# Ein Ansatz zur Entwicklung von Modellierungswerkzeugen für die softwaretechnische Lehre

**Dissertation**

zur Erlangung des Grades eines

D o k t o r s  d e r  N a t u r w i s s e n s c h a f t e n

der Universität Dortmund
am Fachbereich Informatik
von

**Jörg Pleumann**

Dortmund
2007

Tag der mündlichen Prüfung: 12. April 2007
Dekan: Prof. Dr. Peter Buchholz
Gutachter: Prof. Dr. Ernst-Erich Doberkat,
Prof. Dr.-Ing. Gernot Fink

## Zusammenfassung

Beim Lehren und Lernen graphischer Modellierungssprachen wie der Unified Modeling Language (UML) ist eine Unterstützung durch entsprechende Werkzeuge sinnvoll und wünschenswert – nicht zuletzt, weil es die Lernenden frühzeitig an einen Umgang mit Werkzeugen gewöhnt, wie er im professionellen Umfeld Standard ist. Die meisten existierenden Modellierungswerkzeuge (z.B. IBM Rational Rose oder Borland Together) richten sich jedoch ausschließlich an die Zielgruppe der professionellen Software-Entwickler und lassen einen Einsatz in der Lehre völlig außer Acht. Das Ergebnis sind ausgesprochen schwergewichtige Produkte (im Sinne von Funktionalitätsumfang, benötigtem Hauptspeicher und CPU-Leistung), deren reichhaltige Möglichkeiten zwar den Bedürfnissen eines professionellen Umfelds entgegenkommen, aber weit über das hinausgehen, was in einem Praktikum oder einer Übungsgruppe benötigt wird oder angemessen ist. Zu viele Funktionen lenken die Studierenden vom eigentlichen Lehrstoff ab und führen dazu, dass mehr Zeit in die Erlernung der Verwendung des Werkzeugs als in die eigentlich zu vermittelnde Modellierungssprache investiert wird. Kommen mehrere Modellierungssprachen – und damit mehrere Werkzeuge – zum Einsatz, multipliziert sich dieser Aufwand, da die einzelnen Werkzeuge einander meist nicht ähneln. Bei einer großen Anzahl von Studierenden können auch Lizenzkosten schnell zu einem Problem werden.

Um diesen Schwierigkeiten zu begegnen, wurde im Rahmen der vorliegenden Arbeit eine Familie von graphischen Modellierungswerkzeugen auf der Basis eines speziellen Meta-CASE-Frameworks ausschließlich für die Lehre entwickelt. Diese Familie umfasst derzeit verschiedene Vertreter für strukturelle und dynamische Anteile der UML, Petrinetze sowie Prozessmodellierung und -begleitung auf der Basis des Unified Process. Bei der Planung und Realisierung dieser Werkzeuge wurde bewusst Wert darauf gelegt, nicht mit professionellen Produkten zu konkurrieren, sondern stattdessen leichtgewichtige Werkzeuge zu schaffen, die auf die Kernfunktionalität des Modellierens reduziert sind. Da alle Werkzeuge die gleiche technische Basis besitzen, war es möglich, eine einheitliche Benutzungsschnittstelle zu etablieren, die sich auf notwendige Elemente konzentriert und damit den Einarbeitungsaufwand minimiert. Gleichzeitig wurde didaktisch motivierte Funktionalität in die einzelnen Werkzeuge eingebracht, die in professionellen Produkten nicht zu finden ist. Diese zusätzliche Funktionalität beinhaltet zum Beispiel ein Hypertextsystem zur Integration von Lehrstoff sowie Simulations-, Analyse- und Visualisierungsmöglichkeiten, durch welche die Studierenden beim Lernen der jeweiligen Modellierungssprache unterstützt werden. Einige der Werkzeuge wurden im Rahmen der Lehre eingesetzt und evaluiert. Die Erfahrungen, die bei diesen Einsätzen gewonnen wurden, waren sehr positiv.

## Abstract

With today's software systems becoming more and more complex, teams getting larger, and development itself being distributed across space and time, the importance of a good model of the system under construction is growing. In order to prepare new software engineers for these requirements, it is necessary to teach them during their studies basic modeling concepts as well as concrete modeling languages, the Unfied Modeling Language (UML) surely being one, but not the only one of these. If the size of models used, for example, in assignments approaches that of real-life systems, tool support becomes necessary.

Unfortunately, the industrial modeling tools typically used for that purpose, such as Borland Together or IBM Rational Rose, have significant drawbacks when applied in an educational setting. These drawbacks stem from the fact that industrial tools are rather heavyweight pieces of software, both in terms of their feature set and the hardware required to run them smoothly. As a consequence of the complexity, there is a risk that merely the tool handling is taught instead of the particular modeling language or method. If different tools are used for different notations, this situation becomes even worse, since the students have to be familiar with each of the tools before being able to work with them effectively. Being targeted at professional developers who are assumed to be proficient in modeling, industrial tools usually do not include functionality that supports learning a modeling language. Last, but not least, the price of industrial tools quickly becomes a problem for academic institutions.

As a solution to the aforementioned problem, the author proposes a specialized Meta-CASE approach to building dedicated modeling tools for Software Engineering education. Based on the approach, a product family of modeling tools has been developed. This family currently provides support for modeling structural and dynamical aspects of the UML, for modeling Petri Nets, and for process modeling based on the Unified Process. While designing and implementing the tools, emphasis was placed upon not competing with professional tools. Instead, the tools are restricted to the core functionality of modeling – thus lightweight – and have a clear focus on usability. Since all tools are built upon the same technical foundation in form of a Java framework, it was easy to establish a consistent user interface that minimizes the learning effort for the tools themselves.

Additionally, the tools have been augmented with new functionality that is motivated by didactical considerations and is typically not found in industrial tools. This includes, for instance, a hypertext system that allows linking hypertext pages to model elements and vice versa. It can be used for documenting a tool, a modeling language, or a particular model. Some of the tools also include facilities for simulating, analyzing, or visualizing models, in order to support learning the semantics of a modeling language. These facilities also play an important role in motivating the students. The tools have been and still are used in Software Engineering education with great success. Formative evaluations have been conducted for one representative tool of the family. The results of these evaluations have been very promising.

Meiner Mutter,
für alles...

# Vorwort

Diese Arbeit habe ich während meiner Beschäftigung am Lehrstuhl für Software-Technologie der Universität Dortmund verfasst. Auch wenn ich letztlich als einziger Autor auf dem Titel erscheine, haben viele Personen auf die eine oder andere Weise Anteil daran gehabt und sich damit ein Dankeschön verdient.

Mein Dank gilt zunächst meinem Doktorvater, Prof. Dr. Ernst-Erich Doberkat, für die Betreuung der Dissertation und dafür, dass er im richtigen Moment angefangen hat, „auf die Uhr zu schauen“. Prof. Dr.-Ing. Gernot Fink danke ich für die Übernahme des Zweitgutachtens bei einer ihm unbekannten Arbeit.

Die Werkzeuge wären nicht die Werkzeuge ohne die Beiträge meiner exzellenten Hiwis und Diplomanden Zahir Amiri, Christian Mocek, Robert Petermeier, Stephan Schulte und Sven Wenzel. Danke für Eure tatkräftige Hilfe!

Meinen (nunmehr Ex-) Kollegen Dr. Klaus Alfert, Dr. Stefan Dissmann und Dr. Alexander Fronk und Dr. Doris Schmedding danke ich für – je nach Bedarf – mal aufmunternde und mal strenge Worte, Beiträge und Feedback zu den Werkzeugen sowie das Lesen von Teilen der Arbeit. Jens Schröder danke ich für die produktive, aber leider viel zu kurze Zusammenarbeit. Die nach wie vor spannende Kooperation mit Dr. Stefan Haustein hat mich bis nach San Francisco gebracht. Auch dafür danke!

Dem gesamten Lehrstuhl für Software-Technologie danke ich für sechseinhalb tolle Jahre. Ich werde Euch alle vermissen!

