Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Ausarbeitung
# Seminar

Modellgetriebene Softwareentwicklung reaktiver Systeme

Oliver Neumann

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*
*Betreuender Prüfer: Prof. H. Kaltenhäuser*

**Oliver Neumann**

**Thema der Ausarbeitung**
**Seminar**

Modellgetriebene Softwareentwicklung reaktiver Systeme

**Stichworte**

MDA, reaktive Systeme, XMI

**Kurzzusammenfassung**

Entwicklung von Software gestaltet sich zunehmend Komplexer. In vielen Bereichen der Softwareentwicklung ist daher eine Detaillierte Planung und Dokumentation eines Systems notwendig. Für Objektorientierte Systeme können Diagramme in Quellcode transformiert werden. Für technische Prozesse ist dies nicht der Regelfall. Diese Ausarbeitung soll Lösungswege aufzeigen, wie man reaktive Systeme modellieren kann, und welchen nutzen man daraus ziehen kann.

# Inhaltsverzeichnis

# 1 Einleitung

Das Implementieren von Anwendung gestaltet sich zunehmend komplexer. Ursachen für diese Entwicklung sind unter anderem die Komplexität der Systeme, sowie die zunehmende Anforderung das verteilte Systeme realisiert werden. Dieser Umstand macht das planen und strukturieren von Anwendungen unerlässlich. UML ist ein Standard, der Entwickler an dieser Stelle unterstützt. Objektorientierte Sprachen werden von vielen Werkzeugen unterstützt, da sich Klassendiagramme sehr gut auf konkreten Code-Modellen abbilden lassen. Für die gängigen Programmiersprachen ist Roundtrip-Engineering in vielen Werkzeugen implementiert, sodass Änderungen, die im Code oder im Modell gemacht werden, im jeweils Anderen abgeglichen werden. Das Gleiche gilt für das Entwickeln von Datenbanken, die mit Software-Engineering Werkzeugen in der oben beschriebenen Art und Weise entwickelt werden können.

Für technische Prozesse ist diese Art der Modellierung in wenigen Werkzeugen umgesetzt. Daraus ist zu schließen, das UML Diagrammtypen spezifiziert, die zur Beschreibung technischer Prozesse geeignet sind. Hier sind Statecharts, Activitydiagramme und Collaborationsdiagramme zu nennen. Echtzeitanforderungen, die in technischen Prozessen eine große Rolle spielen, lassen sich mit dem aktuellen Standard (UML2.0 OMG) ebenfalls modellieren. Über Stereotypen, die sich mit Hilfe von UML-Profilen definiert lassen. Für Echtzeitsysteme gibt es vorgefertigte Typen wie «RTClock» und «RTTimer». Des weiteren werden Ports spezifiziert, die es ermöglichen, Interfaces einzelner Komponenten eines Systems zu modellieren. Sie sind in OMG spezifiziert.

## 1.1 Motivation

In der Vergangenheit wurden Anwendungen mit Echtzeitanforderungen in Assembler implementiert. Das Laufzeitverhalten der Programme ließ sich effizienter Programmieren als durch Hochsprachen wie C. Durch steigende Performance von Systemen (Prozessor, Speicher) und effizientere Compiler war das Programmieren in Assembler nicht mehr nötig. Heute sind Werkzeuge und Methoden der Modellgetriebenen Softwareentwicklung so weit fortgeschritten, das es sich lohnt zu betrachten, wie sich diese Arbeitsweise auf technische Prozesse abbilden lässt. Durch den Ansatz der Modellgetriebenen Softwareentwicklung kann das Entwickeln von komplexen Anwendungen beschleunigt und durch die Wahl der höheren Abstraktionsebene deutlich simplifiziert werden. Ein weiterer Vorteil ist, das aus einem erstellten Modell mehr als eine Sprache generiert werden kann. Man kann also nicht nur Anwendungscode erstellen, sondern auch Code, der z.B. ein Automatenmodell auf Laufsicherheit prüft. Ein weiterer Vorteil für den Anwendungsentwickler ist sich mit dem konkreten Problem beschäftigen zu können, und nicht zusätzlich mit den Problemen einer Programmiersprache. Das Erzeugen von Code liegt in der Hand des Werkzeugs.

