dynamics are difficult to model to the full extent due to the large amount of different interactions, internally and related to their environment, using common modelling techniques. This is why the approach was taken to integrate scenario based modellig and graph transformation in order to achieve a more comprehensive and detailed result by modeling and simulating this kind of systems.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

*Strukturelle Dynamik* ist ein Phänomen, das in unterschiedlichen Ausprägungen auftreten kann. Strukturelle Dynamik bezeichnet die dynamische Veränderung der Struktur eines Systems während seiner Benutzung. Mit Struktur sind in diesem Zusammenhang die Beziehungen von Objekten oder Teilsystemen innerhalb eines Systems gemeint. Sie kann virtueller oder physischer Natur sein, wobei mit virtueller Struktur die Beziehung zwischen Softwareobjekten und mit physischer Struktur die logischen und physikalischen Zusammenhänge maschineller Komponenten beschrieben werden.

Besonders mechatronische Systeme können über eine stark ausgeprägte strukturelle Dynamik verfügen. Hierbei überschneiden sich, durch die informationstechnische Steuerung von maschinellen Komponenten, virtuelle und physische Struktur in großem Maße.

Ein Beispiel für Systeme dieser Art ist ein Transportsystem, das selbständig arbeitende Transportroboter (Abb. 1.1) für den automatisierten Transport von Gütern verwendet.


Abbildung 1.1: Transportroboter [Fas12]

Verteilte reaktive Systeme mit struktureller Dynamik sind heutzutage immer häufiger anzutreffen. Sie werden in vielen unterschiedlichen Bereichen der Industrie eingesetzt. Einige Forschungsgruppen definieren solche Systeme, in denen sich informationstechni-

sche Prozesse und mechanische Prozesse gegenseitig auslösen, als *Cyber-Physical Systems* (*CPS*) [Lee08] [K+13]. Systeme dieser Art erfahren in der aktuellen Forschung große Aufmerksamkeit. Anwendungsbeispiele sind unter anderem:

- Vernetzung von Fahrzeugen untereinander bzw. mit der Verkehrsinfrastruktur im Hinblick auf selbstständig fahrende, sich unter einander organisierende PKW, besonders im Hinblick auf die strukturelle Dynamik solcher Systeme [K+13].
- Produktionssysteme und deren Vernetzung auch über Komponenten- / System- / Fabrik- und Firmengrenzen hinweg [K+13].

Betrachten wir ein System zur Materialversorgung von Montagearbeitsplätzen (Abb. 1.2): In diesem Umfeld werden Transportroboter eingesetzt, um benötigtes Material an Montageplätze zu transportieren.

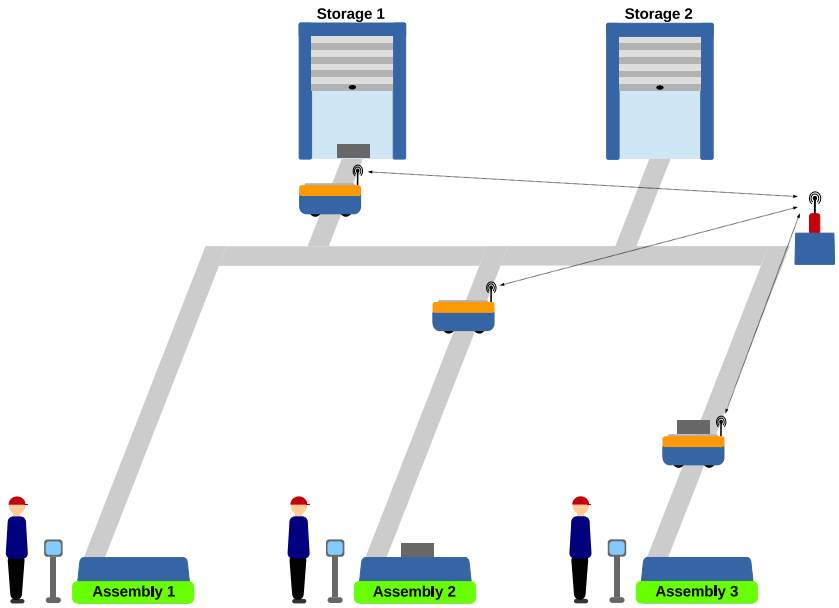


Abbildung 1.2: Schaubild Materialversorgungssystem

Häufig interagieren derartige Systeme mit ihrer Umwelt. Die Grenze zwischen System und Umwelt ist in diesem Fall dort zu ziehen, wo die Möglichkeit der Steuerung durch das System endet. Die Interaktion mit der Umwelt kann dabei auf unterschiedliche Weise erfolgen:

> Es kann eine gezielte Interaktion seitens der Benutzer stattfinden, wie z.B. die Eingabe über ein Benutzerinterface auf einem Computer. Dies stellt einen Umwelteinfluss dar, der einen Prozess im System auslöst und dadurch zu strukturellen Veränderungen im System führen kann.
>
> Auch Sensoren an Teilsystemen können einen Umwelteinfluss aufnehmen. Insbesondere in Produktionsanlagen finden sich häufig optische Sensoren, die der Kontrolle von Werkstücken dienen.

Reaktive Systeme mit struktureller Dynamik sind Systeme, die auf Umwelteinflüsse reagieren und ihre Struktur dynamisch während des Betriebs verändern. Mit wachsender Komplexität erfordert ihre Planung die Verwendung von aussagekräftigen Modellen zur Bestimmung des gewünschten Systemverhaltens.

Eine Technik zur Modellierung der Abläufe in derartigen Systemen stellt die *szenariobasierte Modellierung* dar [BTF+02]. Für diese Art der Modellierung wird das geplante Systemverhalten in Form von Szenarien modelliert. Szenarien bilden das System aus der Perspektive des Benutzers ab. Auf diese Art und Weise lässt sich die vorgesehene Reaktion des Systems auf zu erwartende Eingaben der Anwender planen. Über eine anschließende Simulation kann das geplante Verhalten des Modells überprüft werden. Szenarien werden häufig in Form von *MSDs* notiert [HM08]. Diese Erweiterung von UML-Sequenzdiagrammen ermöglicht die Modellierung dessen, was das System "machen soll", "machen muss" und "nicht machen darf". Szenarien, die in MSDs beschrieben sind, können mit Hilfe des *play-out* Algorithmus simuliert werden, um so das modellierte Systemverhalten auf Korrektheit zu überprüfen [HM03].

Strukturelle Dynamik lässt sich mit szenariobasierter Modellierung jedoch nur bedingt abbilden. Es können mittels Szenarien zwar kleine strukturelle Änderungen modelliert und simuliert werden. Bei komplexeren Systemen, und damit komplexerer struktureller Dynamik, stößt die szenariobasierte Modellierung allerdings an ihre Grenzen.

Um eine gute Spezifikation eines solchen Systems anzufertigen, ist es nötig, das Verhalten möglichst exakt planen zu können. Zu diesem Zweck ist es wichtig, auch die strukturelle Dynamik abbilden zu können. Andernfalls ist es schwierig, präzise Voraussagen über die Reaktion des Systems auf die Vielfalt an Umwelteinflüssen zu treffen. Besonders das Verhalten selbstständig agierender Komponenten, wie die Transportroboter aus obigem Beispiel (Abb. 1.2), ist genau zu betrachten, da die Roboter beim Transportieren schwerer Lasten auch ein Sicherheitsrisiko für die Mitarbeiter darstellen können.

Eine Technik, um veränderliche Strukturen von Systemen grafisch darzustellen, ist Graphtransformation. Hierbei werden Systemstrukturen als Graphen abgebildet und über sogenannte Graphtransformationsregeln werden strukturelle Veränderungen an dem modellierten Graphen vorgenommen.

Die Integration dieser beiden oben genannten Modellierungstechniken soll dazu führen, dass sowohl Nachrichtenkommunikation zwischen Systemobjekten als auch strukturelle Dynamik in Kombination miteinander modelliert und simuliert werden können, um so eine noch detailliertere Verifikation des geplanten Systems zu ermöglichen.

## 1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, szenariobasierte Modellierung und Graphtransformation zu integrieren, um ein noch aussagekräftigeres Simulationsergebnis erzielen zu können. Diese Integration soll über die Kopplung von zwei Modellierungtools realisiert werden.

Als Grundlage für die Toolkopplung soll zuerst die innere Funktionsweise beider Tools analysiert werden, um mögliche Ansatzpunkte für die spätere Integration von szenariobasierter Modellierung und Graphtransformation zu bestimmen.

Die danach folgende Toolkopplung soll es möglich machen, das modellierte Systemverhalten in Bezug auf die Kommunikation zwischen den Systemobjekten (szenariobasiert) und in Bezug auf strukturelle Dynamik (Graphtransformation) zu simulieren und auf diesem Wege ein umfangreicheres und aussagekräftigeres Simulationsergebnis zu erhalten.

## 1.3 Struktur der Arbeit

Nach der Kapitel 1 zugrunde liegenden Einleitung werden in Kapitel 2 die relevanten Grundlagen für das Verständnis der Problemstellung und die im Rahmen dieser Arbeit verwendeten Tools dargestellt. In erster Linie werden hier die zu integrierenden Techniken detailliert erläutert und der Zusammenhang dieser Techniken mit den zu koppelnden Tools herausgestellt.

Kapitel 3 setzt sich mit dem dieser Arbeit zu Grunde liegenden Beispiel auseinander. Die Komponenten des modellierten Beispiels werden erläutert und ihre Zusammenarbeit als Gesamtsystem beschrieben. Ferner werden die Anforderungen an die Integration der beiden Techniken analysiert und formuliert.

Kapitel 4 erörtert das Lösungskonzept bezüglich der Integration beider Modellierungstechniken und der damit verbundenen Toolkopplung. Hier werden die theoretische und technische Umsetzung der Integration vorgestellt und definiert.

In Kapitel 5 wird die technische Umsetzung der vorgesehenen Toolkopplung beschrieben.

Kapitel 6 nimmt Bezug auf mit dieser Arbeit verwandte Arbeiten und gewährt einen Einblick in andere Ansätze der Integration von zwei Modellierungstechniken oder alternative Verwendungsformen der hier angewandten Techniken.

In Kapitel 7 wird eine Zusammenfassung der Arbeit inklusive eines Fazits formuliert. Das Kapitel enthält darüber hinaus einen Ausblick.

# 2 Grundlagen

In den folgenden Abschnitten werden grundlegende Konzepte und Methoden der zu koppelnden Modellierungstechniken erläutert, um im späteren Verlauf der Arbeit die Herangehensweise an die Toolkopplung detailliert beschreiben zu können.

## 2.1 Beispiel Einführung

Das für diese Arbeit erstellte Beispiel beschreibt ein Materialversorgungssystem für Montageplätze. Das Beispielsystem ist eine Vereinfachung eines vollautomatischen Materialversorgungssystems, das Montagearbeitsplätze mit komplexen Baugruppen, einzelnen Maschinenteilen oder benötigten Verbrauchsmaterialien versorgt.

Systeme dieser Art werden heute in vielen Bereichen der industriellen Fertigung verwendet. Unter anderem werden sie in der Automobilindustrie eingesetzt, um die Montage individuell ausgestatteter Automobile teilweise zu automatisieren.

Abbildung 2.1 zeigt die schematische Darstellung des der Arbeit zugrunde liegenden Beispielsystems. Mitarbeiter können über Terminals an ihren Arbeitsplätzen für die Montage von Gütern benötigte Bauteile bestellen. Eine zentrale Steuerung (Transportsystem) erzeugt einen Transportauftrag, der von einem Transportroboter bearbeitet wird. Transportroboter holen die benötigten Bauteile aus Lagern ab und stellen diese den Mitarbeitern an ihren Arbeitsplätzen zur Verfügung.

Unter Einsatz dieser Systeme ist es möglich, die exakt benötigten Baugruppen, Maschinen- und Kleinteile rechtzeitig an den jeweiligen Montageplätzen zur Verfügung zu stellen. Solche Materialversorgungssysteme werden häufig vollautomatisch gesteuert, um eine gezielte zeitgenaue Materialversorgung der Montagearbeitsplätze sicher zu stellen.

Da der Fokus in dieser Arbeit nicht auf dem Zeitmanagement der Materialversorgung liegt, haben wir uns für eine Vereinfachung des Beispielsystems entschieden:

Der Schwerpunkt der Betrachtung liegt in dem hier eingeführten Beispiel auf der Umsetzung des Materialtransports von dem Zeitpunkt an, an dem ein Bauteil benötigt wird, bis zu dem Zeitpunkt der Lieferung an den jeweiligen Arbeitsplatz. Für das verwendete Beispiel gehen wir davon aus, dass ein Mitarbeiter jeweils einen Gegenstand für seinen Montageplatz anfordern kann, der ihm als nächstes geliefert wird.

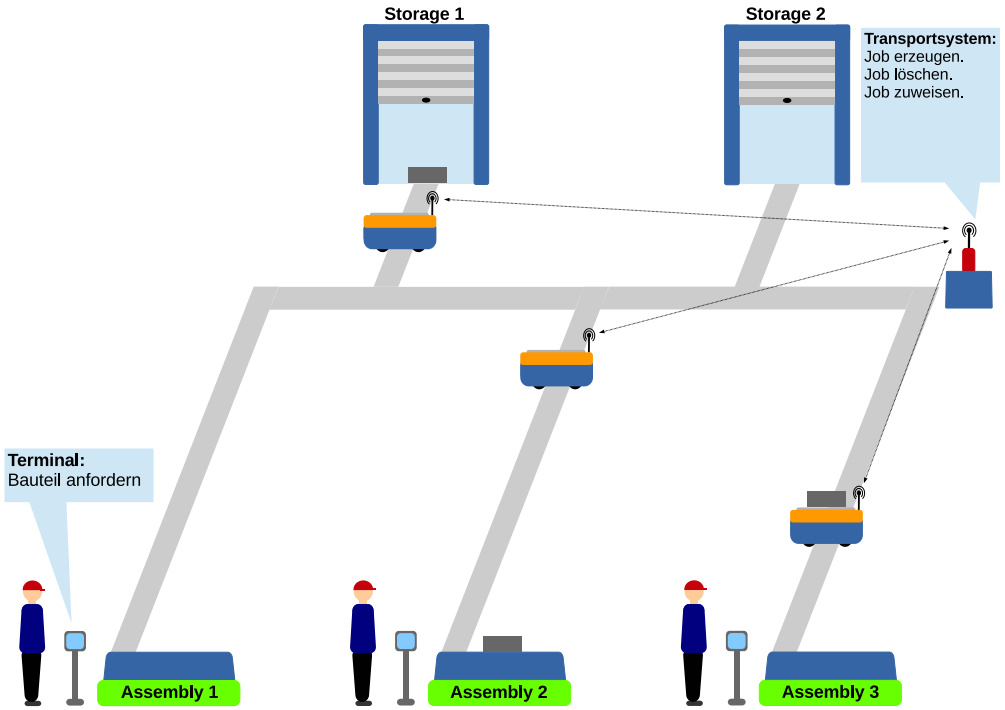


Abbildung 2.1: Schema Materialversorgungssystem

## 2.2 Reaktive Systeme mit struktureller Dynamik

Reaktive Systeme mit struktureller Dynamik finden sich seit einigen Jahren besonders im Bereich der mechatronischen Systeme immer häufiger. Beispiele hierfür sind unter anderem moderne Produktionsanlagen, in denen maschinelle Komponenten einen Großteil der mechanischen Arbeit verrichten.

Mit struktureller Dynamik bezeichnen wir eine Eigenschaft, die die Veränderungen der sowohl *virtuellen* als auch *physikalischen* Struktur eines Systems beschreibt.