Mülheim a.d. Ruhr, Januar 2007

Jörg Pleumann

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Teil I

# Kontext der Arbeit

# 1 Einleitung und Motivation

Viele naturwissenschaftlich-technische Disziplinen bedienen sich graphischer Modelle, um komplexe Phänomene anschaulich beschreiben zu können. Die Informatik stellt hier keine Ausnahme dar. Insbesondere in der Softwaretechnik existiert ein reichhaltiger Fundus von graphischen Modellierungssprachen, die in verschiedenen Phasen des Software-Entwicklungsprozesses zum Einsatz kommen. Sie bieten zum Beispiel Unterstützung bei der Analyse eines Problems, der konzeptionellen Spezifikation einer Lösung oder der Dokumentation eines implementierten Softwaresystems. Bekannte Vertreter sind Klassendiagramme sowie deren aus der Datenbankwelt stammende Vorfahren, Entity-Relationship-Modelle, Anwendungsfalldiagramme, Zustandsdiagramme oder Petrinetze. Hinzu kommt eine Vielzahl von Bäumen, Graphen oder Flussdiagrammen zur Repräsentation spezifischer Datenstrukturen oder Algorithmen, die auf diesen Datenstrukturen operieren.

Die Vermittlung von Syntax, Semantik und Pragmatik dieser graphischen Modelle sowie der ihnen zugrundeliegenden formalen Sprachen stellt einen wesentlichen Inhalt der Softwaretechnik-Vorlesungen an Universitäten und Fachhochschulen dar. Ähnlich wie bei der Lehre von Programmiersprachen und -fertigkeiten ist aber ein rein instruktiver Zugang über Vorlesungen nicht ausreichend, um eine Modellierungssprache zu erlernen. Vielmehr müssen die Studierenden das in Vorlesungen erworbene Wissen vertiefen und festigen, indem sie Erfahrungen im Umgang mit der Sprache sammeln, diese also zum Lösen praktischer Probleme anwenden. Letzteres geschieht üblicherweise im Rahmen des Übungsbetriebs oder innerhalb von Praktika. In Fällen, in denen die Komplexität der gestellten Aufgabe sich der Größe realer Probleme annähert (was sinnvoll ist, da z.B. die Notwendigkeit von Klassendiagrammen bei trivialen Problemen nur schwer zu erkennen ist), empfiehlt sich die Verwendung von graphischen Modellierungswerkzeugen. Dies entspricht auch dem Status Quo der Informatikausbildung an Universitäten und Fachhochschulen.

Leider ist es aber so, dass die meisten existierenden Modellierungswerkzeuge, z.B. solche zum Computer Aided Software Engineering (CASE) auf der Basis der Unified Modeling Language (UML) [BRJ05], sich im wesentlichen an die Zielgruppe der professionellen Software-Entwickler richten. Das Ergebnis sind äußerst „schwergewichtige“ Werkzeuge (im Sinne etwa von Funktionalität, benötigtem Hauptspeicher und CPU-Leistung), deren reichhaltiger Funktionsumfang zwar den Bedürfnissen eines professionellen Umfelds entgegenkommt, aber weit über das hinausgeht, was in einem Praktikum oder einer Übungsgruppe notwendig oder angemessen ist. Zu viele Funktionen lenken die Studierenden vom eigentlichen Lehrstoff ab und führen dazu, dass mehr Zeit in das Erlernen der

Verwendung des Werkzeugs als in die eigentlich zu vermittelnde Modellierungssprache investiert wird. Bei einer großen Anzahl von Studierenden können auch die Kosten eines professionellen Werkzeugs schnell zu einem Problem werden.

Modellierungswerkzeuge, die speziell an die Bedürfnisse von Vorlesungen, Übungen oder Praktika angepasst sind, sind rar. Prinzipiell müssten diese Werkzeuge zunächst „leichtgewichtig“ in dem Sinne sein, dass ihr Umfang auf exakt jene Kernfunktionalität beschränkt ist, die für das grundlegende Arbeiten mit einer spezifischen Modellierungssprache notwendig ist. Gleichzeitig ist es wünschenswert, dass die Werkzeuge zusätzliche Funktionalität enthalten, die bei der Vermittlung von Syntax und Semantik der Sprache helfen können. Im einfachsten Fall kann das (hyper-) textueller Lehrstoff sein, der geeignet in das Werkzeug integriert wird und die Syntax der Modellierungssprache erläutert. Fortgeschrittenere Funktionalität kann helfen, die Semantik einer Sprache besser zu vermitteln, etwa indem diese durch Simulationen oder Analysen „erfahrbar“ gemacht wird. Die resultierenden Werkzeuge sind dann Hybride aus CASE-Werkzeugen und Lernumgebungen.

Für die Programmierausbildung existieren derartige Werkzeuge bereits. Die Programmierumgebung BlueJ [KQPR03] ist ein gutes Beispiel für ein bewusst eingeschränktes, aber dennoch mächtiges Werkzeug zur Vermittlung grundlegender Kenntnisse im Bereich objektorientierter Programmierung mit Java. Neben der geforderten Leichtgewichtigkeit bietet BlueJ didaktisch motivierte Funktionalität zur Unterstützung der Vermittlung von Java: Die Instanzebene eines Programms wird visualisiert und für Inspektionen oder gezielte Methodenaufrufe verfügbar gemacht. Damit kann die Wirkung jedes einzelnen Java-Sprachkonstrukts experimentell erfahren werden. Andere Programmierumgebungen wie Dr. Java [ACS02] erzielen eine vergleichbare Wirkung, indem sie für die eigentlich compiler-basierte Sprache Java einen Interpreter zur Verfügung stellen, der den Studierenden ein unkompliziertes interaktives Experimentieren mit der Sprache ermöglicht.

Im Bereich der Modellierungswerkzeuge existieren bislang keine vergleichbaren Arbeiten. Für verbreitete Modellierungssprachen wie Petrinetze oder UML-Klassendiagramme findet man zwar eine Vielzahl von frei verfügbaren, durchaus leichtgewichtigen Werkzeugen. Diese sind in der Lehre einsetzbar und werden auch oft dort genutzt, bieten aber keinerlei didaktisch motivierte Funktionalität, die über das reine Modellieren hinausgeht. Zudem sind diese Werkzeuge meist Unikate, die von verschiedenen Urhebern stammen und jeweils von Grund auf entwickelt wurden. Aus Sicht der Entwickler hat dies redundanten Implementierungs- und Wartungsaufwand zur Folge. Für potenzielle Benutzer innerhalb der Lehre ergibt sich daraus eine jeweils unterschiedliche Benutzungschnittstelle und Handhabung des Werkzeugs, die – gerade bei der Verwendung mehrerer Werkzeuge innerhalb einer Veranstaltung – in jedem Fall eine erneute Einstiegshürde darstellt, die es zu überwinden gilt, bevor mit dem eigentlichen Lehrinhalt begonnen werden kann.