# 2 Reaktive Systeme

Reaktive Systeme grenzen sich durch ihre fortlaufende Interaktion mit ihrer Umgebung von transformationalen Systemen (z.B. Compilern) ab. Reaktive Systeme verarbeiten Eingaben und generieren Ausgaben fortlaufend und zueinander asynchron. Was meint, das die Ein- und Ausgaben spontan auftreten und zu jeder Zeit unvorhersehbare Werte annehmen können. In transformationalen Systemen ist das Timing von Ein- und Ausgabe deutlich besser vorhersehbar. Sie warten auf einen Eingabestrom, verarbeiten diesen und geben die Ergebnisse aus. Dieses deterministisches Verhalten ist charakteristisch für Systeme dieser Art.

## 2.1 Technische Prozesse

Technische Prozesse sollen an dieser Stelle in zwei Kategorien eingeteilt werden.

- Hoch reaktive technische Prozesse, die Aufgrund Ihrer hohen Anzahl an Zuständen und Zustandsübergängen so komplex werden, das die Modellierung mit Statecharts die Entwicklung von Anwendungen nicht vereinfacht. Für dieser Art von Systemen gibt es andere, vereinfachende Lösungsansätze, wie z.B. Subsumtion-Architekturen, die von Brooks (1985) beschrieben werden. Beispiele für diese Systeme sind mobile Roboter, die auf eine Vielzahl von Sensordaten unterschiedlich reagieren müssen.

- Technische Prozesse, die durch ihren Aufbau und ihre Abarbeitung einen begrenzte Ereignis- und Zustandsraum haben. Als Beispiel seien hier Produktionsstraßen genannt, die Werkstücke analysieren und bearbeiten.

Weitere Unterscheidungsmerkmale sind:

- Ereignisgetrieben $\leftrightarrow$ getaktet. Ein Prozess kann Ereignisbasiert ablaufen, oder durch ein strenges Zeitmuster umgesetzt werden. Getaktete Systeme haben hohe Echtzeitanforderungen. Ein Zeitraster garantiert, das diese eingehalten werden.

- Stückprozesse $\leftrightarrow$ Flussorientierte Prozesse. Produktionsstraßen oder Lüftungs- und Heizungssysteme.

- Gesteuert $\leftrightarrow$ geregelt. Gesteuerte Systeme bilden eine lineare Wirkungskette ohne Rückführung. Rückgekoppelte Systeme, in denen ein gemessener Ist-Wert in das Verhalten des Systems wieder einfließt, sind Regelungen. Als einfaches Beispiel sei bei einem Heizkörper das einfache Stellventil genannt, welches eine Temperatursteuerung darstellt. Dazu im Gegensatz ist der Heizungsthermostat eine Temperaturregelung.

Alle weiteren Betrachtungen legen ihren Fokus auf Prozesse, die einen begrenzten Ereignis- und Zustandsraum habe. Sollten sich die hier angewandte Methoden bewähren, bleibt zu prüfen, ob das angewandte Verfahren sich auch auf hoch reaktive Systeme abbilden lässt.

### 2.1.1 Mealy und Moore

Mealy und Moore haben die Theorien für transformationale Systeme geprägt. Sie beschreiben Automaten, die auf einfache Weise Eingaben verarbeiten. Sie werden Hauptsächlich für den Compilerbau angewandt und sollen helfen ein Grundverständniss für formale Sprachen zu verstehen. Unterschiede zwischen Mealy und Moore Automaten sind, dass ein Moore-Automat nur in den Zuständen mit Ausgaben reagiert. Eine Mealy Automat macht Ausgaben, wenn ein Zustandswechsel stattfindet.

### 2.1.2 Harel

Harel hat die Notation der hierarchischen Zustandsdiagramme entwickelt. Im wesentlichen hat er die Ideen von Mealy und Moore vereint, indem er Aktionen im Zustand sowie bei einem Zustandsübergang zulässt. Er spezifiziert, Zustände die ineinander geschachtelt werden können. Daraus ergibt sich eine hierarchische Struktur von Automaten und Subautomaten. Er hat die Semantik orthogonaler Subautomaten definiert und Wächter nach dem Beispiel von 'guarded Commands' nach Hoare eingeführt. Harel-Automaten haben eine Historie, die es ermöglicht einen Subautomaten zu verlassen und bei späterem Wiedereintritt in den Subautomat, statt des Startzustandes, den Zustand anzunehmen, den er beim Verlassen hatte.