Die virtuelle Struktur eines Systems ist in diesem Zusammenhang als die Struktur der nicht-materiellen Komponenten zu verstehen. Nicht-materielle Komponenten sind beispielsweise Softwareobjekte, die miteinander interagieren oder in speziellen Beziehungen zu einander stehen.

Unter physikalischer Struktur verstehen wir die Struktur des Systems in der physikalischen Welt. Maschinelle Systemkomponenten sind als Teile der physikalischen Struktur zu bezeichnen. Wenn ein Transportroboter ein Lager anfährt, dort ein Bauteil aufnimmt und es an einer Bearbeitungsstation abliefert, ändert sich die physikalische Struktur des Systems gleich mehrfach: Der Roboter wechselt laufend seinen Aufenthaltsort und auch das transportierte Bauteil bildet unterschiedliche neue Beziehungen aus.

Virtuelle und physikalische Struktur treten häufig parallel auf und überschneiden sich in

vielen Systembereichen. Wird beispielsweise einem Transportroboter ein Job zugewiesen, wird eine Beziehung zwischen nicht-materieller und materieller Komponente erzeugt. Dies ist möglich, da der Transportroboter mit einer eigenen elektronischen Steuerung ausgestattet ist, die auch die Kommunikation mit dem Transportsystem gewährleistet.

Abbildung 2.2 zeigt den Prozess des Bauteiltransports als Graphtransformation (Abschnitt 2.5) für die strukturelle Dynamik des Beispiels.

**1** Der Roboter hat einen Job zugewiesen bekommen, der ein Bauteil betrifft, das einem Lagerbereich zugeordnet ist.

**2** Der Roboter ist zu dem betreffenden Lagerbereich gefahren und hat das benötigte Bauteil aufgenommen.

**3** Der Roboter ist zu dem Arbeitsplatz gefahren, an dem das Bauteil bestellt wurde.

**4** Der Roboter hat das Bauteil an dem betreffenden Arbeitsplatz abgeliefert.

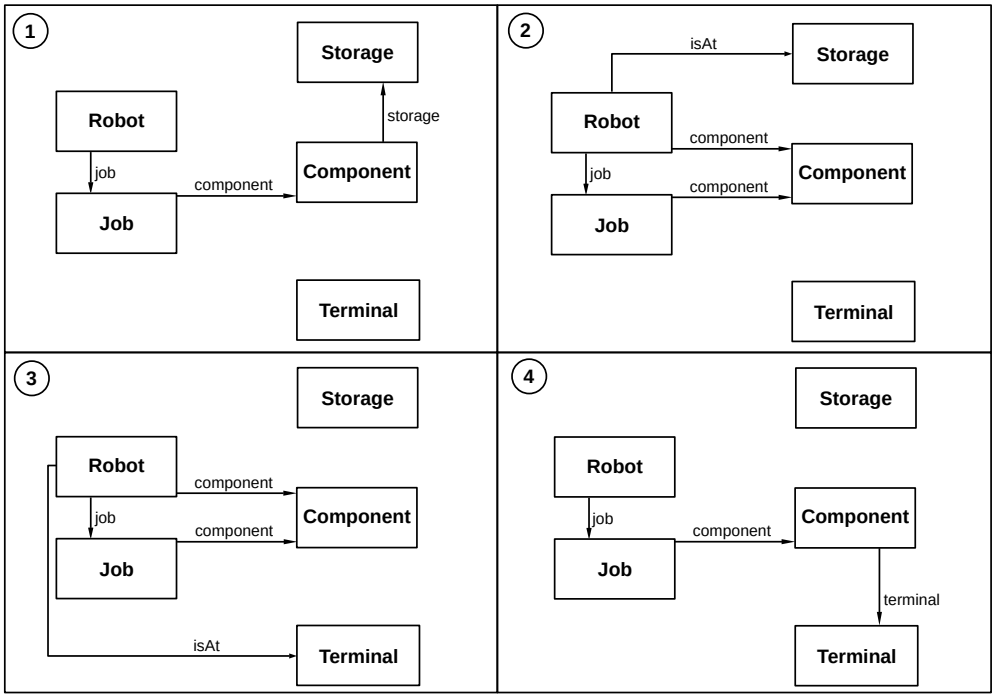


Abbildung 2.2: Beispiel für strukturelle Dynamik

Reaktive Systeme zeichnen sich durch die Fähigkeit aus, auf Ereignisse in ihrer Umwelt zu reagieren. Die Umwelt wird vom System an der Systemgrenze, an der die Möglichkeit der Steuerung durch das System endet, abgegrenzt. Wenn ein Impuls durch ein Objekt der Umwelt an das System auftritt, spricht man von einem Umwelteinfluss.

Umwelteinflüsse können in unterschiedlicher Weise auftreten:

Sie können unter anderem Eingaben an das System durch einen Benutzer dieses Systems sein. Wenn, bezogen auf das Beispiel, ein Mitarbeiter an einem Terminal ein Bauteil bestellt, stellt dies einen Umwelteinfluss dar. Außerdem sind Umwelteinflüsse beispielsweise durch Sensoren an Systemkomponenten gegeben. Wenn ein selbstständig fahrender Transportroboter auf ein Hindernis trifft, setzen die Sensoren des Roboters einen Prozess in Gang, der den Roboter anhält und eine Warnmeldung an die zentrale Steuerung sendet. Abbildung 2.3 zeigt zwei Beispiele für Umwelteinflüsse.

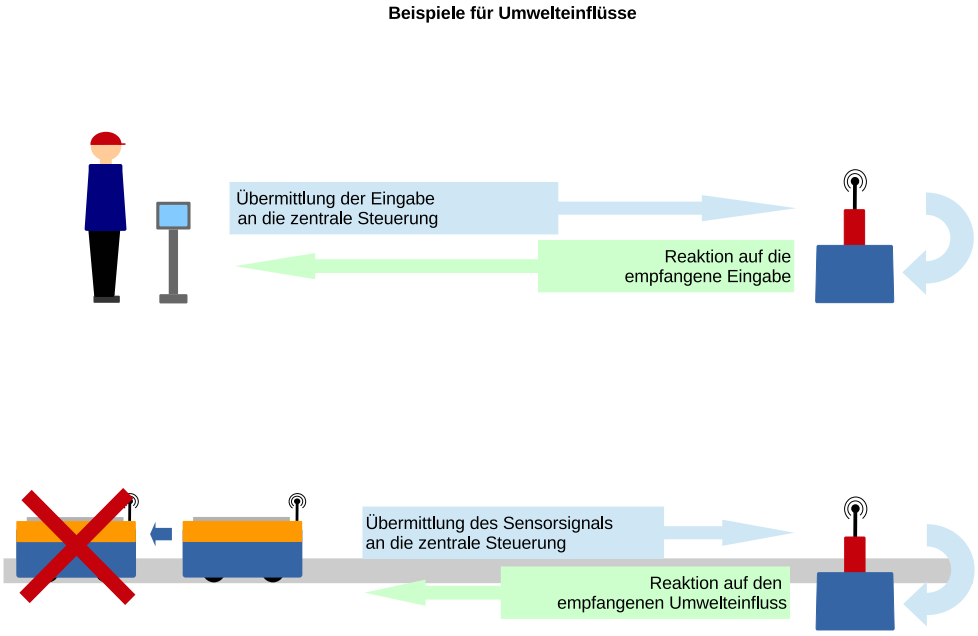


Abbildung 2.3: Beispiele für Umwelteinflüsse

## 2.3 Modellgetriebene Softwareentwicklung

*Modellgetriebene Softwareentwicklung*, bezeichnet als Model Driven Software Developement (MDSD), ist eine Methode zur Planung und Entwicklung von Software [SVEH07].

Die Verwendung von Modellen ist schon seit langem gängige Praxis in der Softwareentwicklung. Besonders seit der Definition der Unified Modeling Language (UML) finden Modelle eine immer größere Verbreitung in der Softwareentwicklung. Hierbei ist allerdings festzustellen, dass die Verwendung von UML-Modellen oftmals lediglich eine dokumentierende Funktion hat. Die Verbindung zwischen UML-Modell und der entworfenen Software ist gedanklicher Natur; daher spricht man bei der ausschließlichen Verwendung von UML-Modellen von modellbasierter Softwareentwicklung [SVEH07].

Die modellbasierte Softwareentwicklung birgt dabei ein Risiko. Besonders in der Anfangsphase der Softwareentwicklung verändert sich die geplante Software zum Teil stark. Da das UML-Modell eine die Softwareentwicklung begleitende Dokumentation ist, muss sie sorgfältig gepflegt werden, um Diskrepanzen zwischen geplanter Software und dem UML-Modell unbedingt zu vermeiden.

Bei der modellgetriebenen Softwareentwicklung ist das Modell nicht nur Teil der begleitenden Dokumentation. Das Modell nimmt in diesem Fall eine zentrale, treibende Position im Softwareentwicklungsprozess ein.

Aus den Modellen wird direkt Programmcode generiert, was dazu führt, dass der generierte Programmcode immer ein direktes Abbild des Modells ist. Das Modell ist dabei die "treibende" Komponente des Softwareentwicklungsprozesses.

### 2.3.1 Meta Object Facility

Die *Meta Object Facility (MOF)* ist ein von der Object Management Group (OMG) definiertes Metametamodell [SVEH07]. Die Unified Modeling Language (UML) stellt das hierzu gehörende Metamodell dar. Die OMG verwendet hierfür vier Metaebenen, die von 0 (Instanzen) bis 3 (Metamatamodell) durchnummeriert sind [SVEH07]. Abbildung 2.4 zeigt die schematische Struktur der vier Metaebenen, die in der MOF definiert sind.

Die vier Metaebenen der MOF definieren die Zusammenhänge zwischen konkreten Instanzen (M0) und dem das Modell definierenden Meta-Metamodell (M3). Eine Ebene ist jeweils die Instanz von der ihr übergeordneten Ebene, wobei sie gleichzeitig die unter ihr liegende Ebene definiert. Als Ausnahme gilt die Ebene M3, die sich selbst definiert und instanziert. Sie bildet somit die Basis der vier beschriebenen Metaebenen. Die vier Metaebenen sind im Detail wie folgt definiert (Abb. 2.4):

**M3** ist die Ebene der Meta-Metamodelle. Dies ist die Ebene, auf der die MOF in ihrer Struktur definiert ist. Sie beschreibt und instanziert sich - wie bereits beschrieben - selbst, um eine fortlaufende Metaisierung einzudämmen.

**M2** ist die Ebene der Metamodelle. Sie definieren die Struktur der Modelle der Ebene M1. Auf dieser Ebene ist beispielsweise die Struktur der UML definiert, die auch eine Entwicklung der OMG ist. Auf dieser Ebene wird die Struktur der Elemente festgelegt, die auf Ebene M1 instanziert werden.

**M1** ist die Ebene der Modelle. Auf dieser Ebene befinden sich z.B. UML-Modelle, die z.B. ein Softwaresystem beschreiben können. Auf dieser Ebene werden die Ausprägungen des Systems in Bezug auf die Implementierung modelliert und damit definiert.

**M0** ist die Ebene der Instanzen. Auf dieser Ebene befinden sich die konkret ausgeprägten Daten, die in der Ebene M1 modelliert sind.

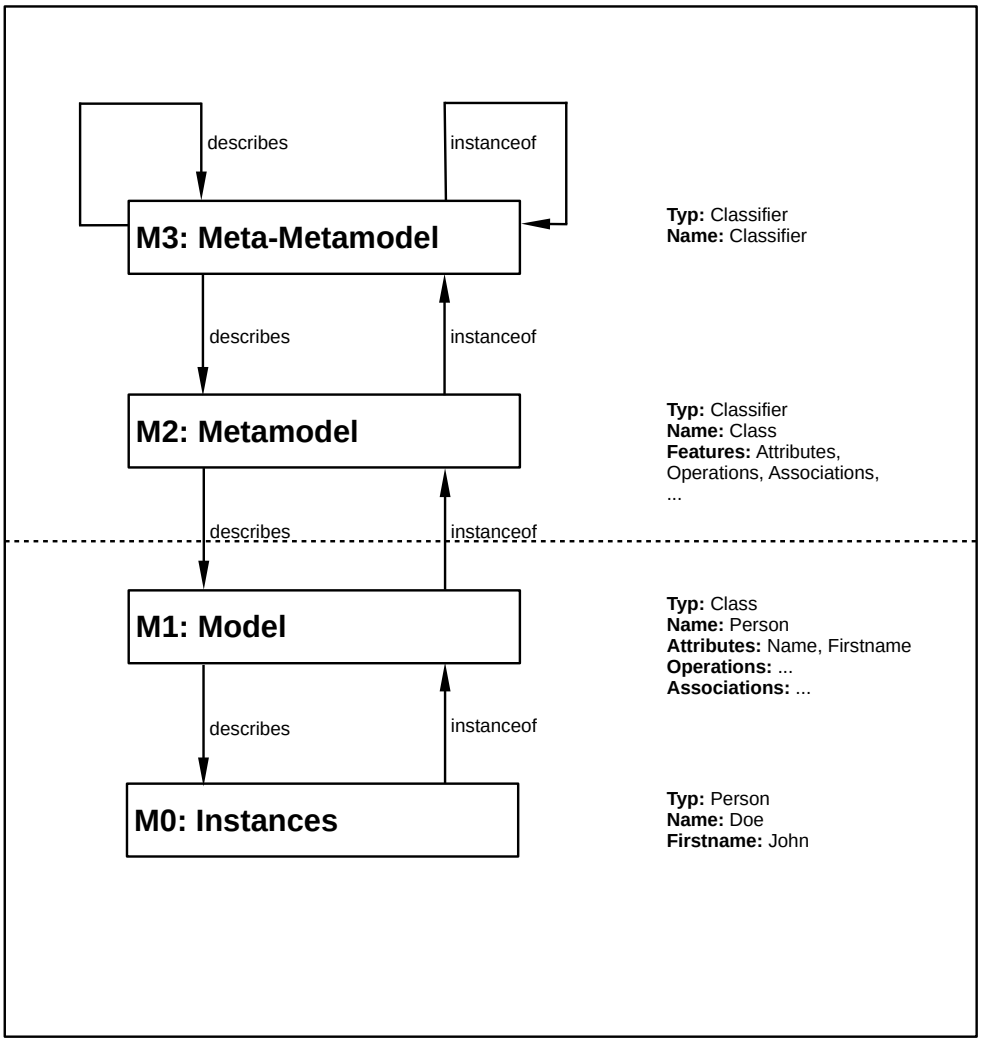


Abbildung 2.4: Metaebenen der MOF in Anlehnung an [SVEH07]

### 2.3.2 Eclipse Modeling Framework und Ecore

Das *Eclipse Modeling Framework (EMF)* ist ein quelloffenes Modellierungs-Framework für die integrierte Entwicklungsumgebung Eclipse [SBPM09]. Das zentrale Modell des EMF ist das sog. *Ecore-Modell*. Dieses Domänenmodell lässt sich in Form eines Klassendiagramms beschreiben.

Auf Basis dieser Ecore-Modelle lassen sich mittels des EMF Instanzen erzeugen. Diese Instanzen könne mit einem in EMF enthaltenen Editor abgefragt und verändert werden. Die dynamischen Instanzen werden im sog. *xmi*-Format *(XML Metadata Interchange)*

serialisiert, um für Persistenz in den Modellen zu sorgen [SBPM09].

Das xmi-Format ist ein von der OMG entwickeltes Austauschformat, das den Austausch von Daten zwischen Modellen, die auf dem Meta-Metamodell der MOF (Abschn. 2.4) basieren, erlaubt. Dies ermöglicht den Datenaustausch zwischen UML- und Ecore-Modellen, da beide Metamodelle von UML und Ecore auf der MOF basieren.

Ecore-Modelle nehmen in beiden Tools, die im Rahmen dieser Arbeit gekoppelt werden sollen, eine zentrale Rolle ein.