Das Ziel der vorliegenden Arbeit ist die Konzeption, prototypische Umsetzung und Evaluation von graphischen Modellierungswerkzeugen für die softwaretechnische Lehre. Die entwickelten Konzepte sollen jedoch nicht auf diesen Bereich beschränkt sein, sondern sich prinzipiell auch auf andere Disziplinen übertragen lassen, in denen zweidimensionale, diagrammatische Modelle Verwendung finden. Das Vorgehen innerhalb der Arbeit

lässt sich grob in die folgenden vier Schritte aufteilen, die auch gleichzeitig eine erste Gliederung der Arbeit vorgeben:

- **Teil I** beschreibt den Kontext und die Aufgabenstellung der Arbeit. Zunächst werden einige Grundbegriffe aus dem Bereich der Modellierung und der entsprechenden Werkzeuge geklärt. Anschließend werden die Defizite existierender Werkzeuge herausgearbeitet. Ergebnis dieses ersten Teils ist ein Anforderungskatalog für lehretaugliche Modellierungswerkzeuge, der die Grundlage der weiteren Arbeit darstellt und gleichzeitig deren Ziel definiert.
- **Teil II** der Arbeit entwickelt das Lightweight Modeling Framework (LiMo), ein Meta-CASE-Framework, das die gemeinsame Grundfunktionalität für eine Vielzahl von leichtgewichtigen Modellierungswerkzeugen verfügbar macht. Das Framework integriert bereits didaktisch motivierte Funktionalität, die unabhängig von konkreten Modellierungssprachen realisiert werden kann. Dazu zählt etwa ein Hypertext-System, das die Verknüpfung von Modellen mit Annotationen ermöglicht.
- **Teil III** der Arbeit stellt eine Familie von konkreten Werkzeugen vor, die auf der Basis des Frameworks entstanden sind. Die Modellierungsmöglichkeiten der Werkzeuge umfassen die Struktur und das Verhalten eines Softwaresystems, den Entwicklungsprozess sowie Data-Mining-Anwendungen. Die Werkzeuge decken damit ein weites Spektrum graphischer Sprachen ab. Anhand einiger Werkzeuge wird unter anderem demonstriert, wie Simulationen und Analysen zur Verdeutlichung der Semantik einer Sprache beitragen können.
- **Teil IV** der Arbeit dokumentiert zwei umfangreiche evaluierte Einsätze eines der Werkzeuge, die am Lehrstuhl für Software-Technologie der Universität Dortmund im Lehrbetrieb durchgeführt wurden. Die Evaluationen belegen die Tragfähigkeit des vorgestellten Ansatzes.

Im Anhang der Arbeit findet sich zusätzliches Material, das zwar relevant ist, aber den Lesefluss der einzelnen Kapitel unnötig gestört hätte. Dazu zählen neben technischen Details insbesondere die Aufgabenzettel, die Fragebögen und die kompletten Resultate der beiden Evaluationen.

Einige Ergebnisse der Arbeit wurden bereits an anderer Stelle publiziert [WPD01, Ple03, APS03, APS04a, APS04b, Ple04, PS04, FP04, FP05b, FP05a, DEH+05, FP06]. Die Veröffentlichung [Ple04] hat dabei den Preis für den besten Beitrag der Deutschen e-Learning Fachtagung Informatik (DeLFI) 2004 gewonnen. Das Werkzeug DAVE (siehe Abschnitt 12) wurde zudem als erstes akademisches Projekt mit dem Qualitätssiegel E-Learning (QSeL) [Paw05] ausgezeichnet.

**Danksagung:** Teile der Arbeit sind im Rahmen des BMBF-Verbundprojektes Multimedia in der SoftwareTechnik (MuSofT) entstanden [DEH+05]. MuSofT wurde gefördert durch das Bundesministerium für Bildung und Forschung unter den Kennzeichen 08NM098A und 01NM304A.

# 2 Modellierung

Modellierung ist eine bewährte Vorgehensweise in den traditionellen Ingenieurwissenschaften. Disziplinen wie die Architektur, der Maschinenbau oder die Elektrotechnik machen Gebrauch von Modellen, um die technischen Systeme, die in ihrem Rahmen erschaffen werden, zu planen, zu untersuchen und zu dokumentieren. Mit der stetig wachsenden Komplexität softwarebasierter Systeme und der zunehmenden räumlichen und zeitlichen Verteilung der Entwicklung hat sich die Modellierung auch in der Informatik als zentrales Hilfsmittel etabliert. Die Softwaretechnik hat dazu in den 70er und 80er Jahren eine Vielzahl von Modellierungsansätzen hervorgebracht, aus denen in den 90er Jahren mit der Unified Modeling Language (UML) eine standardisierte objektorientierte Sprache zur Modellierung von Softwaresystemen hervorgegangen ist [BRJ05], die durch eine reichhaltige Werkzeuglandschaft unterstützt wird. Während Modelle bis vor wenigen Jahren als reine Zwischenschritte auf dem Weg zum Quellcode eines Systems angesehen wurden, nehmen sie in neueren Ansätzen wie der Model-Driven Architecture (MDA) selbst die Rolle von zentralen Artefakten im Entwicklungsprozess ein, aus denen weitere Artefakte durch Transformation entstehen [OMG03a, KWB03].

Die folgenden Abschnitte arbeiten die wesentlichen Grundbegriffe auf dem Gebiet der Modellierung auf, beleuchten den Aufbau von Modellierungssprachen und stellen einige der für die Softwaretechnik relevanten Sprachen vor. Das Kapitel schließt mit einer Betrachtung der Modellierung aus Sicht der Lehre.

## 2.1 Modelle und Instanzen

Ein Modell ist eine vereinfachte Darstellung einer existierenden oder geplanten Realität. In der Softwaretechnik beinhaltet diese Realität meist ein softwarebasiertes System unter Einfluss dessen unmittelbaren Umfeldes. Das Modell gibt nur ausgewählte Aspekte des Systems wieder und abstrahiert von anderen. Modelle sind also stets mit einem Informationsverlust gegenüber dem eigentlichen System behaftet, der aber gewünscht ist, um die Komplexität des Systems beherrschen zu können.

Die Beziehung zwischen einem Modell und einem System, das die Anforderungen dieses Modells erfüllt, wird in der Literatur je nach Kontext unterschiedlich bezeichnet und interpretiert. Das System wird zum Beispiel als Repräsentant des Modells gesehen oder als konform zu diesem Modell bezeichnet [Béz04]. Im Rahmen dieser Arbeit soll, in Anlehnung an die Terminologie der objektorientierten Softwareentwicklung (siehe z.B.

[Mey97]), das System als Instanz des Modells betrachtet werden. Im allgemeinen Fall existiert mehr als eine Instanz eines Modells. Enthält das Modell Widersprüche, dann ist die Menge seiner Instanzen leer.

Es sei angemerkt, dass sich der Modellbegriff in der Softwaretechnik und anderen Ingenieurwissenschaften von dem in der Mathematik gebräuchlichen unterscheidet. Während ein Modell in der Mathematik eine konkrete Ausprägung einer allgemeineren Theorie ist, wird ein Modell in der Softwaretechnik als das allgemeinere Konstrukt betrachtet, das eine Menge von möglichen Ausprägungen vorgibt. Der mathematische Modellbegriff bezeichnet ein Modell im Sinne der Logik, der softwaretechnische eines im Sinne der Spezifikation [Bau96]. Was die Mathematik ein Modell nennt, hat aus der Sicht der Informatik also den Charakter einer Instanz.

## 2.2 Sichten und Architekturen

Modelle – speziell solche komplexerer Software-Systeme – sind üblicherweise nicht monolithisch. Stattdessen setzen sie sich meist aus mehreren komplementären Teilmodellen zusammen, die ausgewählte Aspekte betonen und von anderen abstrahieren. Auch hier trägt wieder die Analogie zu den Ingenieurwissenschaften: Die Baupläne für ein Haus bestehen aus Teilen, welche die unterschiedliche Interessen von Maurern, Installateuren, Elektrikern etc. berücksichtigen. Jedes der Teilmodelle stellt eine Sicht auf das Gesamtmodell dar. Die Summe der Sichten wird in der Literatur auch als Architektur des Systems bezeichnet [PW92, SG95].

Bei einem Software-System werden in den meisten Fällen Sichten für funktionale, strukturelle und dynamische Details des Systems benötigt, es existieren jedoch auch andere, teilweise orthogonale Ansätze: Die 4+1-Architektur [Kru95] geht von Sichten für Entwurf, Implementierung, Verteilung und Nebenläufigkeit aus, die mit einer Anwendungsfallsicht abgeglichen werden. Die MDA [OMG03a, KWB03] verwendet eine externe, eine plattformunabhängige und eine plattformabhängige Sicht auf das System. Der IEEE-Standard 1471 für Software-Architekturen [MEH01] sieht ebenfalls die Existenz verschiedener Sichten auf das zu entwickelnde System vor, überlässt dem Entwickler aber deren Auswahl. Die einzelnen Sichten müssen in jedem Fall hinreichend detailliert sein, um bei der Entwicklung helfen zu können, aber gleichzeitig widerspruchsfrei, damit das System überhaupt realisierbar ist.