## 2.2 Modellierung reaktiver Systeme

Das Erstellen von Modellen ist das Erfassen von Ideen und informellen Beschreibungen, um konkrete Beschreibungen zu definieren. Nach Harel (1989) eignen sich drei Diagrammtypen für die Modellierung technischer Prozesse:

### 2.2.1 Die strukturelle Sicht (Moduldiagramme)

Modul - oder Collaborationsdiagramme dienen zur physikalischen Beschreibung eines Systems. Mit diesen Diagrammen werden nicht nur Softwarekomponenten modelliert, sondern auch die beteiligte Hardware oder in den Prozess eingebunden Akteure. Gegenstand der Modellierung sind die Nachrichtenflüsse in dem System. Vereinfacht wird also dargestellt, in welche Subsysteme sich das Gesamtsystem unterteilt, und wie diese miteinander kommunizieren und interagieren. Hohe Anforderungen an Eingenimplementation, da die physikalischen Schnittstellen oft Plattformabhängig sind. Der Grad an Codegenerierung ist gering.

### 2.2.2 Die Zustandssicht (Zustandsdiagramme)

Mit Zustandsdiagrammen lassen sich die reaktive Kontrollstrukturen von reaktiven Systemen beschreiben. Es beschreibt das Systemverhalten in Abhängigkeit zur Zeit mit der Abhängig-

keit der Dynamik der Aktivitäten, also dem Wechsel vom Zustand eines Automaten durch ein bestimmtes Ereignis. Das Zustandsdiagramm (Statechart) ist der Diagrammtyp, der sich am besten dazu eignet Software zu generieren. Über diesen Diagrammtyp können die Modelle für AIRA/KNT realisiert werden, die in 2.3.1 beschrieben werden.

### 2.2.3 Die funktionale Sicht (Aktivitätsdiagramme)

Das Aktivitätsdiagramme beschreibt die Systemfunktionen, dessen Prozesse und die beteiligten Objekte (bzw. Automaten). Außerdem werden in dieser Ansicht die Ein- und Ausgaben des Systems integriert. Diese Diagrammtypen stehen in Analogie mit AIRA/ANW die ebenfalls in 2.3.1 beschrieben werden.

## 2.3 AIRA

AIRA steht für **A**utomaten **i**n **R**eaktiven **A**nwendungen und ist eine semantische Spezifikation von Automatenmodellen, für die es zur Zeit eine textuelle Beschreibungssprache gibt. In ihr laufen die Ideen von Harel und Douglass (1999) zusammen. Die Spezifikation der Sprache ist nachzulesen in Kaltenhäuser (2001). In Diplomarbeiten und Projekten ist ein Compiler und eine Laufzeitumgebungen für AIRA entstanden.

### 2.3.1 AIRA-Reports

Die AIRA Reports beschreiben die Syntax und Semantik der Sprache AIRA. AIRA besteht aus drei Sprachreports.

- AIRA/DAT. Die Sprachkomponente, über die sich die Datentypen einer Anwendung spezifizieren lassen. Diese Sprachkomponente soll in den Report AIRA/KNT und AIRA/ANW einfließen.
- AIRA/KNT ist die Komponente, die das reaktive Verhalten eines Prozesses beschreibt. Mit ihr werden hierarchische und zeitgesteuerte Automatenmodelle beschrieben. AIRA/KNT verwendet für lokale Datenstrukturen die mit AIRA/DAT spezifizierten Datentypen einer Anwendung.
- AIRA/ANW dient zur Beschreibung eines System- bzw. Anwendungsaufbaus komplexer, verteilter Anwendungen. Die einzelnen nebenläufigen Komponenten dieser Anwendungen werden mit AIRA/KNT beschrieben. AIRA/ANW ermöglicht die Spezifikation von Kommunikationskanälen, die zwischen den einzelnen Kontrollerkomponenten aufgebaut werden und die Spezifikation von Nachrichten die über diese Kanäle fließen.