## 2.4 Szenariobasierte Modellierung

*Szenariobasierte Modellierung* ist nach BAI et al. eine Technik zur Spezifikation von Prozessen, die Teil eines Systems sind [BTF+02]. Die Perspektive auf das System bei dieser Art der Modellierung ist häufig die Sicht des Benutzers des Systems. Es werden Szenarien beschrieben, in denen der Benutzer eine Funktion des Systems aufruft, die damit enden, dass der Benutzer das gewünschte Ergebnis erhält.

In Bezug auf das hier verwendete Beispiel ist ein Szenario die Bestellung eines Bauteils durch einen Mitarbeiter am Terminal des Transportsystems. Dieses Szenario beginnt mit der Eingabe der Bestellung und endet mit der Lieferung des Bauteils durch einen Transportroboter. Auf diese Weise lassen sich bestimmte Anforderungen an das System anschaulich modellieren und auf einfache Weise mit einem definierten Ergebnis des Szenarios verknüpfen.

Die systeminternen Prozesse sind jedoch häufig komplexer, als eine einfache Anfrage an das System mit einem Ergebnis zu verknüpfen. Szenarien können daher in einzelne Teilszenarien zerlegt werden. Bezogen auf das Beispiel könnte das Erzeugen eines Transportjobs ein Unterszenario sein, das als Prozess eigenständig definiert ist und von unterschiedlichen anderen Szenarien als Teilszenario verwendet wird.

Scenariobasierte Modelle können sowohl textuell als auch grafisch beschrieben werden. Die grafischen Notationen ermöglichen eine intuitive Herangehensweise an die Modellierung von Systemprozessen. Unter anderem sind Use Case Diagramme eine häufig verwendete Methode zur Beschreibung von Szenarien, ebenso können modale Sequenzdiagramme hierfür verwendet werden.

### 2.4.1 Modale Sequenzdiagramme - MSDs

*Modale Sequenzdiagramme* (*Modal Sequence Diagram - MSD*) werden von HAREL und MOAZ als eine erweiterte Form von UML-Sequenzdiagrammen und *Life Sequence Charts* (*LSC*) definiert [HM03] [HM08]. Der Fokus von MSDs liegt auf der Betrachtung der Nachrichtenkommunikation zwischen Systemkomponenten zur Beschreibung von Szenarien. Die Nachrichten sind in diesem Fall Methodenaufrufe, die einen bestimmten Prozess oder eine Methode beim Empfänger der Nachricht starten.

MSDs sind eine Erweiterung von LSCs. LSCs wiederum sind eine Erweiterung von *Message Sequence Charts (MSC)* [ITU96].

MSCs stellen eine Methode dar, Szenarios und das mit ihnen verbundene Zusammenwirken von Objekten und Prozessen innerhalb eines Systems grafisch oder textuell abzubilden. MSC ist ein Standard der International Telecommunication Union (ITU) und wurde 1996 in einem technischen Report der ITU empfohlen [ITU96]. Verwendet man UML als Modellierungssprache, erfüllen Sequenzdiagramme die Funktion von MSCs.

LSCs sind eine Erweiterung von MSCs. Die Notation wurde dahingehend erweitert, dass man den Elementen der Sequenzdiagramme ein weiteres Merkmal hinzugefügt hat. Dieses Merkmal wird nach DAMM und HAREL als *Temperatur* bezeichnet und hat die Ausprägungen *hot (heiß)* oder *cold (kalt)* [DH01]. Hot beschriebt in diesem Fall, dass eine Aktion während des Durchlaufs eines Szenarios ausgeführt werden muss. Eine kalte Nachricht kann im Ablauf eines Szenarios auftreten, muss dies aber nicht. Man hat demnach die Möglichkeit, die Elemente einer LSC in notwendige und optionale Prozessschritte zu unterteilen.

MSDs als Erweiterung von LSCs verfügen darüber hinaus über eine weitere Kategorisierung von Nachrichten - den *Execution Mode* [HKM07]. Der Execution Mode kann entweder *Execute* oder *Monitor* sein. Mittels der Verwendung von Temperatur und Execution Mode haben Softwareentwickler die Möglichkeit, in Sequenzdiagrammen zu modellieren, welche Prozesse in Szenarien ablaufen müssen, können oder nicht ablaufen dürfen. Ohne diese Erweiterungen wäre dies deutlich schwieriger zu bewältigen.

Eine Spezifikation, die mit MSDs modelliert ist, besteht für gewöhnlich aus mehreren MSDs, in denen vorgesehene Abläufe des modellierten Systems dargestellt sind. Diese Abläufe werden normalerweise durch einen Umwelteinfluss gestartet und können sich über mehrere MSDs erstrecken. MSDs bestehen aus Objekten, die durch Lifelines repräsentiert, und Nachrichten, die als Pfeile zwischen diesen Lifelines dargestellt werden, mit der Notation entsprechend der obigen Beschreibung.

Nachrichten haben immer ein sendendes und ein empfangendes Objekt, wobei am empfangenden Objekt eine zur Nachricht gleichnamige Methode durch das Empfangen der Nachricht gestartet wird. Das Senden und Empfangen der Nachricht wird als *Messageevent* bezeichnet [BGP13]. Wenn ein Messageevent im System auftritt, das in seiner Struktur bezüglich Nachricht und empfangenden und sendenden Objekten der ersten Nachricht in einem MSD entspricht, wird das MSD zu einem *aktiven MSD*. Solange weitere Messageevents eintreten, die den Nachrichten des MSDs entsprechen, wird dieser Prozess weiter fortgeführt. Dieser Fortschritt wird durch den *Cut* festgehalten, der für alle Lifelines den aktuellen Fortschritt in Bezug auf Messageevents und ihre korrespondierenden Nachrichten speichert [BGP13]. Erreicht der Cut das Ende eines MSD, wird dieses beendet. Der Cut stellt demnach eine Fortschrittsmarkierung für die einzelnen Lifelines dar.

Befindet sich der Cut auf den die Nachricht betreffenden Lifelines vor den jeweiligen Message Events, ist die Nachricht *enabled* (freigegeben) [BGP13]. Der Cut verfügt an diesem Punkt über die gleichen Eigenschaften bezüglich Temperatur und Execution

Mode wie die entsprechende Nachricht. Da gleichartige Nachrichten in unterschiedlichen MSDs auftreten können, ist es möglich, dass Konfliktsituationen entstehen. Wenn zum Beispiel Message Events auftreten, die zu einer Nachricht passen, die aktuell nicht enabled (freigegeben) ist, kommt es zu einer *Violation (Verletzung)* [BGP13]. Es gibt unterschiedliche Typen von Violations. Ist der Cut hot (heiß), handelt es sich um eine *Safety-Violation*. Ist der Cut cold (kalt), spricht man von einer *Cold-Violation*. Wenn der Cut executed ist, muss das Szenario weitergeführt werden. Sollte dies nicht passieren, würde eine *Liveness-Violation* auftreten [BGP13].

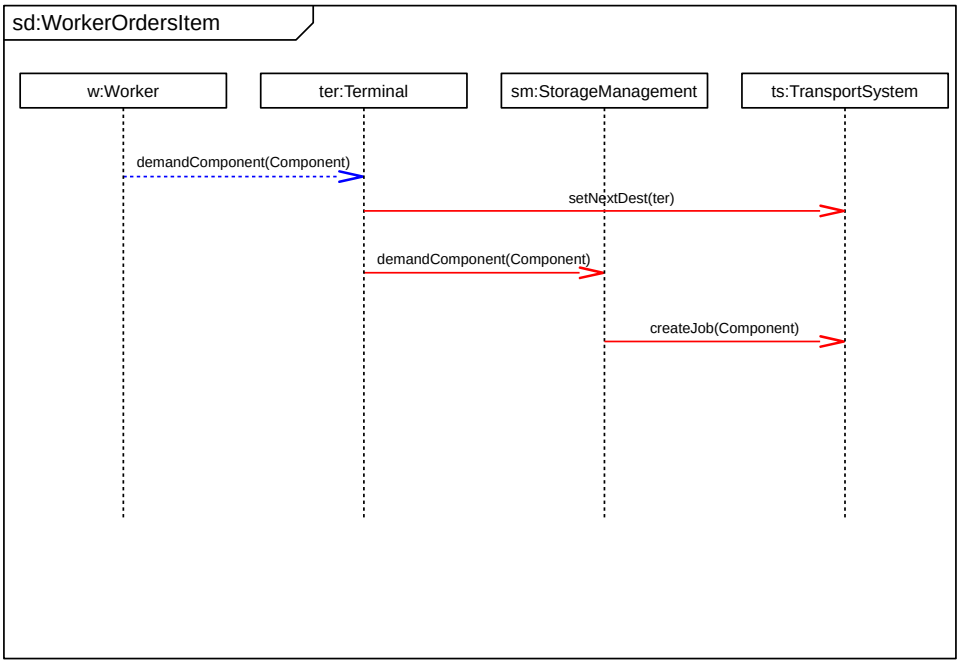


Abbildung 2.5: MSD WorkerOrdersItem mit modalen Aspekten

In Abbildung 2.5 ist das Sequenzdiagramm "WorkerOrdersItem" aus dem hier verwendeten, modellierten Beispiel dargestellt. Es stellt den Prozess von der Eingabe der Bestellung des Mitarbeiters am Terminal bis zur Erzeugung des Jobs dar. Beide Ausprägungen von Execution Mode und Temperatur sind in diesem MSD vertreten. Offensichtlich sind einige Nachrichten mit dem Parameter "Component" versehen. Nachrichten innerhalb MSDs können parametriert sein. Diese Parameter können Standartdatentypen entsprechen oder auch beispielsweise Objekte eines Modells sein, wie es in diesem Fall Component ist. Das im Rahmen dieser Arbeit verwendete ScenarioTools erlaubt einen Parameter je Nachricht.

### 2.4.2 ScenarioTools

*ScenarioTools* ist ein Softwaretool entwickelt von der DEEPSE Group des Politecnico di Milano und der Fachgruppe Softwaretechnik der Universität Paderborn [Gre14]. Es ist als Entwicklungstool für die Eclipse Entwicklungsumgebung entwickelt worden.

ScenarioTools ermöglicht die Modellierung von softwareintensiven Systemen mittels MSDs und eine anschließende Simulation dieser Modelle mit Hilfe einer erweiterten Version des *play-out* Algorithmus [HM03].

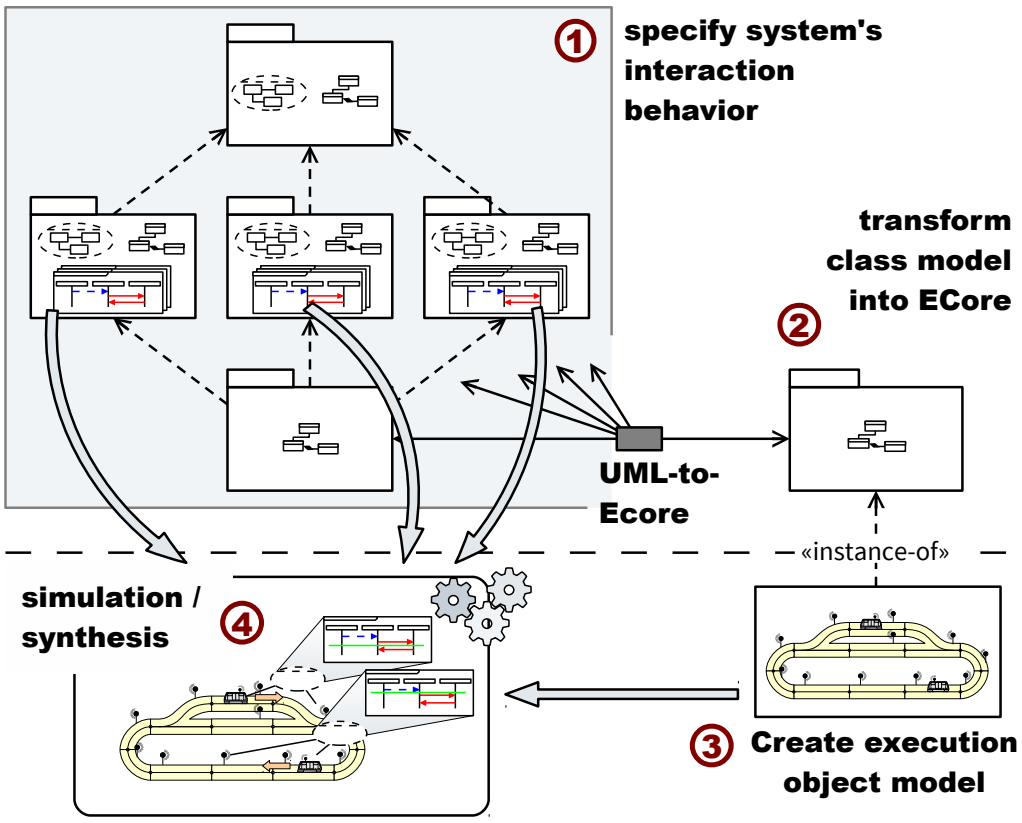


Abbildung 2.6: ScenarioTools Überblick, entnommen von [Gre14]

Abbildung 2.6 zeigt schematisch die Abläufe in der Anwendung von ScenarioTools von der Erstellung des UML-Modells bis zur Durchführung der Simulation.

**(1)** Zu Beginn der Modellierung wird das geplante System in UML modelliert. Hierzu wird ein Klassendiagramm des Systems erstellt, in dem die Klassen inklusive Attributen, Operationen und Beziehungen untereinander definiert werden. Ferner werden Kollaborationsdiagramme und MSDs/Sequenzdiagramme erzeugt, in denen die Interaktion der Objekte definiert wird.

**(2)** Im nächsten Schritt wird aus dem UML-Modell mittels ScenarioTools ein Ecore-

Modell (Abschn. 2.3.2) generiert. Im Zuge dieses Prozesses erzeugt ScenarioTools ein UML-to-Ecore-Mapping. Mittels dieses Mappings wird eine logische Verbindung zwischen den Elementen des UML-Modells und denen des Ecore-Modells erzeugt, die im späteren Verlauf der Simulation benötigt wird. Bezogen auf die MOF (Abb. 2.4) ist sowohl das UML- als auch das Ecore-Modell der Ebene M1 zuzuordnen.

**(3)** Anschließend wird auf Basis des Ecore-Modells ein dynamisches Objektmodell erzeugt, das eine dynamische Instanz des Ecore-Modells repräsentiert. Dieses Instanzmodell ist das Objektsystem, das für die Simulation verwendet wird. Bezogen auf die MOF (Abb. 2.4) ist das Instanzmodell der Ebene M0 zuzuordnen.

**(4)** Im letzten Schritt wird das modellierte System unter Verwendung des play-out-Algorithmus simuliert. Die Simulation verwendet hierfür die MSDs des UML-Modells und die dynamische Objektinstanz des Ecore-Modells. An diesem Punkt wird das in Schritt 2 verwendete UML-to-Ecore-Mapping verwendet und dient als Referenz zwischen UML- und Ecore-Modell.

Mit den oben beschriebenen Methoden ist es möglich, eine UML-Spezifikation (Klassendiagramm, Kolaborationsdiagramme, MSDs) unter Verwendung einer dynamischen Objektinstanz (Instanz des Ecore-Modells), ohne vorherige Generierung von Programmcode, zu simulieren.

## 2.5 Graphtransformation

Mittels *Graphtransformation* können dynamische Veränderungen an Objektsystemen modelliert werden. Prinzipiell werden Modelle als getypte und attributierte Graphen interpretiert, die mittels Graphtransformationsregeln verändert werden können.