## 2.3 Zweck der Modellierung

Ein Modell wird stets zu einem festgelegten Zweck erstellt: Es beantwortet eine begrenzte Menge von Fragen über das System auf die gleiche Weise wie es das System selbst tun würde [Béz04]. In der Softwaretechnik sind Modelle an verschiedenen Stellen des Entwicklungsprozesses hilfreich [Bau96, BRJ05]:

- **Analyse** – Die Eigenschaften eines Systems können auf der Basis des Modells untersucht werden, bevor das System tatsächlich realisiert wird.
- **Entwurf** – Ein Modell kann Struktur und Verhalten eines Systems spezifizieren und als Blaupause für dessen Implementierung dienen.
- **Test** – Ein System kann getestet werden, indem sein Verhalten mit dem des Modells verglichen wird. Das Modell dient dabei als Testorakel.
- **Dokumentation** – Ein Modell kann zur Dokumentation eines existierenden Systems genutzt werden und damit dessen Wartbarkeit und Erweiterbarkeit begünstigen.

Hinzu kommt der nicht zu unterschätzende Wert des Gewinns an Erkenntnis über das zu entwickelnde System, der sich bei dessen Modellierung einstellt, sowie die Möglichkeit der Nutzung eines Modells als gemeinsames Kommunikationsmedium aller Beteiligten eines Projekts.

## 2.4 Modellierungssprachen

Zur Repräsentation eines Modells und seiner Sichten wird eine Notation benötigt. Diese kann textueller Natur sein, wie etwa im Fall von Z [Spi92], CSP [Hoa78] oder algebraischen Spezifikationen [Wir94]. In vielen Fällen wird aber aufgrund der intuitiveren Zugänglichkeit eine graphische Repräsentation bevorzugt. Da die wesentlichen Informationen, die bei der Modellierung von Software-Systemen anfallen, qualitativer Natur sind (im Unterschied etwa zu sozial- oder naturwissenschaftlichen Modellen), reicht hier üblicherweise eine diagrammatische Repräsentation durch spezielle Formen von Graphen aus [Har88]. Diese machen Gebrauch von Typisierung, Beschriftung und gegebenenfalls eigenen Darstellungsweisen für die verschiedene Typen von Knoten und Kanten.

In der Literatur werden diagrammatische Notationen auch als visuelle Notationen oder visuelle Sprachen bezeichnet [MMW98], um den nicht-textuellen Charakter hervorzuheben. Harel verwendet zudem den Begriff der topologisch-visuellen Formalismen [Har88], um darauf hinzuweisen, dass die Diagramme visuelle Darstellungen relationaler und mengentheoretischer Phänomene enthalten und außerdem meist eine formale Basis besitzen. Die vorliegende Arbeit verwendet im Weiteren der Einfachheit halber den Begriff der Modellierungssprache und meint damit eine graphische Notation mit den zuvor erwähnten Eigenschaften.

## 2.5 Beispiele für Modellierungssprachen

Die zentrale graphische Modellierungssprache der Softwaretechnik ist gegenwärtig die UML, eine auf objektorientierten Prinzipien basierende Sprache zum Spezifizieren, Visualisieren, Konstruieren und Dokumentieren von Softwaresystemen [BRJ05]. Die Ge-

nealogie der Sprache ist in der einschlägigen Literatur hinreichend beschrieben, so dass an dieser Stelle nur erwähnt werden soll, dass die UML als Synthese der Object Modeling Technique (OMT) [RBP+91], der Booch-Methode [Boo94] und des Object Oriented Software Engineering (OOSE) [JCJv92] entstanden ist. Zahlreiche weitere Modellierungsansätze haben über die Jahre unmittelbar Eingang in die UML gefunden oder deren Konzepte und Notationen beeinflusst. Nicht übernommen wurde von den drei direkten Vorgängern der UML der Gedanke, ein festes Vorgehensmodell in die Sprache zu integrieren oder mit dieser zu propagieren. Gleichermaßen neutral steht die UML der Wahl von Programmiersprachen, Werkzeugen und Anwendungsgebieten gegenüber. Nicht zuletzt durch diese Neutralität ist sie seit dem Ende der 90er Jahre zur *lingua franca* der Softwaretechnik geworden.

Die UML bietet in der Version 2.0 insgesamt 13 verschiedene Diagrammarten, von denen sechs der Beschreibung der Struktur und weitere sechs der Bescheibung des Verhaltens eines Systems dienen. Eine Diagrammart ist der Modellierung funktionaler Details eines Systems zuzuordnen. Komplettiert wird die UML durch die Object Constraint Language (OCL), eine auf der Prädikatenlogik basierende textuelle Sprache zur Formulierung zusätzlicher Randbedingungen, welche die graphische Notation nicht auszudrücken vermag [WK95].

Neben der UML existieren in der Softwaretechnik zahlreiche weitere Modellierungssprachen, die teils älter als diese sind und/oder in speziellen Bereichen der Softwareentwicklung Anwendung finden. Aufgrund der unüberschaubaren Zahl sollen hier nur einige der bekanntesten Vertreter genannt werden, die auch Teil der meisten Informatik-Curricula sein dürften: Datenflussdiagramme [DeM78, GS79] betonen die Dekomposition eines Systems in datenverarbeitende und datenspeichernde Komponenten sowie den Datenfluss zwischen diesen. Entity-Relationship-Diagramme [Che76] dienen der Spezifikation statischer Datenmodelle. Wenngleich die in der UML enthaltenen Klassendiagramme als Obermenge von ER-Diagrammen angesehen werden können, besitzen letztere eine starke Tradition im Datenbankbereich. Petrinetze [Pet62] und deren modernere Varianten wie gefärbte Netze [Jen96] oder Funsoft-Netze [EG95] dienen oft der Modellierung organisationaler Abläufe in Workflow-Management-Systemen. Speziell im Bereich eingebetteter Systeme finden Statecharts [Har87] Anwendung, eine um Nebenläufigkeit und Hierarchie erweiterte Variante endlicher Automaten, die in leicht modifizierter Form auch Eingang in die UML gefunden hat. Eine relativ junge Sprache, die dem starken Wachstum im Bereich der Serviceorientierten Architektur (SOA) Rechnung trägt, ist die Business Process Engineering Language (BPEL) [BPE06], die ihrerseits Einfluss auf Teile der UML 2.0 genommen hat.

## 2.6 Anatomie einer Modellierungssprache

Wenn eine Modellierungssprache als Kommunikationsmedium oder sogar als Grundlage zur Code-Generierung dienen soll, dann muss sie von allen Beteiligten auf die gleiche

Weise verstanden werden. Dies setzt voraus, dass eine allgemein akzeptierte Definition der Sprache existiert. Die beiden zentralen Elemente einer solchen Sprachdefinition sind Syntax und Semantik (für einen jüngeren Überlick über das Thema siehe [HR04]).

### 2.6.1 Syntax

Die Syntax einer Sprache gibt die Menge der möglichen Sätze an, die sich in der Sprache formulieren lassen. Da diese Menge in den meisten Fällen nicht endlich ist, wird sie nicht unmittelbar aufgezählt, sondern induktiv durch ein Regelsystem beschrieben. Bei textuellen Sprachen geschieht dies mit Hilfe einer Grammatik, die sich aus terminalen und nicht-terminalen Symbolen, Ableitungsregeln und einem Startsymbol zusammensetzt. Je nach erlaubter Komplexität der Ableitungsregeln ergeben sich Sprachen unterschiedlicher Ausdrucksmächtigkeit. Die Chomsky-Hierarchie gliedert diese in vier Stufen (unbeschränkt, kontextsensitiv, kontextfrei und regulär), wobei die kontextfreien Sprachen in der Informatik eine besondere Rolle spielen, da die Syntax praktisch aller textuellen Programmiersprachen auf dieser Basis definiert ist. Ein Parser entscheidet für eine Sprache $L$ und einen Satz $S$, ob $S \in L$ gilt. Im positiven Fall liefert er dabei eine Ableitung von $S$ aus dem Startsymbol der Grammatik, welche für kontextfreie Sprachen die Form eines Baums hat. Da formale textuelle Sprachen aufgrund des Bezugs zum Compilerbau in der Informatik seit langem wohlbekannt sind, ist es naheliegend, diese auch zur Definition visueller Sprachen zu verwenden.