### 2.3.2 AIRA-Compiler

Der AIRA-Compiler prüft AIRA-Code auf lexikalisch und syntaktische Korrektheit, bevor ein Automatenmodell aufgebaut wird, mit dessen Hilfe sich die semantische Korrektheit des in AIRA beschriebenen Automaten prüfen lässt. Werden diese Schritte erfolgreich abgearbeitet, kann über ein Backend Zielcode erzeugt werden. Das Backend ist über ein PlugIn-Mechanismus erweiterbar. Es kann für verschiedene Zielsysteme und Zielsprachen Quellcode erstellen. Man kann an dieser Stelle sagen, das es sich bei AIRA-Code um ein plattformunabhängiges Modell (PIM[1]) einer Anwendung handelt. Der Compiler ist in Eiffel geschrieben. Eiffel setzt das 'Design by Contract' Konzept um. Das Verhalten von Softwaremodulen wird also per Vertrag implementiert. Dies geschieht durch Vor- und Nachbedingungen, die für jede Methoden definiert sein müssen. Diese Tatsache ermöglicht nicht nur das Prüfen von Typsicherheit durch den Compilers, sondern auch die semantische Prüfung von Ein- und Ausgangsparameter. Dieses hohe Maß an Sicherheit ist bei der Realisierung von technischen Prozessen in der Regel unerlässlich.

### 2.3.3 AIRA-Laufzeit

Die Abarbeitung von Automatenmodellen ist durch eine Laufzeitumgebung realisiert. In der Laufzeitumgebung ist das Verhalten von Aktivitäten und Zuständen organisiert. Eine Aktivität steht als synonym für einen Automat und entspricht somit einer Menge von Zuständen. Ein Zustand spiegelt die aktuelle Situation eines technischen Prozesses wieder und kann selbst eine Liste von Subaktivitäten enthalten. Die Laufzeitumgebung ruft rekursiv die Prozedur `reagieren(...)` auf, die als Parameter ein Ereignis übergeben bekommt. Die Laufzeit kennt ihren aktuellen Zustände und versucht das Ereignis zu verarbeiten. Je nachdem, ob und wie der Automat in seinen hierarchischen Ebenen seinen Zustand gewechselt hat wird ein entsprechender Parameter zur Auflösung der Rekursion zurückgegeben. Hat eine hierarchisch niedrigere Ebene das Ereignis verarbeitet, reagiert keine der höheren Hierarchieebenen.

### 2.3.4 AIRA-Laufzeit für QNX

Die QNX-Laufzeitumgebung ist eine konkrete Implementierung der oben beschriebenen Laufzeit. Sie wird im FAUST-Projekt eingesetzt. Als Echtzeitbetriebssystem wird auf dem Fahrzeug QNX eingesetzt. Die Laufzeit ist als Shared-Library realisiert. Entwickelte Automaten müssen also nicht zusammen mit dem Laufzeitcode übersetzt werden. Die Implementierung ist in BsC A. Pröhl (2005) beschrieben.

[1] Wird in 3 erklärt.

# 3 Modellgetriebene Softwareentwicklung

Die OMG (Object Management Group) hat das Ziel, die Standardisierung von Methoden zur Beschreibung von Software voranzutreiben. Neben UML als visuelle Beschreibungssprache von Software sind unter anderem Standards wie CORBA entstanden. Für die Modellgetriebene Softwareentwicklung ist diese Organisation im wesentlichen an der MOF (Meta-Object Facility) beteiligt. MOF ist eine Sprache zur Beschreibung des UML-Modells.