Als Modell werden häufig gerichtete Graphen bestehend aus Graphknoten und Graphkanten verwendet. Die Knoten repräsentieren die unterschiedlichen Objekte, während die Kanten Beziehungen zwischen den einzelnen Objekten darstellen. Diese Beziehungen können unterschiedliche Relationen zwischen den Objekten beschreiben.

Um eine Simulation des modellierten Systems durchzuführen, werden Graphtransformationsregeln auf den Graphen angewendet. In den Regeln sind die vorgesehenen Aktionen des Systems modelliert. Durch die Anwendung von Graphtransformationsregeln wird die Struktur des modellierten Graphen verändert.

### 2.5.1 Graphen

Die Graphen, die mittels Graphtransformation verändert werden, sind *Objektgraphen*, die durch ihr zugehöriges Klassenmodell *getypt* sind. Objektgraphen sind bezogen auf die MOF (Abschn. 2.4) Teil der Ebene M0. Das dazugehörige Klassenmodell (Abb. 2.7)

ist Teil der Ebene M1 und instanziert dementsprechend den *Host-Graph* (Objektgraph) [RSV04]. Durch das Typen der Graphen wird die in einem Klassendiagramm festgelegte Struktur bezüglich Relationen und ihren Kardinalitäten sowie Art und Anzahl von Attributen und Operationen auf den Objektgraphen übertragen. Effektiv gibt der Typ-Graph sowohl die Grenzen als auch den "Modifikationsspielraum" für den Objektgraphen vor.

Abbildung 2.7 zeigt die Zusammenhänge zwischen Typ-Graph und Host-Graph und deren Einordnung in das MOF Metamodell (Abschn. 2.4) anhand eines Teilausschnitts des verwendeten Beispiels.

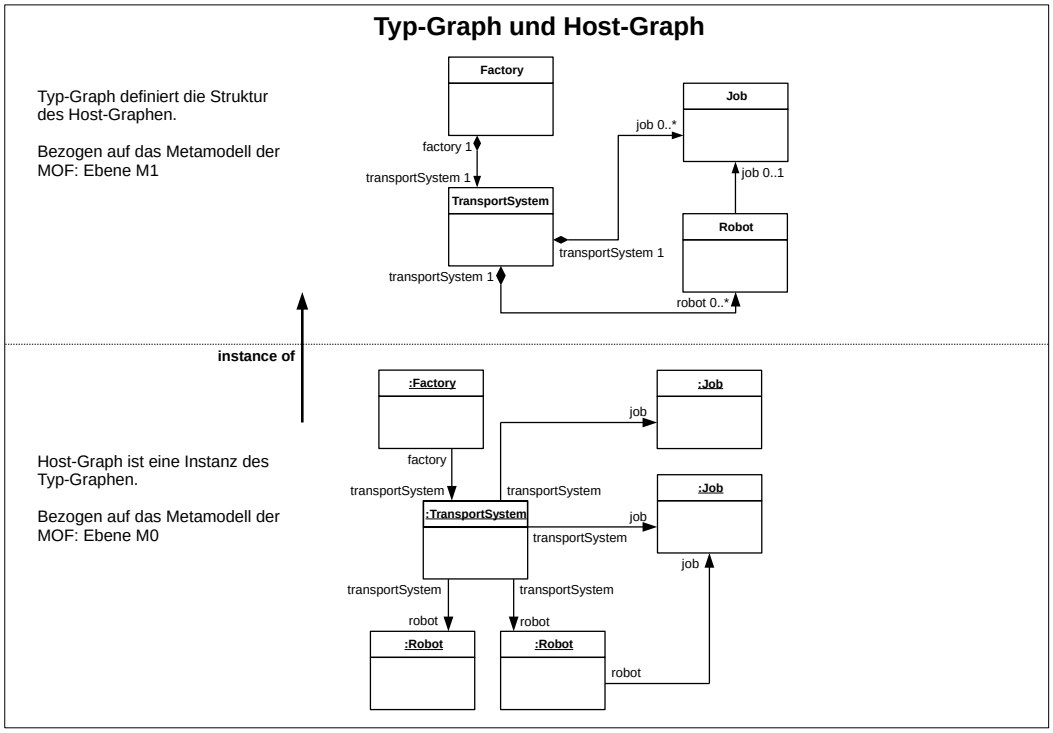


Abbildung 2.7: Typ-Graph und Host-Graph

Ein Knoten kann in unterschiedlichen Beziehungen zu anderen Knoten des Modells stehen. Diese Beziehung kann sich auf Zugehörigkeiten bezüglich bestimmter Kooperationen beziehen, die beispielsweise für die Beziehung zwischen einem Roboter und dem ihm zugeteilten Auftrag bestehen kann.

Der Objektgraph stellt die aktuelle Konfiguration des Systems nach einer bestimmten Anzahl an Aktionen des Systems dar. Dieser Zustand ist ein Ausschnitt des gesamten Zustandsraums des modellierten Systems. Es werden keine zeit-kontinuierlichen Veränderungen dargestellt, sondern es werden einzelne, die Struktur verändernde Schritte nacheinander unter der Anwendung von Graphtransformationsregeln durchgeführt. Diese Graphtransformationsregeln werden im Folgenden näher erläutert.

### 2.5.2 Graphtransformationsregeln

*Graphtransformationsregeln* (im Folgenden *Regeln*) beschreiben Veränderungen am Graphen, die mit Anwendung eben dieser Regeln umgesetzt werden. Regeln sind ebenfalls als Graphen modelliert.

Regeln bestehen üblicherweise aus zwei Graphen: Einer gibt die strukturelle Vorbedingung für die Anwendung der Regel an (left hand side; auch *LHS*), während der andere die Ausprägung des Graphen nach der Anwendung der Regel angibt (right hand side; auch *RHS*) [RSV04]. Die RHS gibt die Nachbedingung der Regel an.

Üblicherweise werden Regeln so notiert, dass man sie intuitiv von links (LHS) nach rechts (RHS) lesen kann. Sowohl für die LHS als auch die RHS ist es ausreichend, die relevanten Vorbedingungen und Veränderungen am Graphen zu modellieren. Teile des Modellgraphen, die nicht an der Umsetzung der Regel beteiligt sind, können für die Modellierung der Regel vernachlässigt werden, was der Übersichtlichkeit der Transformationsregeln zu Gute kommt.

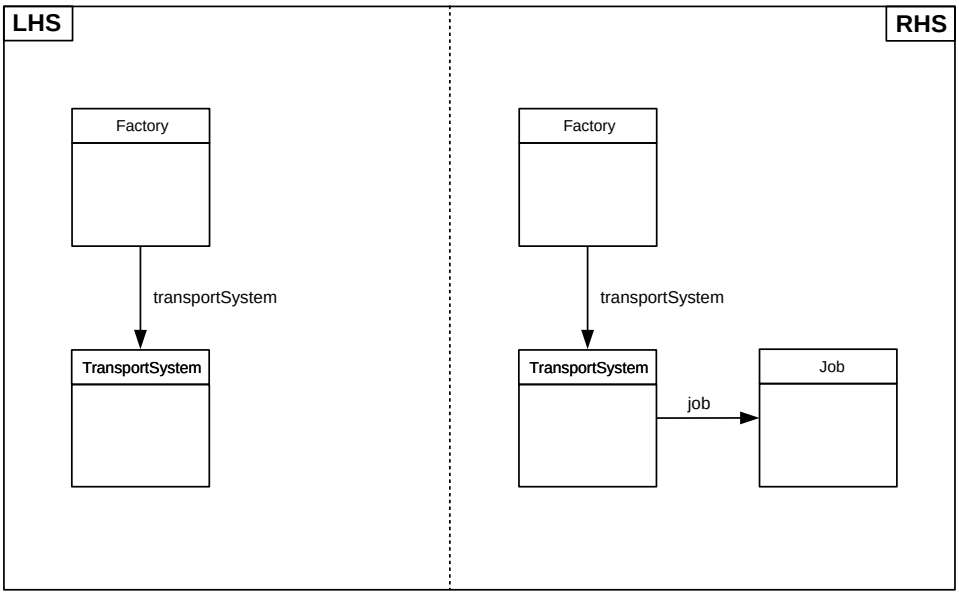


Abbildung 2.8: Graphtransformationsregel mit LHS und RHS

In Abbildung 2.8 ist eine Regel mit LHS und RHS dargestellt. Die exemplarisch modellierte Regel beschreibt den Vorgang der Erzeugung eines neuen Jobs durch das Transportsystem.

Die Vorbedingungen (LHS) für die Ausführung dieser Regel sind in diesem Fall einfach gehalten: Es muss eine Fabrik mit einem zugehörigen Transportsystem existieren. Wenn diese Voraussetzungen gegeben sind, kann die Regel ausgeführt werden. Das Graphtransformationssystem sucht dieses in der LHS modellierte Graphmuster. Wenn das der LHS

entsprechende Muster gefunden wurde, wird es gelöscht und durch den Teilgraphen der RHS ersetzt. Die Nachbedingungen (RHS) beschreiben den Zustand des Graphen nach Anwendung der Regel, wobei in diesem Fall ein Job-Objekt und die Relation zwischen dem Transportsystem und dem neuen Job-Objekt hinzugekommen sind. Die Regel erzeugt einen Job, der sodann dem Transportsystem zugeordnet wird. Dieses Verfahren wird als *pattern-matching* bezeichnet [RSV04].

Viele Werkzeuge zur Graphtransformation, auch das für diese Arbeit verwendete Tool, sind dazu übergegangen, Regeln als Kombination von LHS und RHS abzubilden.

In diesem Fall werden die Vor- und Nachbedingungen nicht mehr als zwei getrennte Teilgraphen dargestellt. Vielmehr wird den Elementen des Graphen durch Färbung oder zusätzliche Kennzeichnung eine Funktion in der Umsetzung der Regel zugeteilt.

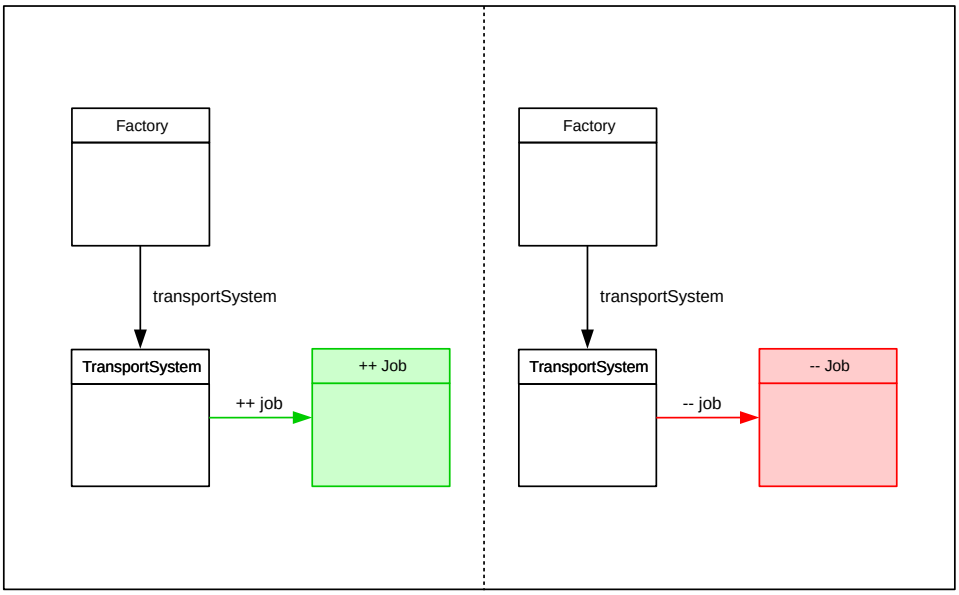


Abbildung 2.9: Graphtransformationsregeln

In Abbildung 2.9 ist auf der linken Seite die selbe Regel wie in Abbildung 2.8 umgesetzt. Die LHS und RHS wurden hier in einem Graphen zusammengefasst. Im Falle von ModGraph, aber auch in anderen Graphtransformationswerkzeugen, werden Elemente, die sowohl Teil der LHS als auch Teil der RHS sind, in einer unauffälligen Farbe (hier schwarz) eingefärbt. Diese Elemente erfahren unter Anwendung der Regel keine Veränderung.

Die grünen Elemente, die zusätzlich durch “++“ gekennzeichnet sind, beschreiben die Elemente, die unter Anwendung der Regel dem Graphen hinzugefügt werden. Häufig werden diese erzeugenden Elemente als “Creator“ bezeichnet [RSV04].

Auf der rechten Seite von Abbildung 2.9 ist analog zum vorherigen Beispiel die Löschung eines Jobs als Regel dargestellt. Die hier rot eingefärbten Elemente, die zusätzlich durch

"- -" gekennzeichnet sind, beschreiben Elemente, die durch die Anwendung der Regel aus dem Graphen gelöscht werden. Häufig werden diese Elemente als "Eraser" bezeichnet [RSV04].

Es gibt über diese Beispiele hinaus diverse andere Formen der Notation von Regeln. Häufig sind es attribut-abhängige Voraussetzungen oder negative Anwendungsbedingungen. Diese legen zusätzliche, über die reine Struktur hinausgehende, Voraussetzungen fest, die zur Ausführung der Regel benötigt werden. Negative Anwendungsbedingungen können dabei Konstellationen beschreiben, unter denen die Ausführung der Regel verhindert wird.

### 2.5.3 ModGraph

*ModGraph* ist ein Softwarewerkzeug, das mittels Graphtransformation (Kap. 2.5) Veränderungen auf Objektgraphen umsetzt [BWW11]. Auf diese Weise ist es möglich, strukturelle Dynamik sowohl in Bezug auf Datenobjekte als auch in Bezug auf Objekte der physikalischen Welt abzubilden.

Diese Herangehensweise erlaubt es dem Anwender, die strukturelle Dynamik eines modellierten Systems schrittweise oder in der ganzheitlichen Betrachtung komplexer Folgen von Arbeitsschritten zu simulieren.

Anhand der erzielten Ergebnisse können so Informationen über das modellierte System erlangt und unerwünschtes Systemverhalten bereits vor Beginn der Implementierung korrigiert werden.

Im Umfeld von ModGraph werden Ecore-Modelle (Abschn. 2.3.2) als Objektgraphen (Kap. 2.5.1) betrachtet. Auf diese Graphen können vorab definierte Regeln angewendet werden, die die Struktur des Graphen manipulieren.

Im Zuge dieser Regelanwendungen können Objekte hinzugefügt oder entfernt werden. Alternativ können Attribute von Objekten modifiziert oder Beziehungen zwischen den unterschiedlichen Objekten erzeugt, gelöscht oder verändert werden.

Regeln sind derart gestaltet, dass sie nur unter bestimmten Voraussetzungen angewendet werden können. Bedingungen, die Voraussetzung für die Anwendung einer Graphtransformation sind, können struktureller Natur sein oder auch als Bedingung in *Object Constraint Language (OCL)* textuell formuliert werden.

Beispielsweise ist es möglich, für eine Regel, die ein neues Objekt erzeugt, die Vorbedingung festzulegen, dass der als Parameter übergebene Name für das neue Objekt nicht *null* ist. Ein Fehlen dieses Parameters würde die Ausführung dieser Regel verhindern.

Es können auch festgelegte Graphzustände, die eine Regelanwendung explizit verhindern (*Negative Application Conditions*), der Transformationsregel hinzugefügt werden.

Die Verwendung von Graphtransformationen bietet alle Möglichkeiten, den Objektgraphen beliebig - im Rahmen der im Ecore-Modell festgelegten Struktur - zu manipulieren.