Ein verbreiteter Ansatz hierzu sind Graphgrammatiken (einen Überblick leistet etwa [Roz97]). Diese übertragen die Prinzipien von Chomsky-Grammatiken und Termersetzungssystemen auf Graphen. Die Regeln einer Graphgrammatik werden üblicherweise visuell spezifiziert. Die linke Seite gibt einen zu identifizierenden Teilgraphen an, die rechte Seite dessen Ersetzung bei Anwendung der Regel. Ein Typgraph und die Möglichkeit der Spezifikation negativer Anwendungsbedingungen verfeinern das Regelsystem. Es existieren diverse Varianten von Graphgrammatiken, die sich unter anderem in Details der Regelanwendung unterscheiden [ER97, DKH97, CEH+97, EHK+97, Sch97, Min00]. Zur Beschreibung relevanter Modellierungssprachen sind – im Unterschied zur Situation bei Programmiersprachen – kontextfreie Graphgrammatiken nicht ausreichend, so dass nach [MMW98] die meisten auf Graphgrammatiken beruhenden Ansätze nicht effizient zu parsen sind. Zudem resultiert der gesamte Ansatz in Spezifikations- und Entwicklungsphasen, die teilweise sehr komplex werden können (vergl. [CDP04]).

Eine Alternative zur Syntaxdefiniton auf der Basis von Graphgrammatiken stellt die Metamodellierung (für einen Überblick siehe etwa [Küh06]) dar. Bei dieser wird die abstrakte Syntax einer graphischen Sprache mit den Mitteln eines Modells – meist einer speziellen Art von ER-Diagramm – beschrieben. Metamodellierung besitzt im Vergleich zu Graphgrammatiken den Nachteil der geringeren Ausdrucksmächtigkeit. Dieser Nachteil kann jedoch durch eine Annotation des Metamodells mit einer logikbasierten Sprache (wie zum Beispiel OCL) kompensiert werden. Der große Vorteil der Metamodellierung ist

die Zugänglichkeit des Ansatzes [CDP04]. Wer sich mit Modellierungssprachen auseinandersetzt, wird auch in der Lage sein, eine Sprache mit diesen Mitteln zu beschreiben, da sich Sprache und Metasprache vielfach ähneln. Zudem ist ein Metamodell leicht in eine Klassenstruktur einer objektorientierten Programmiersprache zu überführen.

### 2.6.2 Semantik

Die Semantik einer Sprache weist jedem Satz der Sprache eine Bedeutung zu. Die Definition der Semantik geschieht in zwei Schritten. Zunächst wird ein wohldefinierter semantischer Geltungsbereich ausgewählt. Anschließend wird eine Abbildung definiert, welche jedem Satz der Sprache die entsprechende Bedeutung innerhalb des Geltungsbereichs zuweist. Diese Abbildung kann informell, etwa mit Hilfe von Beispielen, oder formal (im Sinne einer totalen, mathematischen Abbildung) definiert sein. Sie kann zudem axiomatisch, denotationell oder operational sein.

Eine axiomatische Semantik basiert auf der Spezifikation von Vor- und Nachbedingungen für die einzelnen Konstrukte einer Sprache, z.B. mit Hilfe des Hoare-Kalküls [Hoa69]. Der Geltungsbereich einer denotationellen Semantik ist eine formale Spezifikationssprache wie Z [Spi92] oder ein Teilbereich der Mathematik. Im Fall einer operationalen Semantik werden die syntaktischen Elemente der Sprache auf eine ausführbare Sprache, z.B. eine Programmiersprache oder eine Abstract State Machine (ASM) [BS03], abgebildet oder es wird ein Algorithmus zu deren Interpretation angegeben. Die Komplexität der Definition einer Semantik lässt sich leichter beherrschen, wenn diese stückweise angegeben wird und sich an den syntaktischen Konstrukten der Sprache orientiert, wie etwa im Fall der Structural Operational Semantics (SOS) [Plo81].

### 2.6.3 Metamodellierung am Beispiel der UML

Innerhalb der UML wird Metamodellierung zur Definition der abstrakten Syntax verwendet. Die konkrete Syntax wird mit Hilfe von graphischen Beispielen vorgegeben. Das Metamodell, die so genannte UML-Superstruktur, ist Bestandteil der Spezifikation und liefert damit eine präzise Syntaxbeschreibung, die insbesondere für die Entwicklung von Werkzeugen hilfreich ist. Als Sprache zur Konstruktion des Metamodells kommt eine begrenzte Untermenge der UML selbst zum Einsatz: Einfache Klassendiagramme (Klassen, Vererbung, Attribute und Assoziationen) dienen zur Beschreibung der grundlegenden syntaktischen Strukturen. Zusätzliche Randbedingungen werden mit OCL formuliert. Paketdiagramme strukturieren das Metamodell und machen es damit handhabbar.

Vom formalen Standpunkt aus betrachet ist die durch das Metamodell beschrieben Sprache nur dann präzise definiert, wenn auch die Sprache, in der das Metamodell beschrieben ist, präzise definiert ist. Diese Definition wird – ebenfalls auf der Basis von Metamodellierung – durch ein Meta-Metamodell geleistet, die sogenannte UML-Infrastruktur. Abbildung 2.1 zeigt den Zusammenhang zwischen den einzelnen Abstraktionsstufen der

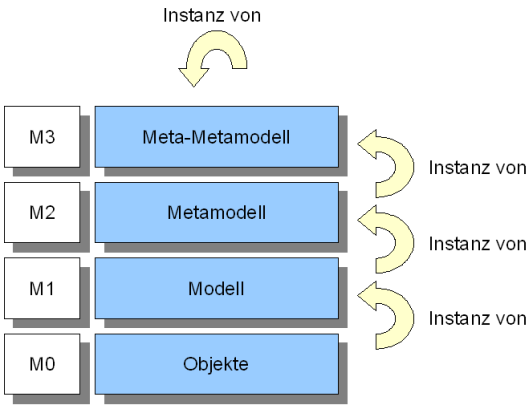



Abbildung 2.1: Spracharchitektur der UML

UML-Syntaxdefinition. Zwischen dem Modell des Benutzers (M1) und dem Metamodell (M2) sowie dem Metamodell (M2) und dem Meta-Metamodell (M3) besteht jeweils eine Instanziierungsbeziehung. Die Sprache des Meta-Metamodells ist durch sich selbst definiert, so dass hier ebenfalls eine Instanziierungsbeziehung besteht. Auf der untersten Stufe (M0) befinden sich konkrete Instanzen von Klassen des Modells (M1), etwa zur Laufzeit eines Programms.

Die Semantik von UML ist – trotz oder gerade aufgrund des Sprachumfangs – nicht formal definiert. Der überwiegende Teil der Semantik ist natürlichsprachlich und auf der Basis von Beispielen beschrieben, wodurch sich ein hinreichend großer Interpretationsspielraum ergibt. Die einzige Ausnahme bilden UML-Zustandsdiagramme, eine der komplexesten Diagrammarten der UML. Zu diesen gibt die Spezifikation eine zwar ebenfalls teilweise natürlichsprachlich beschriebene, aber recht präzise operationale Semantik an. Deren Kernelement ist ein Algorithmus zur Ausführung eines „Schritts“ eines Zustandsdiagramms. Die UML-Semantik profitiert dabei von der Tatsache, dass Semantiken für Zustandsdiagramme (und Statecharts) bereits lange Gegenstand der Forschung sind (vergl. [HN96, vdB94]).