In der Modellgetriebenen Softwareentwicklung geht man davon aus, das es eine klare Trennung zwischen der Logik einer Anwendung und der Ausführungsumgebung gibt. Der Vorteil, der sich daraus ergibt, ermöglicht es das Modell einer Anwendung für eine Vielzahl unterschiedlicher Ausführungsplattformen zu generieren. Erreicht wird dies durch das Transformieren von unterschiedlich abstrakten Modellen. Eine Transformation wird in der Regel von einer abstrakten in eine konkretere Ebene vorgenommen. Hierbei wird zwischen 'Modell nach Code' und 'Modell nach Modell' Transformationen unterschieden. Bei einer 'Modell nach Code' Transformation wird Anwendungscode erstellt, der übersetzt und ausgeführt werden kann. 'Modell nach Modell' Transformation beschreibt das übertragen eines bestehenden Modells in ein anderes. Als Beispiel sei hier eine Transformation von Klassen- nach ER-Diagramm aufgeführt. Die Modelle sind:

- das **Computation Independent Model (CIM)** welches eine umgangssprachliche Beschreibung eines Modells ist. Das CIM hat den höchsten Abstraktionsgrad und beschreibt keine technischen Details. Hierfür werden oft Use-Case- , Interaktion- und Aktivitätsdiagramme verwendet.

- das **Platform Independet Model (PIM)** das den Anwendungsaufbau (Prozessabläufe) beschreibt, aber in seiner Beschreibung noch keine Plattforminformationen enthält. So lässt sich ein PIM-Model durch Anreicherung an Informationen in ein PSM-Model(s.U.) tranformieren, aus welchem dann Code einer Software in Java oder aber in C++ generieren werden kann.

- das **Platform Specific Model (PSM)**. Architekturbeschreibung einer Anwendung. Ein PSM-Modell wird mit Plattformabhängigen Informationen angereichert. So können hier z.B. die Typinformationen der Zielplattform konkretisiert werden. Für Zeichenketten in Java java.lang.String oder in C++ als char*.

- das Codemodell enthält den Code, der zur ausführbaren Anwendung führt. Meist ist dieses Modell noch nicht komplett. Einzelne Funktionen müssen oft noch implementiert werden, da sie aufgrund ihrer Komplexität an die Grenzen der Modellierungssprachen stoßen und somit nicht modelliert werden können.

Modellgetriebene Softwareentwicklung ist seit langer Zeit erklärtes Ziel der Informatik. Schon vor 20 Jahren war es Vision der Informatik aus Zeichnungen, die ein Software-Modell

beschreiben auf 'Knopfdruck' ausführbaren Code zu erstellen. Der MDA-Ansatz ermöglicht es, die Architektur eines Systems gut abzubilden. Alle Objekte und ihre Beziehungen untereinander lassen sich sehr gut abbilden. Leider ist es aber zur Zeit noch nicht möglich komplexe Anforderungen an Methoden und Funktionen zu modellieren. Dadurch schwankt, je nach Komplexität der Anwendung, der Anteil an generierbarem Code zwischen 20% bis 80%.

## 3.1 Meta-Object Facility

Die Object Management Group (OMG) hat die Spezifikationen der Meta-Object Facility (MOF) verabschiedet. MOF nutzt die UML als Modell und deren graphische Notation zur Darstellung von MOF Modellen. Die Meta-Object Facility ist eine abstrakte Beschreibung von Modellen und definiert 4 Schichten (M3 - M0). Der Abstraktionsgrad steigt von M0 nach M3. Eine niedrigere Schicht instanziert die jeweils höhere Modellierungsebene. Siehe Bild 1.

- M3 Ein Satz von Konstrukten zur Erzeugung von Meta-Modellen. Also das Meta-Metamodell. Beispiele: MOF Class, MOF Attribute
- M2 Meta-Modelle, die Instanzen eines MOF-Konstrukts Beispiele: UML-Class, UML-Attribut
- M1 Modell, die Instanzen eines M2-Objektes sind. Beispiele: Beschreibung einer Klasse „Video“ mit den Attributen“Title,...“
- M0 Objekte und Daten, die von einer Anwendung instanziert werden.