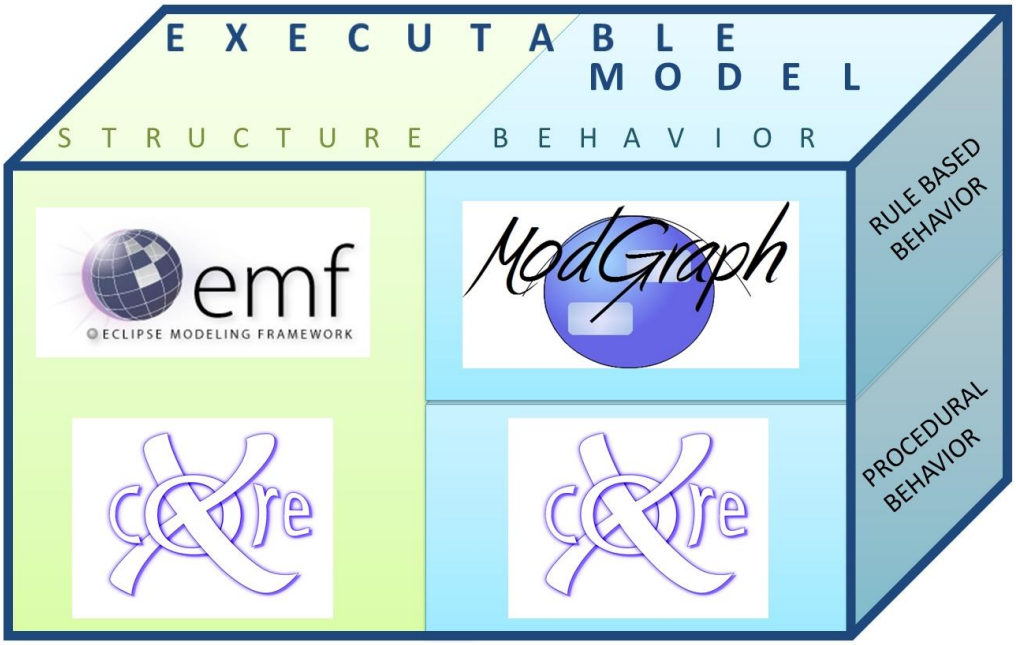


Abbildung 2.10: ModGraph Schema - entnommen von [Win13]

Diese Transformationen sind allerdings nicht in der Lage, selbst festzulegen, wann und in welcher Reihenfolge sie auf den Objektgraphen angewendet werden sollen.

Um zusätzlich das Verhalten der modellierten Systeme zu beschreiben, können mittels *Xcore* Kontrollstrukturen implementiert werden [Ecl12] [WW13]. Darauf aufbauend hängt die Anwendung einer Regel nicht mehr nur von der nötigen Graphstruktur der LHS ab. Vielmehr kann die Anwendung der Regel nun im Rahmen eines programmierten Systemverhaltens angewendet werden.

Xcore ist eine domänenspezifische textuelle Programmiersprache (domain specific language - DSL), mit der sich Ecore-Modelle auf eine alternative Art beschreiben lassen.

Abbildung 2.10 zeigt die Verwendung der unterschiedlichen Technologien, die für strukturelle und verhaltensorientierte Modellierung innerhalb ModGraph angewendet werden.

Ecore-Modelle, die mittels des EMF entwickelt werden, sind ausschließlich strukturelle Modelle. Wenn aus diesen Modellen Code generiert wird, ist der generierte Code auf bestimmte Operationen beschränkt. Diese Operationen sind sowohl Methoden, die Werte von Variablen verändern (Getter- und Settermethoden), als auch Methoden, die neue Objekte oder Beziehungen zwischen Objekten erzeugen oder entfernen [WW13].

Vom Benutzer definierte Methoden werden nur als leere Methodenrümpfe generiert. Xcore erweitert EMF dahingehend, dass der Benutzer die Möglichkeit hat, das auf Ecore basierende Strukturmodell nicht nur in Xcore zu beschreiben, sondern auch selbstständig über die Basismethoden hinausgehende Methoden zu definieren und dem modellierten System zusätzliche Funktionen zu implementieren [Win13].

Durch diese Kombination von Ecore-Modellen, ModGraph-Regeln und Xcore-Operationen ist es möglich, sowohl das Systemverhalten als auch auftretende strukturelle Veränderungen zu simulieren.

# 3 Beispiel und Anforderungsanalyse

Im Folgenden soll das Beispiel detailliert erläutert und die einzelnen Komponenten in ihrer Ausprägung und Funktion dargestellt werden. Anschließend werden die Anforderungen an die geplante Integration beider Modellierungstechniken in Bezug auf das Beispiel analysiert und formuliert.

## 3.1 Das Beispiel im Detail

Das beispielhaft entworfene reaktive System mit struktureller Dynamik ist als ein Materialversorgungssystem modelliert worden, das Mitarbeitern in einer Produktionseinrichtung die Möglichkeit gibt, Bauteile über einen Terminal zu bestellen. Diese Bauteile werden von autonom fahrenden Transportrobotern aus Lagerbereichen geholt und dem Mitarbeiter an seinem Mopntagearbeitsplatz zur Verfügung gestellt. In der Beschreibung der Komponenten des Beispielsystems kommt es zur Unterscheidung zwischen Systemkomponenten/ -teilnehmern und der Umwelt. Diese Unterscheidung dient in erster Linie der Verdeutlichung, ob ein Objekt vom System kontrollierbar oder Teil der mit dem System interagierenden Umwelt ist.

Die Abbildung 3.1 zeigt das Klassendiagramm des modellierten Beispiels. Im Folgenden werden die Klassen des UML-Modells näher erläutert und ihre Funktion im Gesamtsystem beschrieben.

1 ***Factory***

   Die Klasse Factory repräsentiert in diesem Beispiel die physische Fabrik. Sie stellt das Fabrikgebäude dar, in dem das Materialversorgungssystem eingesetzt werden soll.

   Sie dient als UML-Klasse ausschließlich als eine Art "Basisklasse", die die anderen Komponenten enthält, was die große Zahl an Containment-Beziehungen verdeutlicht. Sie verfügt über keine Operationen, die ausgeführt werden können.

2 ***Transportsystem***

   Die Klasse Transportsystem repräsentiert das Softwaresystem, das die Steuerung des Materialversorgungssystems umsetzt. Das Transportsystem nimmt eine zentrale Rolle im System ein. Es erzeugt neue und löscht bearbeitete Transportaufträge (6). Darüber hinaus teilt es den Robotern (3) die Transportaufträge zur Bearbeitung zu.

   Wenn eine Bestellung für ein Bauteil (8) erfolgt, generiert es einen entsprechenden

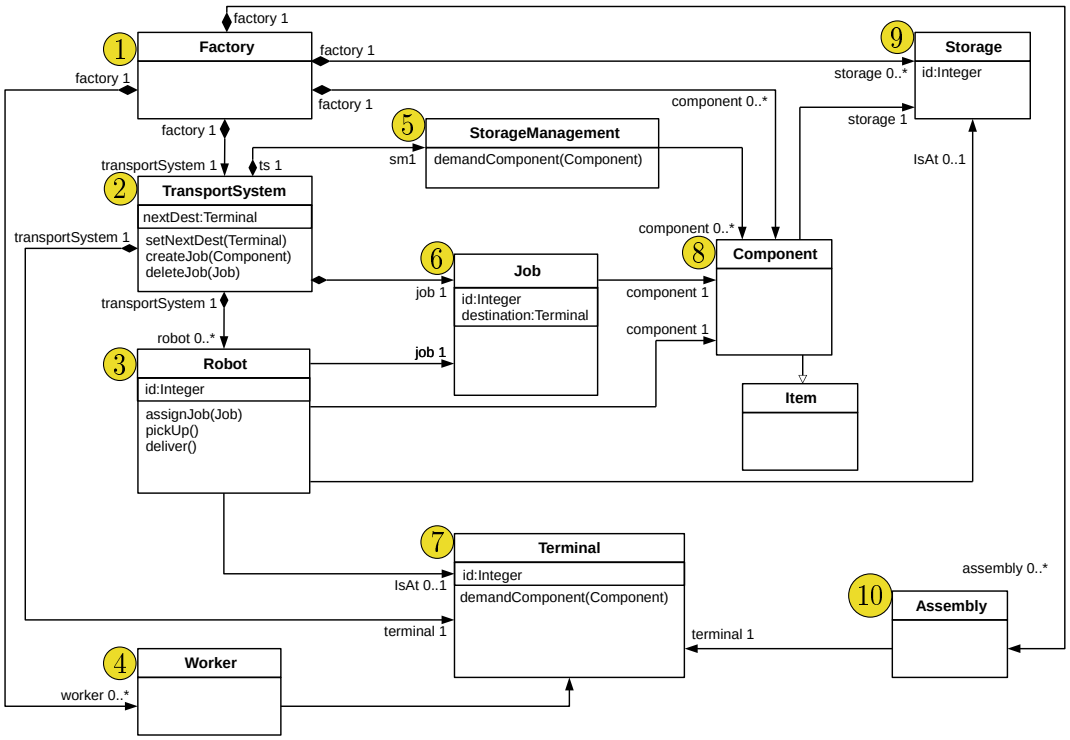


Abbildung 3.1: Klassendiagramm Beispiel

Job (6) unter Einbeziehung des angefragten Bauteils (8), seinem Lagerort (9) und dem Terminal (7), an dem die Bestellung getätigt wurde. Es verwaltet die zu bearbeitenden Transportaufträge und koordiniert die Zuweisung dieser Aufträge an die Roboter (3). Nach eingehender Bestätigung der Erledigung eines Jobs (6), wird der entsprechende Job (6) gelöscht.

**3** ***Robot***

Die Klasse Robot steht stellvertretend für einen Transportroboter. Der Roboter ist ein Systemteilnehmer, der selbstständig komplexe Aufgaben durchführt.

Der Roboter ist die Systemkomponente, die den größten Einfluss auf die strukturelle Dynamik des Systems hat. Er ist die einzige Systemkomponente, die selbstständig seinen Aufenthaltsort wechselt und gleichzeitig andere Objekte an einen anderen Ort befördert. Die Bearbeitung eines Jobs (6) umfasst mehrere Schritte:

- Abholen des angeforderten Bauteils (8) in einem der Lager (9).
- Abliefern des Bauteils (8) an dem dem Job (6) entsprechenden Terminal (7).
- Bestätigen der Durchführung des Jobs (6) an das Transportsystem.

Während der Durchführung dieser Aufgabe verändert der Roboter laufend seine Position in der Fabrik und bewegt Bauteile (8) von einem Lager (9) an den Terminal einer Montagestation (10). Insbesondere diese strukturelle Dynamik soll mittels Graphtransformation simuliert werden.

**4 *Worker***

Die Klasse Worker stellt die Mitarbeiter an den Montagestationen (10) der Fabrik dar und ist Teil der Umwelt. Er ist damit kein kontrollierbares Systemelement.

Der Mitarbeiter gibt durch die von ihm platzierten Bestellungen von Bauteilen (8) die Anfangsimpulse, die die Prozesse im System initiieren.

Seine Eingaben am Terminal (7), mithin die Bestellung eines Bauteils (8), stellen einen Umwelteinfluss dar, der ein nicht kontrollierbarer Impuls für das System ist. Solche Umwelteinflüsse lassen sich nicht durch das Materialversorgungssystem steuern; das System muss dennoch in der Lage sein, auf diese Impulse entsprechend zu reagieren.

**5 *Storagemanagement***

Die Klasse Storagemanagement ist ein reines Datenobjekt ohne einen konkreten Repräsentanten in der physischen Welt. Es ist die Instanz innerhalb des Systems, die Informationen über den Lagerbestand der Fabrik zur Verfügung stellt.

Bauteilbestellungen des Workers laufen immer über das Storagemanagement. Hier wird überprüft, ob das angeforderte Bauteil (8) aktuell vorrätig ist. Ist dies der Fall, wird ein Auftrag zur Erstellung eines Jobs (6) an das Transportsystem übergeben. Andernfalls wird eine Nachricht über das Nichtvorhandensein des angeforderten Bauteils (8) an den Terminal (7) übermittelt, an dem die Bestellung platziert wurde.

**6 *Job***

Die Klasse Job ist ein reines Datenobjekt ohne einen konkreten Repräsentanten in der physischen Welt.

Der Job ist in erster Linie ein Behälter für die relevanten Daten einer Materialbestellung. Er hat eine Beziehung (component) zu dem Bauteil (8), das bestellt wurde. Über die Beziehung (storage) zu dem Lager (9), in dem das Bauteil (8) aufbewahrt wird, ist der Ort der Abholung definiert. Darüber hinaus verfügt er über die Information, an welchen Terminal (7) das bestellte Bauteil (8) zu liefern ist.

**7 *Terminal***

Die Klasse Terminal repräsentiert ein Gerät, beispielsweise einen Computer, der die Schnittstelle zur Umwelt (Worker) ist.

Der Terminal dient als Benutzerinterface aus Sicht der Mitarbeiter an den Montagestationen.

Wenn ein Mitarbeiter ein Bauteil (8) anfordert, wird diese Anfrage vom Terminal an das Materialversorgungssystem weitergegeben. Je nachdem, ob das angeforderte Bauteil (8) verfügbar ist, gibt der Terminal eine Statusmeldung aus, die entweder angibt, dass das Bauteil (8) nicht vorrätig ist oder dass der Auftrag sich in Bearbeitung befindet. Des Weiteren ist der Terminal in der physischen Welt der Übergabeort des Bauteils (8) durch den Roboter.

**8** ***Component***

Die Klasse Component stellt die zu transportierenden Bauteile dar. Hierbei kann es sich in der Praxis um einzelne Maschinenteile, größere Baugruppen oder einen Behälter mit Verbrauchsmaterial handeln.

Die Klasse Component wird für gewöhnlich in einem der Lager (9) aufbewahrt.

In dem Moment, in dem ein Arbeiter ein Bauteil bestellt, prüft das Storagemanagement die Verfügbarkeit. Ist diese gegeben, wird das Bauteil nach der Zuteilung des Jobs informell an diesen gebunden, bis es zu seinem Zielort geliefert wurde und der Job gelöscht wird. Der Roboter nimmt dieses Bauteil während des Transportprozesses im Lager (9) auf und liefert es am Terminal an den Mitarbeiter aus.

**9** ***Storage***

Die Klasse Storage ist ein Materiallager und eine Komponente der physischen Welt der modellierten Fabrik.

Das Lager dient der Lagerung der durch die Klasse Component beschriebenen Maschinenteile. Es können sich in der Fabrik unterschiedliche Lager oder Lagerbereiche befinden, die von dem Roboter zwecks Abholung eines Bauteils angefahren werden können. Ein konkretes Bauteil ist immer genau einem Lager (-bereich) zugeordnet, bis es im Rahmen eines Transportauftrags abgeholt wird.

**10** ***Assembly***

Die Klasse Assembly bezeichnet einen Montageplatz, der beispielsweise ein Teilabschnitt einer Montagestraße ist, an dem einzelne Produktionsschritte durch die Mitarbeiter umgesetzt werden. Es ist eine Klasse, die einen Teil der physischen Welt der Fabrik beschreibt.

Zu einem Montageplatz gehört immer genau ein Terminal. Es gibt genau einen Mitarbeiter, der an diesem Montageplatz und dem entsprechenden Terminal arbeitet.

## 3.2 Anforderungsanalyse

Die prototypische Kopplung von ScenarioTools und ModGraph mit dem Ziel, insbesondere reaktive Systeme mit struktureller Dynamik modellieren/simulieren zu können, muss bestimmten Anforderungen genügen. In diesem Abschnitt werden die relevanten

Anforderungen näher erläutert.