## 2.7 Modellierung aus Sicht der Lehre

Modellierung ist eine elementare Fertigkeit der Software-Entwicklung. Um auf die akademische oder industrielle Praxis der Informatik vorbereitet zu sein, müssen Studierende lernen, ihre Ideen in Modellen auszudrücken und die Modelle anderer zu verstehen. Die UML spielt dabei eine zentrale Rolle, ist aber nicht die einzige relevante Modellierungssprache. Softwaretechnik-Veranstaltungen lehren deshalb zumeist sowohl allgemeine Prinzipien der Modellierung als auch eine oder mehrere konkrete Modellierungssprachen.

Die Vorgehensweise entspricht dabei grob jener der Programmierausbildung: Eine Modellierungssprache wird zunächst instruktiv innerhalb einer Vorlesung vermittelt. Anschließende Übungen oder Praktika vertiefen dieses Wissen mit Hilfe analytischer oder konstruktiver Aufgaben. Dabei gibt es eine Reihe von häufig auftretenden Problemen:

- **Verständnis der Syntax** – Damit überhaupt ein Arbeiten mit einer Modellierungssprache möglich ist, muss zunächst deren Syntax verstanden sein. Bei überschaubaren Sprachen wie Petrinetzen stellt dies meist kein Problem dar. Die Syntax anderer Sprachen, speziell der UML, ist umfangreicher und schwerer zu erlernen.
- **Verständnis der Semantik** – Eine Sprache ohne Semantik ist als Kommunikationsmittel wertlos [HR04]. Folglich setzt das Arbeiten mit einer Modellierungssprache ein präzises Verständnis von deren Semantik voraus. Die komplexe Semantik einiger Sprachen erschwert jedoch sowohl deren Verständnis als auch deren Lehre.
- **Verständnis der Pragmatik** – Modellierung wird von den Studierenden oft als überflüssige formale Hürde auf dem Weg zur Implementierung empfunden. Der Nutzen des Modells als Mittel zum Erkenntnisgewinn oder zur Kommunikation wird übersehen. Darunter leidet vielfach die Motivation zur Modellierung an sich.
- **Verständnis der Abstraktion** – Modelle sind Abstraktionen der Realität. Die Wahl der richtigen Abstraktion für ein Modell benötigt Erfahrung und ist für Studierende oft eine schwierige Aufgabe. Ähnliches gilt für das Erkennen des Zusammenhangs zwischen einem bestehenden Modell und der beschriebenen Realität.

Mehrere dieser Probleme legen es nahe, dass sich die Modelle, mit denen die Studierenden konfrontiert werden, in Umfang und Komplexität realen Aufgabenstellungen zumindestens annähern. So können zum Beispiel Details der Semantik einer Modellierungssprache oft nur erläutert werden, wenn bestimmte Kombinationen von Modellelementen auftreten. Weitaus wichtiger ist eine angemessene Modellgröße jedoch für das Problem von Pragmatik und Motivation. Die Notwendigkeit eines formalen Modells ist für Studierende nicht zu erkennen, wenn das Problem so trivial ist, dass es unmittelbar implementiert werden kann. Für Probleme auf dem Niveau von „Hallo, Welt!“ ist also etwa ein Klassendiagramm didaktisch eher kontraproduktiv.

Es gibt eine Reihe von Gründen, die Modellierungsausbildung durch den Einsatz von Werkzeugen zu unterstützen. Der naheliegendste Grund ist die daraus resultierende Arbeitserleichterung für alle Beteiligten. Modellierung ist ein iterativer Prozess, innerhalb dessen ein initiales Modell schrittweise korrigiert und verfeinert wird. Ab einem gewissen Umfang des Modells (nach obiger Diskussion wünschenswert) ist dies auf dem Papier nur mit großem Aufwand zu erreichen. Studierende können von einem Modellierungswerkzeug als „Zeichenhilfe“ profitieren. Tutoren profitieren von lesbaren Lösungen zu Übungsaufgaben, die gedruckt oder sogar in elektronischer Form abgegeben werden. Zusätzlich ist es wünschenswert, dass ein solches Werkzeug Funktionalität integriert, die didaktisch motiviert ist und zur Lösung der vier zuvor geschilderten Probleme beiträgt. Das nächste Kapitel widmet sich solchen Werkzeugen.

# 3 Werkzeuge

Wenngleich die meisten softwaretechnischen Modellierungssprachen aufgrund ihrer einfachen konkreten Syntax gut zur Arbeit auf dem Papier oder an einer Tafel geeignet sind, kommt der eigentliche Nutzen der Modellierung erst dann zum Tragen, wenn eine Unterstützung durch ein Modellierungswerkzeug gegeben ist. Neben den offensichtlichen Vorteilen der leichteren Bearbeitung, Änderung und Reproduktion eines Modells bietet ein solches Werkzeug z.B. die Möglichkeit, auf syntaktische Fehler hinzuweisen, Analysen durchzuführen oder das Modell in ein anderes Modell oder in Code zu transformieren.

Die Verwendung von Modellierungswerkzeugen wird oft mit dem Begriff des Computer-Aided Software Engineering (CASE) gleichgesetzt, was aber nur teilweise zutreffend ist. Sommerville definiert CASE in [Som04] allgemeiner als „Software zur Unterstützung von Aktivitäten des Software-Entwicklungsprozesses“. Dies schließt zahlreiche weitere technische Werkzeuge wie Compiler und Debugger ein, aber auch eher organisatorische Werkzeuge, etwa solche zur Verwaltung und Verfolgung von Fehlerhinweisen.

Die folgenden Abschnitte beleuchten verschiedene Facetten des CASE-Begriffs. Neben einer Einordnung graphischer Modellierungswerkzeuge in die CASE-Landschaft werden die Funktionalität solcher Werkzeuge und Probleme bei deren Nutzung – insbesondere innerhalb der Lehre – diskutiert. Als Alternative werden didaktische Werkzeuge zur Unterstützung der Programmierausbildung diskutiert. Das Ergebnis des Kapitels ist eine Einschätzung der Defizite existierender Werkzeuge aus Sicht der Lehre sowie eine erste Formulierung von Anforderungen an Werkzeuge, die zur Unterstützung der Modellierungsausbildung geeignet sind.

## 3.1 Klassifikation von CASE-Werkzeugen

Wenn der CASE-Begriff in der oben beschriebenen Allgemeinheit verwendet wird, umfasst er praktisch jedes Werkzeug, das (sinnvoll) im Rahmen der Software-Entwicklung eingesetzt werden kann. Zu einer genaueren Begriffsbestimmung lohnt es sich daher, die einzelnen Werkzeuge zu klassifizieren. Das bekannteste Klassifikationsschema stammt von Fugetta [Fug93]. Er betrachtet primär den Umfang der angebotenen Unterstützung:

- **CASE-Werkzeuge** unterstützen einzelne Aufgaben des Software-Entwicklungsprozesses. Beispiele für CASE-Werkzeuge sind textuelle und graphische Editoren, Compiler oder Debugger. Werkzeuge können isoliert verwendet werden oder zu Werkbänken zusammengefasst werden.

- **CASE-Werkbänke** unterstützen einzelne Phasen oder Aktivitäten des Software-Entwicklungsprozesses, etwa die Anforderungsanalyse, den Entwurf, die Implementierung oder den Test. Sie setzen sich aus einer Menge von CASE-Werkzeugen zusammen, die mehr oder weniger stark integriert sind.
- **CASE-Umgebungen** unterstützen den gesamten Software-Entwicklungsprozess oder zumindestens substanzielle Teile davon. Sie setzen sich aus mehreren CASE-Werkbänken zusammen. Der Grad der Integration zwischen den enthaltenen Werkzeugen ist höher als bei CASE-Werkbänken.