## 3.2 Transformation via XML

XMI (XML Metadata Interchange) ist das Datenaustauschformat zwischen Softwareentwicklungsanwendungen. Es basiert auf dem XML-Standard. Durch seinen Aufbau ist es sehr flexibel und herstellerneutral. Die Meta-Object Facility (MOF) bietet eine Sammlung von Schnittstellen für die Verwaltung von Meta-Objekte an (siehe 3.1). Es fehlte jedoch eine textuelle Beschreibungsprache zur vollständigen Beschreibung der, mit UML nur graphisch formulierbaren, Modelle. Diese Lücke schließt das XMI Format. Der Standard setzt auf der Metasprache XML des World Wide Web Consortiums auf und trägt den Namen XML Metadata Interchange. Im wesentlichen stellt XMI einen standardisierten Formalismus zur Generierung von XML-Vokabularen dar. Hierbei können sowohl Document Type Definitions als auch – ab XMI Version 2 – XML Schemata erzeugt werden. Die Generierung von DTD und Schemas wird hierbei von der Level 2 Schicht übernommen und kann somit für jeden Diagrammentyp unterschiedlich geartet sein. Bereits im Standard enthalten sind zwei mit diesen

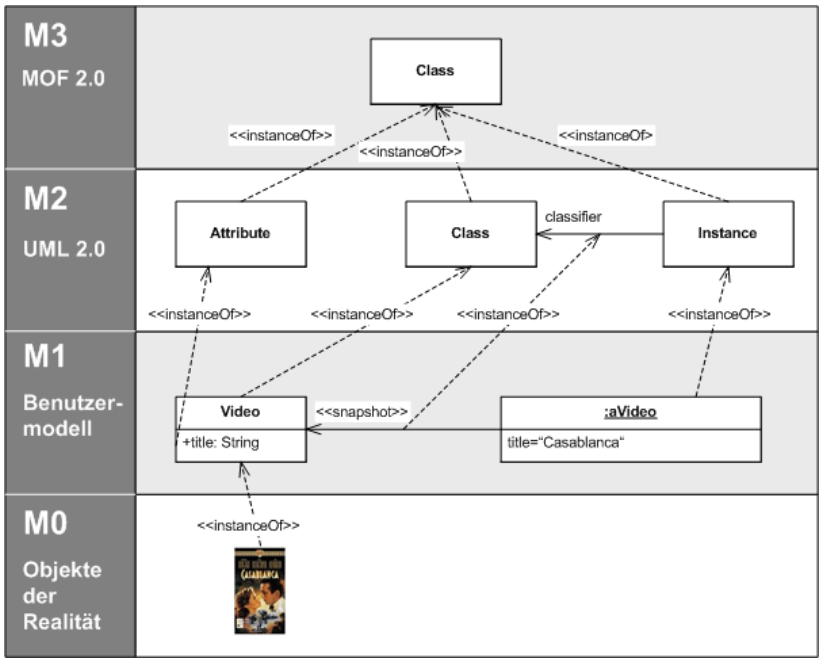


Abbildung 1: Hierarchie der Metamodellierung

Generierungsprinzipien gewonnene Vokabulare. Zum einen das wohl bekannteste zur Darstellung UML-basierter Modelle, zum anderen eine Vokabulardefinition zur Codierung von Metamodellen gemäß dem MOF-Standard.

## 3.3 Eclipse Modelling Framework (EMF)

Das Eclipse Modelling Framework ist ein Plug-In für eclipse[2] .Das EMF ermöglicht, wie MOF, das Beschreiben von Meta-Modellen. Es bietet eine Laufzeitumgebung für die erstellten Modelle. Die Laufzeitumgebung bietet ein Nachrichtensystem, das für die Modelle genutzt werden kann. Somit können Änderungen am Modell an die Umgebung propagiert werden. Desweiteren lassen sich Editoren für das Modell generieren. Die Modelle lassen sich in XMI serialisieren. Der gravierende Unterschied zu MOF ist, dass die Modelle nicht 'gezeichnet' werden müssen. Ein Modell kann auch aus bestehendem Java-Code, einem XMI-Datensatz oder UML-Diagrammen erzeugt werden. Da EMF OpenSource ist, kann es sehr flexibel angepasst werden, um die eigenen Anforderungen umzusetzen.

[2]eclipse ist eine Java-Basierte Entwicklungsumgebung, die sich durch ihre Erweiterbarkeit auszeichnet.