Szenariobasierte Modellierung bietet einen Ansatzpunkt für die Modellierung von verteilten reaktiven Systemen mit struktureller Dynamik. Die Kommunikation zwischen Objekten und Teilsystemen lässt sich mit dieser Methode gut modellieren. Auch die Modellierbarkeit von Umwelteinflüssen auf das System und die entsprechende Reaktion des Systems ist möglich. Mit ScenarioTools als Softwaretool ist ebenfalls eine Plattform für die Simulation der Modelle gegeben. In Bezug auf die strukturelle Dynamik kann diese Art der Modellierung schnell unpraktisch werden oder sogar ganz an ihre Grenzen stoßen. Die Veränderung von einzelnen Attributen ist ohne weiteres möglich. Möchte ein Entwickler allerdings ein neu entstehendes Objekt mit mehreren neuen Beziehungen und einigen zu setzenden Attributen erzeugen, reicht der szenariobasierte Entwurf nicht mehr aus.

Graphtransformation bietet in Bezug auf die strukturelle Dynamik umfangreichere Möglichkeiten. Die strukturellen Änderungen, die innerhalb einer Regel möglich sind, sind nur durch die strukturellen Vorgaben im Typgraphen beschränkt. Aufgrund dieser Tatsache ergibt sich die Möglichkeit, im Zuge der Anwendung einer Regel auf den Hostgraphen komplexe strukturelle Dynamik zu simulieren und im Modell "sichtbar" zu machen.

Getrennt voneinander betrachtet, bieten beide Techniken jeweils die Chance, einen großen Teilbereich von reaktiven Systemen mit struktureller Dynamik zu modellieren. Abbildung 3.2 zeigt eine schematische Darstellung der angestrebten Integration der beiden Techniken, die durch die Kopplung der beiden vorliegenden Tools erzielt werden soll.

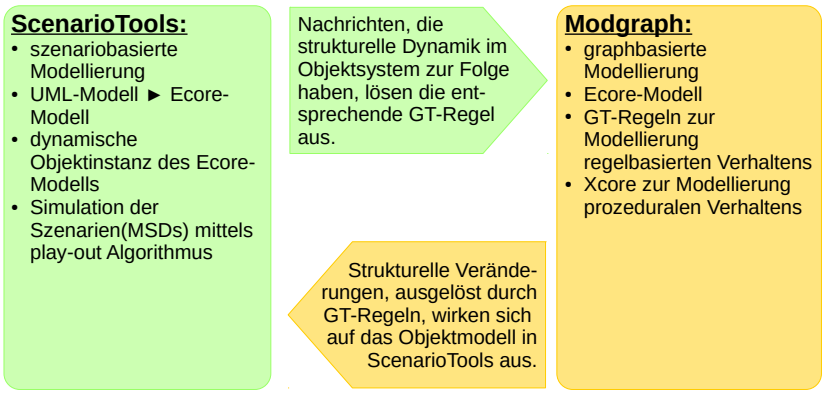


Abbildung 3.2: Diagramm Kopplung

Es ist erstrebenswert, dass beide Tools in der Lage sind, sich gegenseitig zu beeinflussen. Ziel dieser Integration ist es, dass einzelne Nachrichten in den Szenarien komplexe strukturelle Dynamik zur Folge haben können, wenn dies benötigt wird. Ein weiterer Aspekt der Integration ist, dass die Umsetzung von komplexer struktureller Dynamik eine Auswirkung auf das szenariobasierte Modell hat.

Es muss demnach eine Verbindung zwischen beiden Tools etabliert werden, die es erlaubt, Graphtransformationsregeln (ModGraph) und Objekte der dynamischen Objektinstanz (ScenarioTools) von dem jeweils anderen Tool aufzurufen bzw. zu verändern.

Abbildung 3.3 zeigt die Simulationsansicht von ScenarioTools. In der abgebildeten Ansicht wird die Simulation in ScenarioTools ausgeführt und die eintretenden Effekte werden dargestellt. Hierzu gehören unter anderem Konflikte bezüglich der Ausführung von Nachrichten. Identische Nachrichten sind teilweise in unterschiedlichen MSDs zu finden, wenn z.B. eine Nachricht einen Prozessschritt in zwei unterschiedlichen Szenarien darstellt.

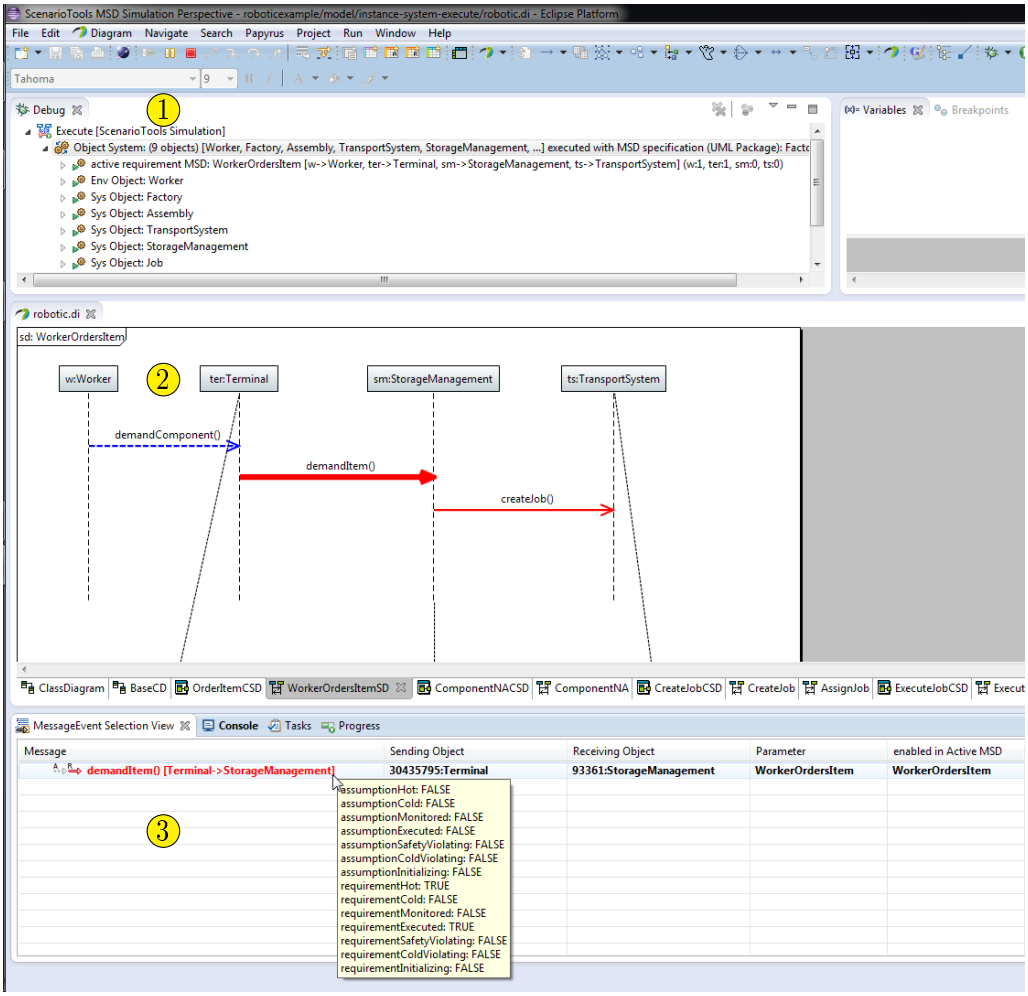


Abbildung 3.3: Screenshot der Simulationsumgebung von ScenarioTools

Die Simulationsoberfläche (Abb. 3.3) von ScenarioTools ist in die Debug-Perspektive von Eclipse integriert. Hier finden sich die aktiven MSDs und die darin enthaltenen Objekte (1). Zusätzlich hierzu können auch sämtliche Sequenz- und Klassendiagramme während der Simulation beobachtet werden (2). Im Message Event Selection View finden sich die aktuell innerhalb des Szenarios ausführbaren Nachrichten (3). Durch einen "Doppelklick" auf die Nachricht wird das Szenario einen Schritt weitergeführt.

Der zentrale Aspekt der geplanten Toolkopplung ist, dem Anwender die Möglichkeit zu geben, Nachrichten, die strukturelle Veränderungen innerhalb des Systems auslösen, derartig anzupassen, dass sie im Zuge ihrer Ausführung in ScenarioTools eine Graphtransformationsregel in ModGraph auslösen. Im Gegenzug sollte die Möglichkeit bestehen, Regeln in ModGraph zu entwerfen, die die Objektstruktur des Modells in ScenarioTools manipulieren.

Anhand von zwei Beispielen soll kurz erläutert werden, welche Effekte im Rahmen der Simulation erzielt werden sollen:

Abbildung 3.4 zeigt das MSD und die Regel, die die Nachrichtenkommunikation zwischen den beteiligten Objekten und die strukturelle Dynamik modellieren. Die strukturelle Dynamik bezieht sich in diesem Fall auf die virtuelle Struktur. Es wird ein Job (Datenobjekt) erzeugt und je eine Beziehung zu dem Transportsystem und zu dem Bauteil erzeugt, das Gegenstand dieses Transportauftrages ist. Diese Art von struktureller Dynamik ist durch die Verarbeitung eines Seiteneffekts einer Nachricht in dem MSD nicht mehr realisierbar, besonders in Bezug auf den Aspekt, dass in ScenarioTools nur ein Parameter je Nachricht zugelassen ist. Der gewünschte Effekt des Erzeugens des Jobs und den beiden zugehörigen Beziehungen sollte daher mittels Graphtransformation umgesetzt werden.

Abbildung 3.5 zeigt das MSD und die Regel ExecuteJob bzw. pickUp(). Das MSD executeJob modelliert die nötigen Einzelschritte zur Bearbeitung des Transportjobs durch den Transportroboter. Die Regel pickUp() dient in diesem Fall als Beispiel für strukturelle Dynamik in der physischen Welt. Der erste Teilschritt im MSD executeJob ist die Nachricht pickUp() an den Roboter, die ihn dazu veranlassen soll, das von dem ihm zugeteilten Job betroffene Bauteil im Lager abzuholen. Wie in der Regel pickUp() zu sehen ist, begibt sich der Roboter in das Lager (++isAt) und nimmt das Bauteil auf (++component). Die gelöschte Beziehung vom Bauteil zu dem Lager (- -storage) soll verdeutlichen, dass sich das Bauteil nicht mehr im Lager, sondern von nun an auf dem Roboter befindet.

In den Abbildungen 3.6 und 3.7 sind der Vollständigkeit halber noch die ergänzenden MSDs und Regeln aufgeführt, die zur Umsetzung des Bestellvorgangs eines Bauteils bis zu seiner Lieferung an den Terminal nötig sind.