Fugetta differenziert zudem Upper-CASE-Werkzeuge, welche die frühen Phasen der Entwicklung (Anforderungen, Analyse, Entwurf) unterstützen, und Lower-CASE-Werkzeuge für die späteren Phasen (Implementierung, Test). Die für die vorliegende Arbeit relevanten graphischen Modellierungswerkzeuge lassen sich primär dem Bereich Upper-CASE zuordnen. Sie gehören zudem in die Kategorie der CASE-Werkbänke, da sie neben dem graphischen Editor meist noch umfangreiche weitere Funktionalität beinhalten, etwa zur Generierung von Quellcode oder Dokumentation.

Das Klassifikationsschema von Sommerville [Som04] ähnelt dem von Fugetta. Er unterscheidet CASE-Werkzeuge anhand ihrer Funktionalität, ihrer Unterstützung für den Software-Entwicklungsprozess und des Grads ihrer Integration. Die von Nagl vorgeschlagene Klassifikation [Nag93] bezieht eine Reihe von weiteren Perspektiven ein, etwa das Interaktionsverhalten von CASE-Werkzeugen gegenüber dem Benutzer oder die Art und Größe der damit zu realisierenden Projekte.

## 3.2 Integration von CASE-Werkzeugen

CASE-Werkbänke und -Umgebungen integrieren Werkzeuge und damit Funktionalität. Wie groß der sich daraus ergebende Mehrwert gegenüber der Nutzung der einzelnen Werkzeuge ist, hängt von der Art und vom Grad der Integration ab. Thomas und Nejmeh unterscheiden in [TN92] vier Dimensionen der Integration:

- **Datenintegration** sorgt für eine konsistente Datenhaltung zwischen einer Menge von Werkzeugen, etwa indem diesen eine gemeinsame Datenbank mit einem zuvor vereinbarten Datenbankschema zur Verfügung gestellt wird.
- **Steuerungsintegration** bedeutet, dass die Funktionalität verschiedener Werkzeuge flexibel kombiniert werden kann. Im einfachsten Fall werden Werkzeuge dazu etwa aus einem gemeinsamen Menü heraus aufgerufen.
- **Präsentationsintegration** erlaubt es mehreren Werkzeugen, auf eine gemeinsame Benutzungsschnittstelle zuzugreifen. Dies verringert (in einer graphischen Oberfläche) die Anzahl der einzelnen Fenster und damit die kognitive Last des Benutzers.

- **Prozessintegration** stellt sicher, dass die Interaktion der einzelnen CASE-Werkzeuge einem definierten Prozess folgt. Dies setzt voraus, dass eine Prozessmaschine existiert, die eine formale Beschreibung des Prozesses interpretiert.

Die Unterstützung möglichst aller vier Arten von Integration war ein Anliegen vieler früher Forschungansätze zum Thema CASE. Beispielhafte Infrastrukturen, die dies ermöglichen, sind das NIST/ECMA-Referenzmodell für CASE-Umgebungen [NE93] oder das Portable Common Tool Environment (PCTE) [BGMT98]. Ein Beispiel für eine Arbeit jüngeren Datums, die PCTE basiert und alle vier Arten von Integration erreicht, ist [Mon03].

In aktuellen CASE-Produkten spielt Prozessintegration meist keine oder eine untergeordnete Rolle. Präsentations- und Steuerungsintegration werden dadurch erreicht, dass Programmierschnittstellen für sogenannte „Plugins“ bereitgestellt werden. Datenintegration geschieht über die Offenlegung interner Datenstrukturen für diese Plug-Ins oder über die Nutzung standardisierter Austauschformate wie das CASE Interchange Format (CDIF) [Par92], das XML Metadata Interface (XMI) [OMG05b] oder die Graph Exchange Language (GXL) [HWS00].

## 3.3 Probleme mit CASE-Werkzeugen

Der Nutzen von CASE-Werkzeugen innerhalb des Software-Entwicklungsprozesses ist weitgehend unbestritten. Studien belegen einen mittelfristig deutlichen Zuwachs an Qualität und Produktivität durch deren Einsatz (für ein Beispiel siehe [Huf92], zitiert nach [Som04]). Gleichzeitig wurde jedoch schon früh die hohe Komplexität der Werkzeuge kritisiert – ein Problem, das sich aufgrund des stetig wachsenden Integrationsgrades und Funktionsumfangs über die Jahre eher verschärft hat:

- Kemerer zitiert in [Kem92] eine Reihe von Untersuchungen, nach denen 70 bis 90 Prozent der von Firmen angeschafften CASE-Werkzeuge nicht genutzt werden. Er führt die mangelnde Akzeptanz auf eine Lernkurve zurück, innerhalb derer die Produktivität bei der Nutzung eines neuen CASE-Werkzeugs zunächst für eine Weile merklich absinkt, bevor ein Zuwachs an Produktivität erkennbar wird. Der Effekt nimmt mit höherem Integrationsgrad – also steigender Komplexität – der Werkzeuge zu.
- Lending und Chervany berichten in [LC98] von einer sehr geringen Nutzung von Modellierungswerkzeugen in den untersuchten Einrichtungen. Wo sie genutzt werden, kommt oft nur grundlegende Funktionalität zum Einsatz. Fortgeschrittene Möglichkeiten zu Analyse und Transformation bleiben ungenutzt. Die Autoren führen dies auf geringe intrinsische („joy of use“) und extrinsische („usefulness“) Motivation zur Nutzung der Werkzeuge zurück.

- Iivari beschreibt in [Iiv96] eine Studie zur Nutzung von CASE-Werkzeugen, die nicht deskriptiv bleibt, sondern Hypothesen testet. Er stellt einen starken Einfluss der empfundenen Komplexität des Werkzeugs auf die Effektivität der Nutzung fest und betrachtet die stetig wachsende Komplexität von CASE-Werkzeugen insgesamt kritisch. Als Hilfsmittel zur Nutzung wird die Integration von Tutor- oder Hypertextsystemen vorgeschlagen.

Auch wenn diese Untersuchungen zunächst die Nutzung von CASE-Werkzeugen im professionellen Umfeld betreffen, sind sie doch auf ein akademisches Umfeld und eine studentische Nutzerschaft übertragbar. Das Problem der Komplexität stellt sich dort offensichtlich in verstärkter Weise: Während bei professionellen Nutzern davon auszugehen ist, dass ihnen die Notation bereits bekannt ist, sie sich also „nur“ mit dem eigentlichen Werkzeug vertraut machen müssen, ist dies nicht der Fall, wenn das Werkzeug zur Lehre der Notation genutzt wird. Wird das Werkzeug zudem nur für eine kurze Zeit eingesetzt, weil die Veranstaltung sich dann anderen Inhalten zuwendet, werden die Nutzer den „negativen“ Bereich der Lernkurve nicht verlassen, also nicht effektiv mit dem Werkzeug arbeiten können. Ein möglicherweise daraus resultierender Nebeneffekt, der aus didaktischer Sicht besonders ungünstig wäre, ist eine Abneigung gegen die spezielle Modellierungsmethode oder Modellierung im Allgemeinen, da das Arbeiten mit dem Werkzeug als unproduktiv empfunden wurde.