### 3.4 Generierung von Code

Die Generierung von Quellcode basiert auf der Anwendung von Transformationsregeln auf XMI-Daten. Codegenerierung ist in Entwicklungswerzeugen eine Standardfunktion. Nachteil von kommerziellen Produkten ist, das die Möglichkeiten zur Modifizierung der Transformationsregeln eingeschränkt, oder gar nicht möglich ist. Muss eine neue Zielsprache umgesetzt werden, hat man nur die Möglichkeit externe Codegeneratoren zu Hilfe zu nehmen. 'AndroMDA' ist ein Werkzeug zur Codegenerierung. Es ist um Cartridges erweiterbar, die das Definieren von eigenen Transformationsregeln von XMI-Daten nach Quellcode zulassen. Cartridges sind Ausgabemodule für die XMI-Parser. Der Anteil an generierbarem Code ist nach meiner Einschätzung sehr hoch, da AIRA die Funktionalität von reaktiven Systemen seht detailliert beschreibt.

## 4 Risiken

Die Modellgetriebene Softwareentwicklung biete Risiken. Es muß sichergestellt werden, das sich Transformationsregeln definieren lassen. Erst diese ermöglichen das Transformieren von Modellen in Code oder andere Modelle. Ein weiterer Kritikpunkt des MDA-Ansatzes ist, dass das Einbringen nachträglicher Änderungen kaum unterstützt wird. Da im Zuge der Codegenerierung kein fertiges Programm, sondern lediglich eine Code-Vorlage zur manuellen Weiterbearbeitung erzeugt wird, müssen spätere Änderungen entweder vollständig manuell eingepflegt werden, oder die eingebrachte Anpassungen der manuellen Fertigstellung geht bei einer erneuten Codegenerierung verloren. Viele Modellierungswerkzeuge enthalten Ansätze, um diese Probleme zu lösen. Auch Wirtschaftlich ist der MDA-Ansatz nicht immer zu empfehlen. Bei kleinen, wenig komplexen Anwendungen, ist der Aufwand der abstrakten Definition von Objekten und Prozessen zeitaufwändiger als die Anwendung zu erstellen.

## 5 Zusammenfassung

Nach Harel bieten sich State- und Aktivitycharts als Basis für die Modellierung an. Der Zusammenhang technischer Prozesse, des Nachrichtenfluß zwischen diesen und die Zustände, in denen sich die Prozesse befinden können, lassen sich durch diese Diagrammtypen abbilden. AIRA ist ein PIM[3] für Automatenmodelle. Es enthält keine plattformabhängigen Informationen. Somit lassen sich aus AIRA heraus Programme für diverse Zielplattformen erstellen. Eine Laufzeitumgebung bietet für erstellte Anwendung eine Ausführungsumgebung, die das logische Verhalten der Automaten implementiert. Viele Entwicklungswerkzeuge ermöglichen es, die erstellten UML-Diagramme im XMI-Format zu speichern. XMI ist die textu-

[3] Siehe Abschnitt 3

elle Beschreibungssprache für UML-Diagramme und die Basis für Transformation in andere Modelle. Für die Transformation gibt es Werkzeuge, die durch Cartridges (ein Plug-In Mechanismus) erweitert werden können. Diese Flexibilität kann genutzt werden um unterschiedliche Zielsprachen für die modellierten Automaten zu erzeugen. Reicht der Umfang von UML nicht aus, die Anforderungen an das Modell zu beschreiben, kann durch UML-Profile das Metamodell um Facetten angereichert werden. Ein weiterer Ansatz ist die Verwendung des EclipseModellingFramework, das durch Quelloffenheit ein hohes Maß an Flexibilität aufweist, ohne die standardisierten Austauschformate (XMI) zu missachten. Die Möglichkeiten schon zur Entwicklungszeit semantische Fehler am Modell aufzudecken ermöglicht ein effizienteres Entwickeln von Anwendungen.