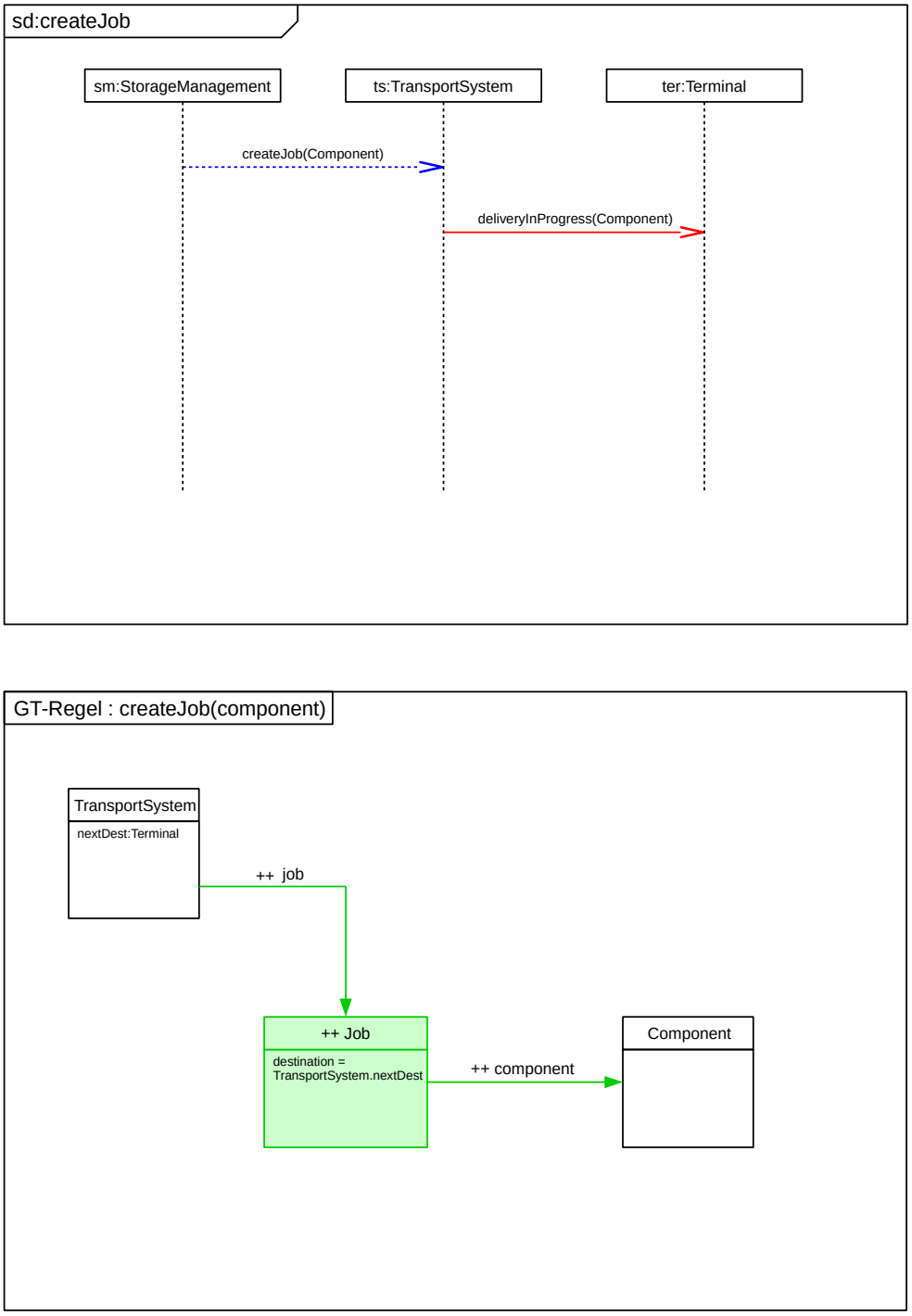


Abbildung 3.4: MSD und Regel: createJob

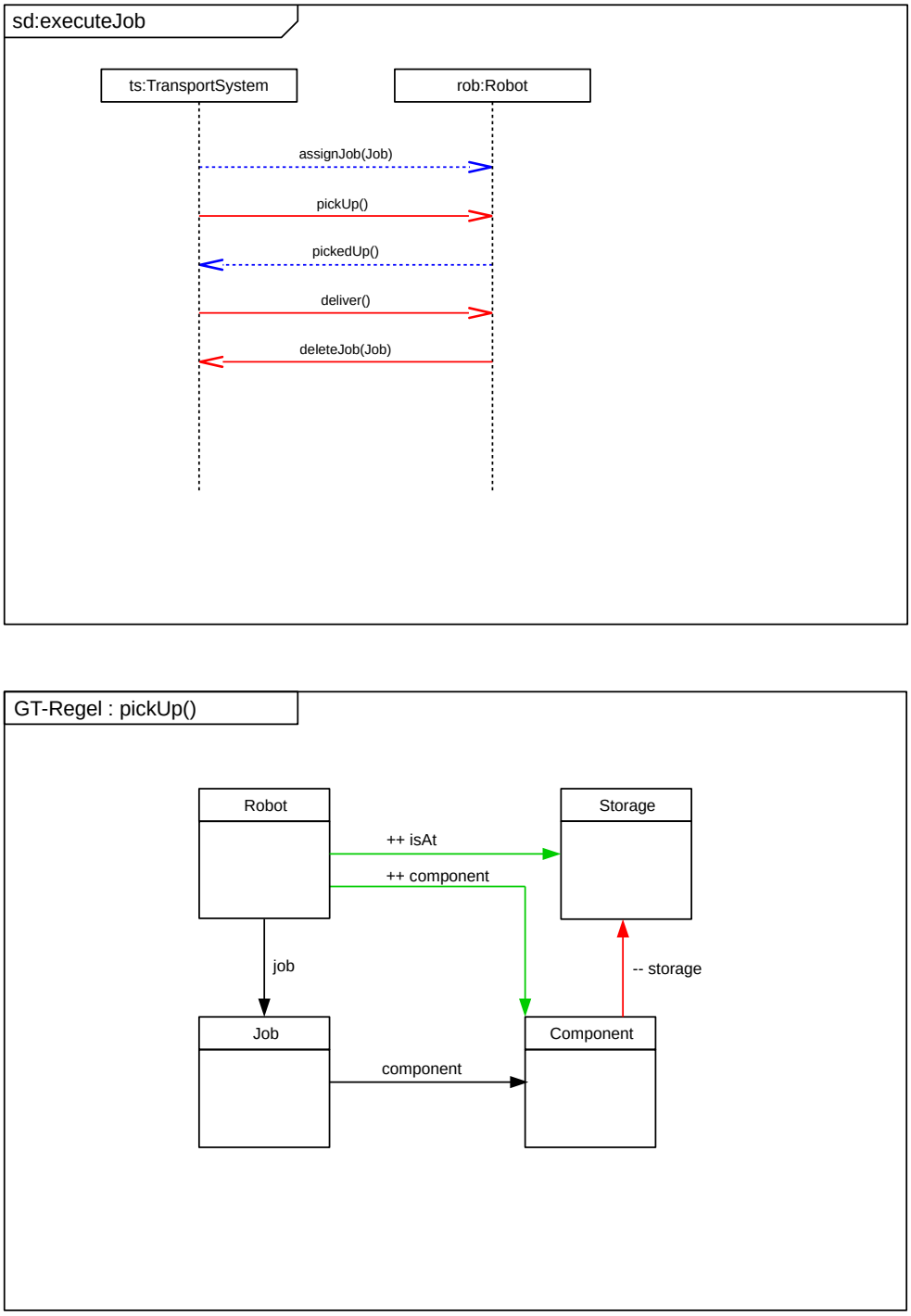


Abbildung 3.5: MSD: executeJob und GT-Regel:pickUP

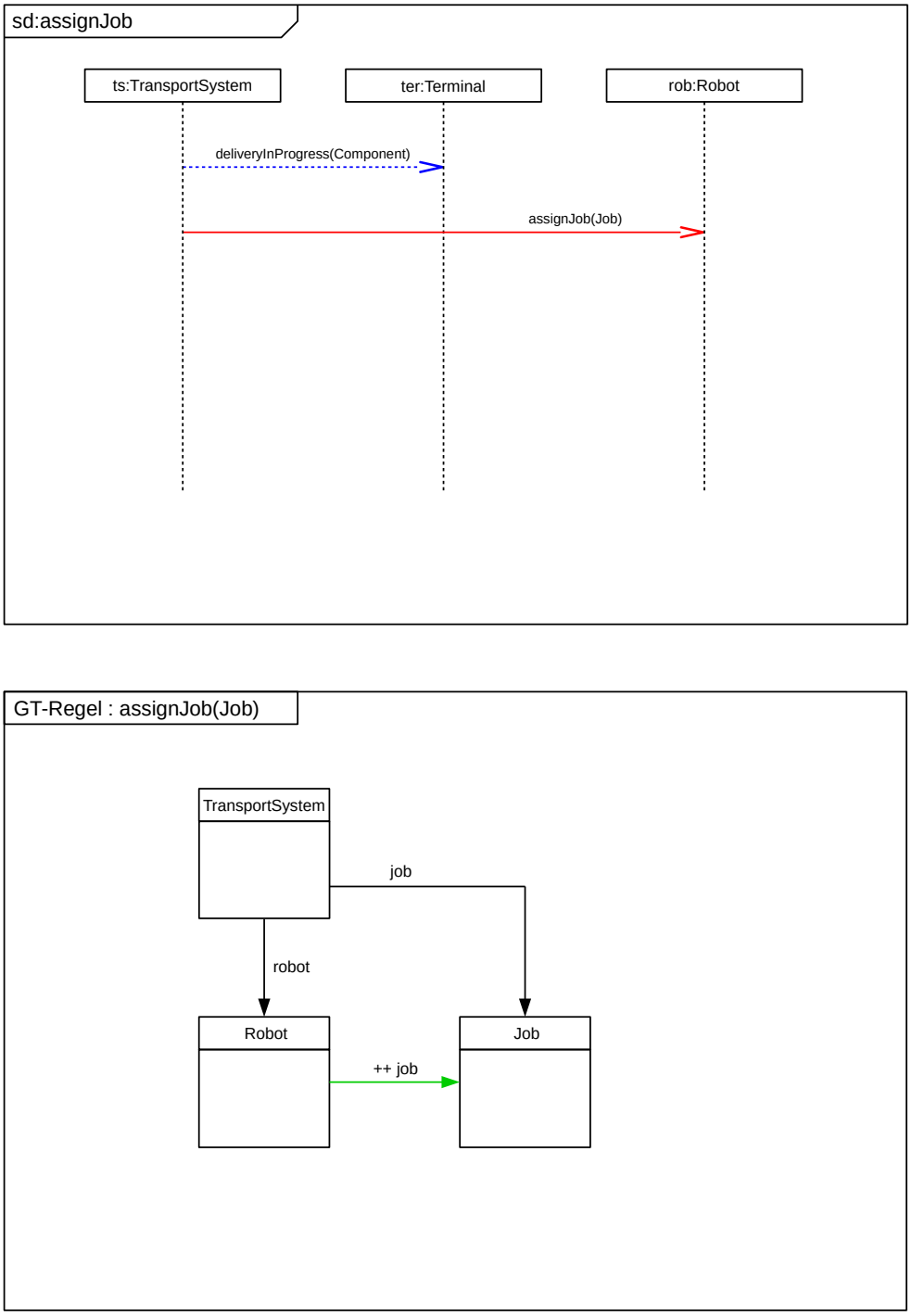


Abbildung 3.6: MSD und Regel: assignJob

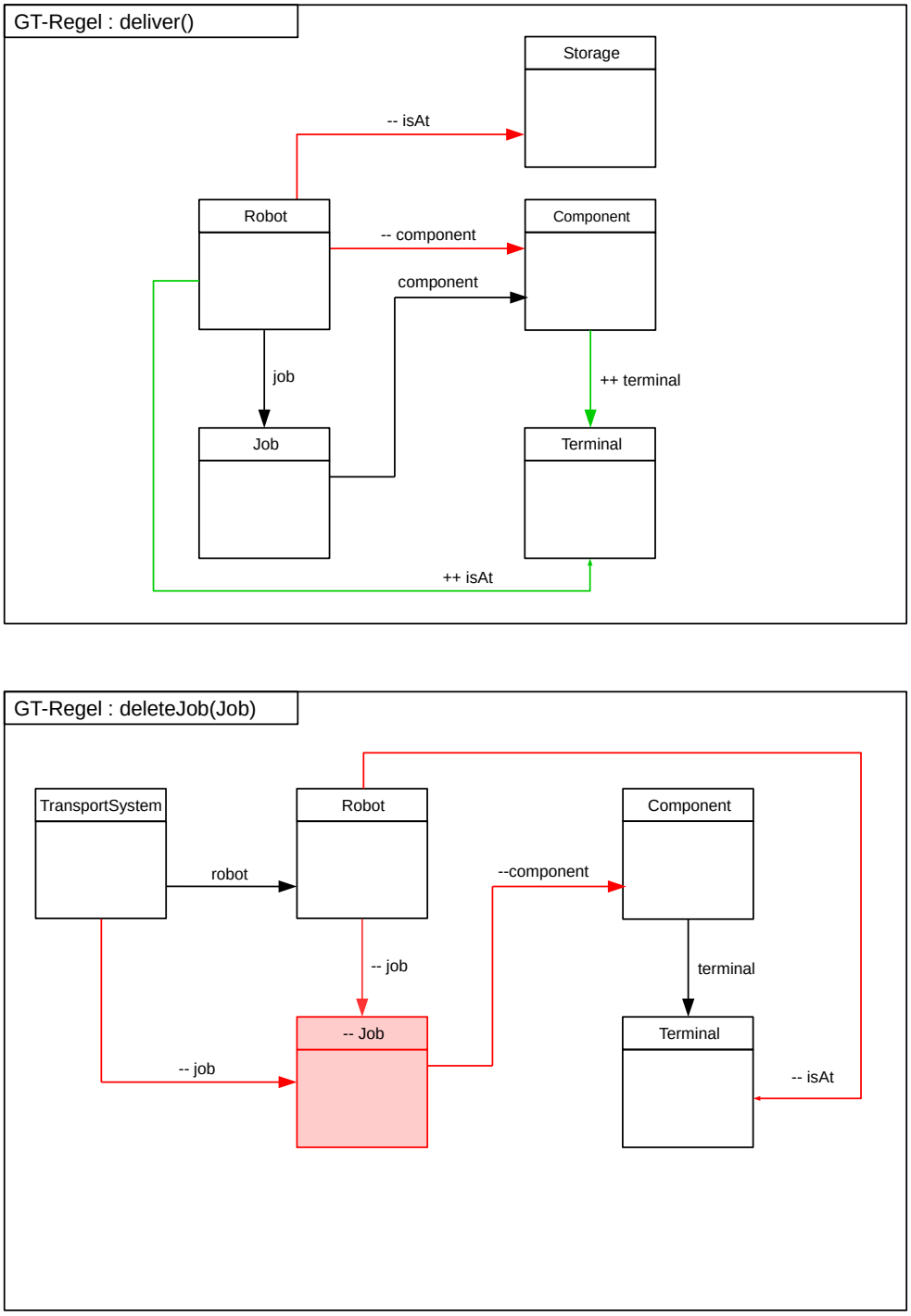


Abbildung 3.7: GT-Regeln deliver und deleteJob

# 4 Lösungskonzept

Dieses Kapitel beschreibt die geplante Herangehensweise an die Kopplung von ScenarioTools und ModGraph. Konzepte und Komponenten, die in die Umsetzung einfließen, werden hier näher erläutert.

In Abschnitt 3.2 sind die funktionalen Anforderungen an die Toolkopplung beschrieben worden. Abbildung 4.1 zeigt schematisch die für die jeweiligen Tools relevanten Objekte. Beide Tools verwenden als Basis für die Simulation/Graphtransformation Ecore-Modelle. In Bezug auf ScenarioTools muss erwähnt werden, dass hier die dynamische Instanz Teil der Simulation ist, die jedoch eine Instanz des zugrundeliegenden Ecore-Modells ist.

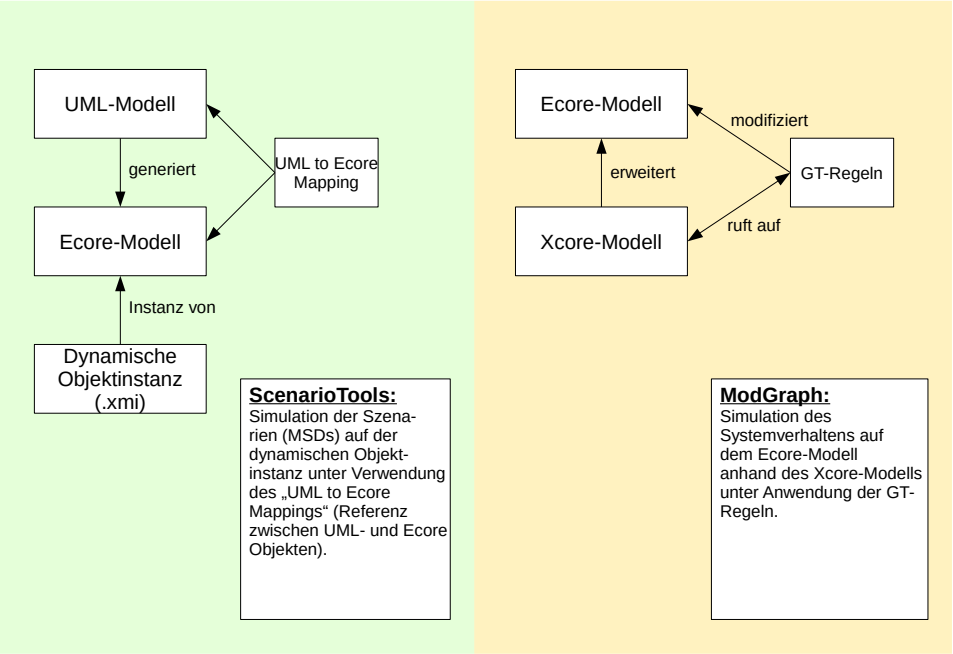


Abbildung 4.1: Schema Tools vor der Kopplung

Abbildung 4.2 beschreibt schematisch die Vision der geplanten Toolkopplung. Die zentrale Idee, die der Kopplung zugrunde liegt, ist die mögliche gemeinsame Nutzung eines Ecore-Modells für beide Tools. Verglichen mit Abbildung 4.1 besteht der Unterschied darin, dass beide Tools dasselbe Modell als Basis der Simulation verwenden sollen. Unter dieser Voraussetzung stellte es sich für beide Seiten deutlich einfacher dar, Objekte im

Modell zu referenzieren. Die zu etablierende Verknüpfung stellt einen großen Schritt in der Integration beider Techniken dar. Wenn es möglich ist, über Graphtransformationen die dynamische Instanz des Ecore-Modells zu verändern, würde dies einen großen Fortschritt bedeuten.

Um diesen Ansatz weiter zu verfolgen, wurde dem Verfasser der Arbeit ein modifiziertes ModGraph-Plugin zur Verfügung gestellt, das die Möglichkeit schafft, alle nötigen Komponenten eines ModGraph-Projektes innerhalb eines ScenarioTools-Projekts zu erzeugen und dort zu verwenden.

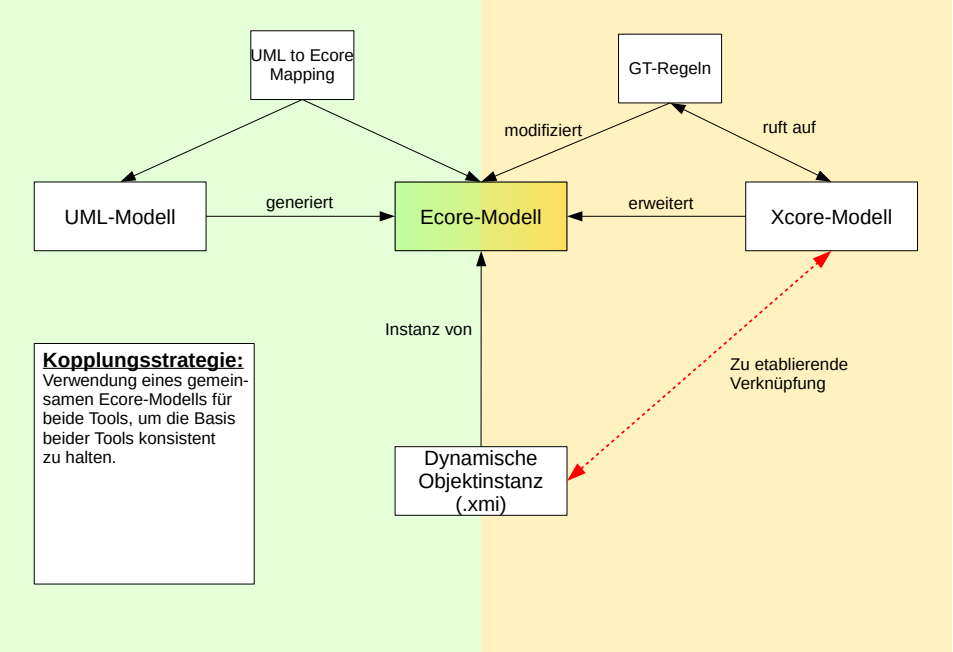


Abbildung 4.2: Schema Tools nach der Kopplung

Ein weiterer Aspekt, der die Verknüpfung zwischen den Nachrichten in MSDs in ScenarioTools und den Graphtransformationsregeln in ModGraph ermöglichen soll, ist die Tatsache, dass Nachrichten in MSDs Seiteneffekte haben können. Diese Seiteneffekte werden im Rahmen der Simulation mit ScenarioTools ausgewertet und üblicherweise dafür verwendet, Attribute des die Nachricht empfangenden Objekts zu verändern. Diese Verarbeitung der Seiteneffekte soll dahingehend erweitert werden, dass es möglich ist, aus der Verarbeitung der Seiteneffekte heraus Graphtransformationsregeln aufzurufen und damit strukturelle Veränderungen am Modell vorzunehmen.