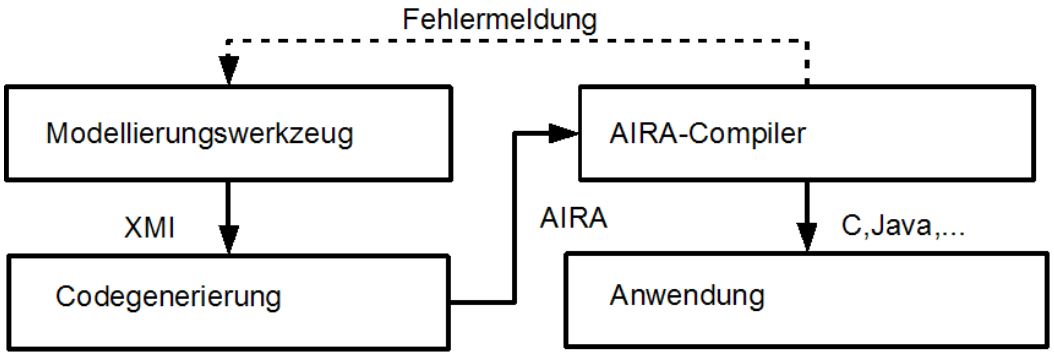


Abbildung 2: Ablauf ein modellgetriebenen Entwicklung

# 6 Ausblicke

Für Werkzeuge zur Modellierung reaktiver Systeme, inklusive der Generierung von Code, gibt es am Markt nur wenige Anbieter. Diese Werkzeuge zeichnen sich durch ihren hohen Preis aus. Die Ideen, die mit dieser Ausarbeitung und der daraus resultierenden Masterarbeit umgesetzt werden sollen, sind eine einfache, günstige und unkompliziert zu bedienende Oberfläche, die das standardisierte Austauschformat XMI unterstützen. Es soll eine Umgebung implementiert werden, die aus einem Statechart via XMI und einer Codegenerierungssoftware AIRA-Quellcode erstellt. Der AIRA-Compiler prüft die syntaktische und semantische Korrektheit des Codes und meldet aufgetretene Fehler zurück an das Modellierungswerkzeug und damit an den Anwendungsentwickler. Ist die Prüfung erfolgreich, wird über das Backend des AIRA-Compilers ein Anwendungscode generiert. Bild 2 beschreibt diesen Ablauf. Die Basis bilden die Automatenbeschreibungssprache AIRA, sowie die in 2.3.3 beschriebene Laufzeitumgebung die speziell für AIRA geschrieben wurde. Somit ist die Zielsystemsprache für ausführbare Anwendungen C und das Zielsystem die Geme-Rechner die im TI-Labor der Informatik auf denen als Betriebssystem QNX eingesetzt wird. Die Konzepte, die für die

Lösung angedacht sind, basieren auf Objektorientierten Ansätzen. AIRA selbst sieht Objektorientierung noch nicht vor. Es bleibt zu klären, ob das Testen der Semantik von dem bereits implementierten AIRA-Compiler übernommen wird, oder ob es effizienter ist, diesen Schritt bei der Codegenerierung des AIRA-Codes abzuarbeiten. Ein weiterer Aspekt, der sich aus der Aufgabe ergeben könnte, ist das Modellchecking. Dazu müsste ein Codegenerator geschrieben werden, der in der Lage ist, AIRA-Quellcode in eine Sprache zu transformieren, die von einer entsprechenden Software (FDR2) analysiert werden kann. Somit könnten erstellte Automaten auf ihre Lauffähigkeit getestet werden. Dies würde einen hohen Grad an Qualität der entwickelten Software gewährleisten. Die Modellierungsumgebung könnte nicht nur für das FAUST-Projekt verwendet werden, sondern auch im Prozesslenkungs-Praktikum des TI-Studiengangs eingesetzt werden.

# Abbildungsverzeichnis



# Literatur

[Brooks 1985] BROOKS, Rodney: A robust layerd control system for a mobile robot, 1985

[BsC A. Pröhl 2005] BSC A. PRÖHL, Dipl.-Ing. tech. Inf. Oliver N.: AIRA-Runtime, 2005

[Douglass 1999] DOUGLASS, Bruce P.: *Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns*. Addison-Wesley, 1999 (Object Technology Series)

[Harel 1989] HAREL, David: Modeling reaktive Systems with Statecharts. In: *The Statemate Approach*, 1989

[Kaltenhäuser 2001] KALTENHÄUSER, Prof. Dr. H.: AIRA Reports, 2001

[OMG ] OMG: Unified Modeling Language: Superstructure version 2.0, URL `www.omg.org`