Die Ausführung einer Regel aus der Verarbeitung eines Seiteneffekts könnte über einen *eInvoke* ermöglicht werden. EInvoke ist eine Operation, die in dem Interface *EObject* implementiert ist [MS09]. EObject ist eine Klasse, die im Rahmen des EMF verwendet wird. EObject ist die Klasse, von der alle Objekte, die in einem Ecore Modell instanziert werden, erben. Die Operation eInvoke ist demnach Teil jedes EObject. EInvoke

eröffnet die Möglichkeit, eine EOperation (eine in einem Ecore-Modell implementierte Operation) aufzurufen.

Damit sollte es möglich sein, eine EOperation über einen eInvoke während der Verarbeitung eines Seiteneffekts aufzurufen. Das Ziel eines solchen Aufrufs sollte nun eine Operation in dem ModGraph zugrundeliegenden Xcore-Modell sein. Es existiert eine XcoreMapper-Klasse als Teil des EMF (Abschn. 2.3.2). In dieser Klasse sind Methoden implementiert, die es ermöglichen sollen, Elemente eines Ecore-Modells auf Elemente eines Xcore-Modells, und umgekehrt, abzubilden.

Unter Verwendung der Methode eInvoke und dem XcoreMapper sollte es möglich sein, während der Verarbeitung des Seiteneffekts einer Nachricht, eine Methode des Xcore-Modells aufzurufen und damit eine Graphtransformationsregel anzuwenden. Dieser Ansatz, in Verbindung mit der Möglichkeit, ModGraph-Funktionalitäten innerhalb eines ScenariTools-Projekts zu verwenden, könnte die Chance eröffnen, eine technische Kopplung von ScenarioTools und ModGraph umzusetzen und damit eine Integration von szenariobasierter Modellierung und Graphtransformation zu erzielen.

# 5 Implementierung

Leider war es nicht möglich, innerhalb der Bearbeitungszeit mit der Implementierung der Toolkopplung zu beginnen. Dennoch sollen an dieser Stelle kurz die vorgesehenen notwendigen Schritte erwähnt werden, die zur Umsetzung der Implementierung nötig wären.

Ein wichtiger Punkt ist es, zu ermitteln, ob der Xcore-Mapper das Mapping zwischen Ecore-Modell und Xcore-Modell, speziell EOperation und XOperation, möglich macht. Darauf aufbauend ist zu prüfen, ob es möglich ist, aus der Verarbeitung eines Seiteneffekts in ScenarioTools heraus eine XOperation mittels eInvoke aufzurufen und hierdurch eine Graphtransformationsregel zu aktivieren. Anschließend wäre zu ermitteln, ob die Anwendung der Graphtransformation das Modell derartig verändert, dass ScenarioTools mit der Simulation des Modells fortfahren kann, ohne dass es zu Abbrüchen kommt.

Wenn es möglich sein sollte, diese oben genannten Schritte umzusetzen, bliebe abschließend zu überprüfen, ob der im Beispiel beschriebene Prozess mit den die Kopplung betreffenden Modifikation erfolgreich umsetzbar ist.

# 6 Verwandte Arbeiten

In diesem Kapitel soll Bezug auf andere Arbeiten genommen werden, die sich mit der Verknüpfung unterschiedlicher Techniken im Bereich des Software Engineering befasst oder eine der dieser Arbeit zugrundeliegenden Techniken in einem anderen Kontext verwendet haben. Besonders Techniken der Modellierung und Simulation sind hierbei von Interesse.

Liang et al. [LDD06] widmen sich in ihrem Artikel eines Konferenzberichts mit dem Titel *A Comparative Survey of Scenario-based to State-based Model Synthesis Approaches* der Analyse von Synthesemethoden zwischen szenariobasierten und zustandsbasierten Modellen.

Unter anderem wird in diesem Artikel darauf eingegangen, dass mit szenariobasierten und zustandsbasierten Modellen unterschiedliche Teilbereiche der Softwareentwicklung erleichtert werden.

Während szenariobasierte Modelle intuitiver zu verstehen seien, eigneten sie sich besser für die Kommunikation der erhobenen Anforderungen mit möglichen Kunden. Zustandsbasierte Modelle hingegen seien für die Generierung von Quellcode besser geeignet. Szenarien oder auch daraus entwickelte Use-Cases könnten jedoch eine treibende Kraft in der Softwareentwicklung darstellen, besonders in Bezug auf die Erstellung von Testfällen.

Als Einsatzbereich für diese Methoden der Modellsynthese wird besonders die Entwicklung von reaktiven Systemen herausgestellt. Gerade in der Entwicklung derartiger Systeme seien beide Verfahren nicht getrennt, sondern aufgrund der sich ergebenden Möglichkeiten als eng verbunden zu betrachten.

Ermel et al. [EHKZ05] widmen sich in ihrem Artikel eines Konferenzberichts mit dem Titel *Animated Simulation of Integrated UML Behavioral Models based on Graph Transformation* der Entwicklung einer Methode zur Überführung von UML-Modellen in eine Graphtransformationsumgebung zur grafischen Simulation und Analyse von Systemen.

Die Autoren bemerken, dass unterschiedliche UML-Diagramme verschieden Aspekte eines modellierten Softwaresystems beschreiben können, es allerdings schwierig sei, die Elemente der Diagramme untereinander zu verknüpfen und in Relation zueinander zu setzen. Die ähnliche Struktur von Klassendiagrammen und Graphen gab den Impuls, die Überführung von UML-Modellen in ein Graphtransformationssystem vorzunehmen.

Die Überführung von UML-Modellen in Graphtransformationssysteme wurde dahingehend weitergeführt, dass Graphtransformationen in Animationen basierend auf den Graphtransformationsregeln ausgeführt werden konnten. Der gewonnene Mehrwert, der

aus der Kombination dieser Verfahren hervorgeht, sei die Möglichkeit, eine flexible domänenbezogene Animation von modellierten Systemen zu erhalten.

Die beiden hier erwähnten Arbeiten dokumentieren, dass in unterschiedlichen Anwendungsbereichen schon Versuche unternommen wurden, unterschiedliche Techniken miteinander zu verbinden, um besonders auf dem Gebiet der Modellierung aussagekräftigere Modelle zu erzeugen.

# 7 Zusammenfassung und Ausblick

In diesem Kapitel werden eine Zusammenfassung und ein Fazit bezüglich der Arbeit formuliert. Im Ausblick wird ein kurzer Überblick über kommende Herausforderungen und Chancen in Bezug auf die Thematik der Arbeit geboten.

## 7.1 Zusammenfassung

Ziel dieser Arbeit war es, zwei Modellierungstechniken - szenariobasierte Modellierung und Graphtransformation - zu integrieren, um dadurch die Aussagekraft der erstellten Modelle zu verbessern, und schließlich eine technische Integration dieser Techniken, in Form einer Kopplung von zwei Modellierungstools, durchzuführen.

Zu Beginn wurden die beiden Techniken und die ausgewählten Tools auf ihre innere Funktion und die zum Einsatz kommenden Methoden untersucht, um mögliche Ansatzpunkte für eine Integration auszumachen. Die Ermittlung der verwendeten Methoden führte zu einem umfassenderen Verständnis der zu integrierenden Techniken. Parallel dazu wurde das Beispiel entwickelt, das sich an den aktuellen Herausforderungen in der Planung von reaktiven Systemen mit struktureller Dynamik orientiert.

Erste Versuche, das Beispiel mit beiden Techniken zu modellieren, führten an die technischen Grenzen der beiden einzelnen Modellierungstechniken und verdeutlichten gleichzeitig die Möglichkeiten, die die Integration beider Techniken bieten könnte. Für das Beispiel wurde ein Prozess definiert, der in beiden Techniken modelliert worden ist. Daran anschließend wurden die Anforderungen an die Integration beider Techniken formuliert. In diesem Schritt wurden die theoretischen Grundlagen mit den erhofften Ergebnissen der Kopplung in Einklang gebracht und auf diesem Wege die Basis für den Lösungsansatz geschaffen.

Für den Lösungsansatz wurde eine theoretische Verknüpfung beider Techniken und Tools entwickelt und der Versuch unternommen, die technische Umsetzung dahingehend vorzubereiten. Hierfür wurde dem Autor ein modifiziertes Plugin des Tools ModGraph von dessen Entwicklern zur Verfügung gestellt, was die Kopplung sichtlich vereinfachen sollte.

Eine Implementierung der Toolkopplung konnte leider im Rahmen der Bearbeitungszeit nicht umgesetzt werden, wodurch diese Aufgabe noch Potenzial zur weiteren Bearbeitung offen lässt.

Schlussendlich bleibt festzustellen, dass bereits die theoretische Integration beider Techniken aufgezeigt hat, dass eine Weiterverfolgung dieses Ansatzes wertvolle Möglichkeiten

im Bereich des Software Engineering erschließen kann.

## 7.2 Ausblick

Die technische Entwicklung der Vergangenheit hat gezeigt, dass auch in Zukunft mit einer wachsenden Komplexität von Systemen zu rechnen ist. Die wachsende Komplexität und der damit größer werdende informationstechnische Vernetzungsgrad haben eine immer aufwendiger werdende Planung der zu entwickelnden Systeme zur Folge.

Es gibt, wie schon bereits in der Arbeit erwähnt, die Vision, dass sich verteilte reaktive Systeme mit struktureller Dynamik in naher Zukunft über die Grenzen von Fabriken hinaus erstrecken werden. Schlagworte dieser Entwicklung zu dem immer größer werdenden Vernetzungsgrad sind unter anderem “Cyber Physical Production Systems“ und “Industrie 4.0 “.

Schon heute, und auch in Zukunft, müssen Entwickler unterschiedlicher Disziplinen diesen stetig größer werdenden Entwicklungsprojekten mit geeigneten Methoden und Werkzeugen begegnen. Es ist anzunehmen, dass es nötig sein könnte, viele unterschiedliche Techniken der Systemplanung und Entwicklung zu integrieren, um mit den technischen Herausforderungen der nahen Zukunft adäquat umgehen zu können.

# Literaturverzeichnis


[BGP13] Christian Brenner, Joel Greenyer, and Valerio Panzica La Manna. The ScenarioTools Play-Out of Modal Sequence Diagram Specifications with Environment Assumptions. In *Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2013)*, volume 58. EASST, 2013.

[BTF+02] Xiaoying Bai, Wei-Tek Tsai, Ke Feng, Lian Yu, and Ray Paul. Scenario-Based Modeling And Its Applications. In *Proceedings of the The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, WORDS '02, page 253 ff, Washington, DC, USA, 2002. IEEE Computer Society.

[BWW11] Thomas Buchmann, Bernhard Westfechtel, and Sabine Winetzhammer. MODGRAPH - A Transformation Engine for EMF Model Transformations. In *Proceedings of the 6th International Conference on Software and Data Technologies*, pages 212 – 219, 2011.

[DH01] Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[Ecl12] Eclipse Foundation. Xcore. `http://wiki.eclipse.org/Xcore`, 2012. Wikisite Xcore. Herausgegeben von der Eclipse Foundation.

[EHKZ05] Claudia Ermel, Karsten Holscher, Sabine Kuske, and Paul Ziemann. Animated Simulation of Integrated UML Behavioral Models Based on Graph Transformation. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, VLHCC '05, pages 125–133, Washington, DC, USA, 2005. IEEE Computer Society.

[Fas12] FastCompany. Most innovative companies 2012 - 23 kiva systems. `http://www.fastcompany.com/3017373/most-innovative-companies-2012/23kiva-systems`, 2012. Fast Company Website.

[Gre14] Joel Greenyer. ScenarioTools. `http://scenariotools.org/`, 2014. Website ScenarioTools.

[HKM07] David Harel, Asaf Kleinbort, and Shahar Maoz. S2a: A Compiler for Multimodal UML Sequence Diagrams. In Matthew B. Dwyer and Antónia Lopes, editors, *FASE*, volume 4422 of *Lecture Notes in Computer Science*, pages 121–124. Springer, 2007.

[HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Program-*

*ming Using LSC's and the Play-Engine*. Springer-Verlag New York, Inc., 2003.

[HM08] David Harel and Shahar Maoz. Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. In *Software and System Modeling*, volume 7, pages 237–252. Springer-Verlag, 2008.

[ITU96] ITU - International Telecommunication Union. Message Sequence Chart (MSC). Technical report, 1996.

[K+13] Stefan Kowalewski et al. Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation. Technical report, VDI/VDE - Gesellschaft Mess- und Automatisierungstechnik, April 2013. Zusammengestellt vom Fachausschuss 7.20 "Cyber-Physical Systems" der VDI/VDE - Gesellschaft Mess- und Automatisierungstechnik.

[LDD06] Hongzhi Liang, Juergen Dingel, and Zinovy Diskin. A Comparative Survey of Scenario-based to State-based Model Synthesis Approaches. In *Proceedings of the 2006 International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools*, SCESM '06, pages 5–12, New York, NY, USA, 2006. ACM.

[Lee08] Edward A. Lee. Cyber physical systems: Design challenges. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, May 2008. Invited Paper.

[MS09] Ed Merks and James Sugrue. Essential emf. `http://refcardz.dzone.com/refcardz/essential-emf`, 2009. Veröffentlicht im Rahmen der Eclipsecon 2009.

[RSV04] A. Rensink, Á. Schmidt, and D. Varró. Model checking graph transformations: A comparison of two approaches. In H. Ehrig, G. Engels, F. Parise-Presicce, and G. Rozenberg, editors, *International Conference on Graph Transformations (ICGT)*, volume 3256 of *Lecture Notes in Computer Science*, pages 226–241, Berlin, 2004. Springer Verlag.

[SBPM09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley, 2009.

[SVEH07] Thomas Stahl, Markus Völter, Sven Efftinge, and Arno Hase. *Modellgetriebene Softwareentwicklung*. dpunkt.verlag, 2. Auflage edition, 2007.

[Win13] Sabine Winetzhammer. ModGraph Behavior Modeling for EMF Based on Graph Transformation Rules. `http://btn1x4.inf.uni-bayreuth.de/modgraph/homepage/index.html`, 2013. Website des Softwaretools ModGraph.

[WW13] Sabine Winetzhammer and Bernhard Westfechtel. Modgraph meets Xcore: Combining Rule-Based and Procedural Behavioral Modeling for EMF. In *Proceedings of the 12th International Workshop on Graph Transformation*

*and Visual Modeling Techniques (GTVMT 2013)*, volume 58 of *Electronic Communications of the EASST*. EASST, March 2013.

# Abbildungsverzeichnis

# Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 29. Mai 2014

_____

Fabian Schmidt