## 3.4 Beispiele für CASE-Werkzeuge

Die folgenden Abschnitte stellen exemplarisch vier Modellierungswerkzeuge vor, von denen zwei kommerzielle Produkte sind und zwei dem akademischen Umfeld entstammen. Sie wurden ausgewählt, weil sie jeweils bekannte, verbreitete Vertreter ihrer Gruppe und somit repräsentativ für kommerzielle bzw. akademische Werkzeuge sind. Alle vier unterstützen die UML und wurden bzw. werden nach dem Wissen des Autors in der einschlägigen Lehre eingesetzt. Das erste Werkzeug steht den Studierenden an der Universität Dortmund innerhalb von Softwaretechnik-Veranstaltungen und während des Software-Praktikums [DZ98] zur Verfügung. Alle vier Werkzeuge sind in Java implementiert und damit plattformübergreifend lauffähig. Tabelle 3.1 listet die jeweiligen Systemanforderungen auf.

| Produkt | Download | Festplatte | Hauptspeicher |
|---|---|---|---|
| Together | 134 MB | 282 MB | 106 MB |
| Magic Draw | 67 MB | 129 MB | 55 MB |
| ArgoUML | 9 MB | 9 MB | 43 MB |
| Fujaba | 9 MB | 12 MB | 36 MB |

Tabelle 3.1: Systemanforderungen verschiedener CASE-Werkzeuge

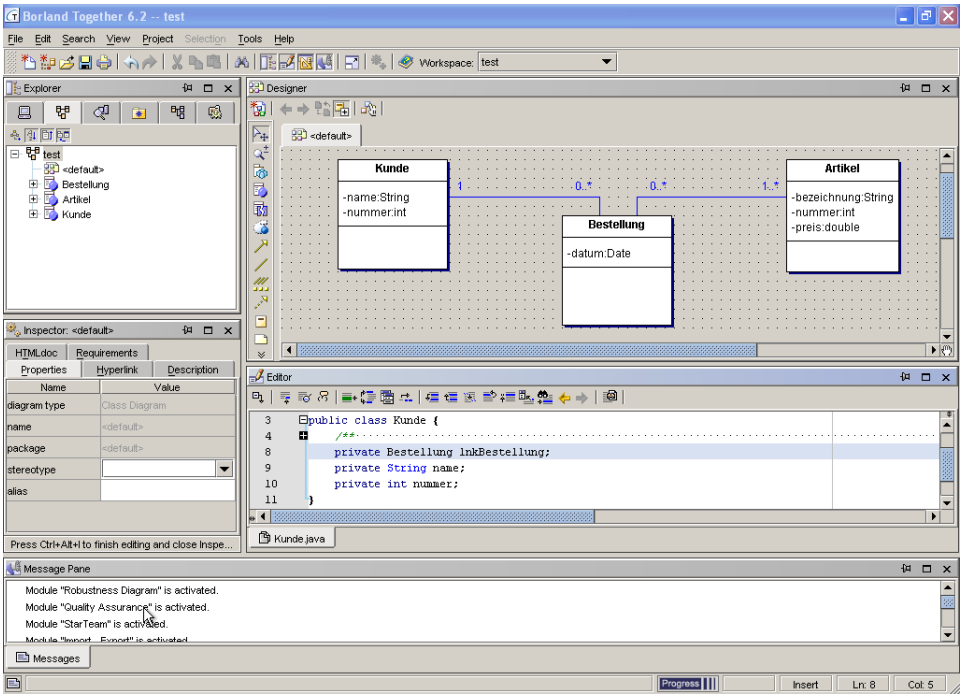



Abbildung 3.1: Benutzungsschnittstelle von Together

### 3.4.1 Borland Together 6.2

Das Werkzeug Together [Tog06] der Firma Borland lässt sich grob der Kategorie der CASE-Umgebungen zurechnen, da es weite Teile des Software-Entwicklungsprozesses von der Modellierung bis hin zur Compilierung und zum Debugging einer Anwendung abdeckt. Der Schwerpunkt liegt jedoch auf der Modellierung. Dazu stehen acht Diagrammtypen der UML und weitere 10 Diagrammtypen aus anderen Bereichen zur Verfügung. Zu einem Strukturmodell wird automatisch Quellcode generiert. Manuelle Änderungen an diesem Quellcode werden in das Modell übernommen. Es existieren zudem Möglichkeiten, Schemata relationaler Datenbanken zu analysieren und zu erzeugen. Modelle können ins XMI-Format, in verschiedene Grafikformate und in eine HTML-basierte Dokumentation exportiert werden. Eine Zusammenarbeit mehrerer Nutzer über ein Versionsmanagement-System wird unterstützt.

Die Benutzungsschnittstelle von Together (siehe Abbildung 3.1) ist – selbst für ein Werkzeug dieses Funktionsumfangs – sehr komplex. Den größten Teil des Fensters nehmen der graphische und der textuelle Editor ein. Hinzu kommt ein Bereich mit verschiedenen hierarchischen Sichten auf das gesamte Modell (Schachtelung der Elemente, Vererbungshierarchie etc.). Zur Bearbeitung vieler Eigenschaften eines Modellelements muss ein zusätzlicher Dialog geöffnet werden. Dieser kann jedoch dauerhaft im linken Bereich des Fensters sichtbar bleiben. In fast allen Bereichen der Benutzungsschnittstelle finden sich

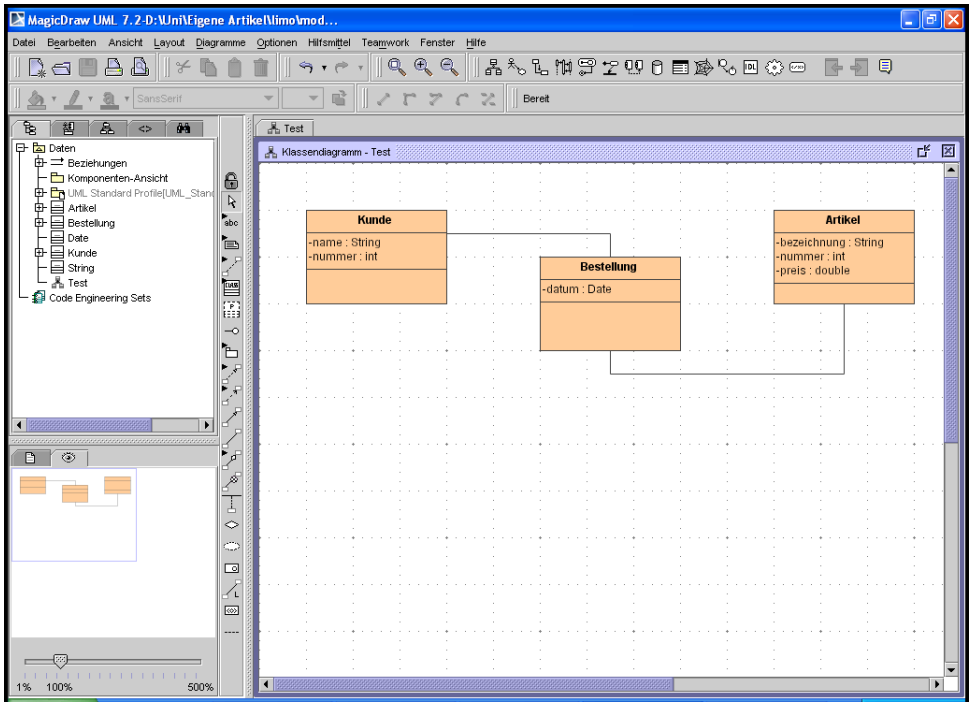



Abbildung 3.2: Benutzungsschnittstelle von MagicDraw

sehr umfangreiche Menüs und Werkzeugleisten, letztere mit kleinen, relativ wenig aussagekräftigen Symbolen.

### 3.4.2 No Magic MagicDraw 7.2

Das von der Firma No Magic stammende Produkt MagicDraw [Mag06] lässt sich als CASE-Werkbank für Analyse- und Design-Tätigkeiten beschreiben. Zur Modellierung stehen sieben Diagrammtypen der UML und ebenso viele Diagrammtypen aus anderen Bereichen zur Verfügung. Aus Strukturmodellen kann Quellcode generiert werden; dies muss aber hier explizit angestoßen werden. Ebenso wird das automatisierte Erzeugen von Modellen zu existierendem Code unterstützt. Auch die Analyse und Erzeugung von Datenbank-Schemata ist möglich. MagicDraw speichert seine Modelle bereits im XMI-Format, so dass die Interoperabilität mit anderen Werkzeugen gegeben ist. Modelle können zudem in verschiedene Grafikformate und eine HTML-basierte Dokumentation überführt werden. Auch MagicDraw unterstützt die Zusammenarbeit mehrerer Nutzer über ein Versionsmanagement-System.

Die Benutzungsschnittstelle von MagicDraw (siehe Abbildung 3.2) ähnelt in der Struktur der von Together, ist aber übersichtlicher. Der graphische Editor nimmt auch hier den größten Teil des Fensters in Anspruch. Am linken Rand des Fensters finden sich ein

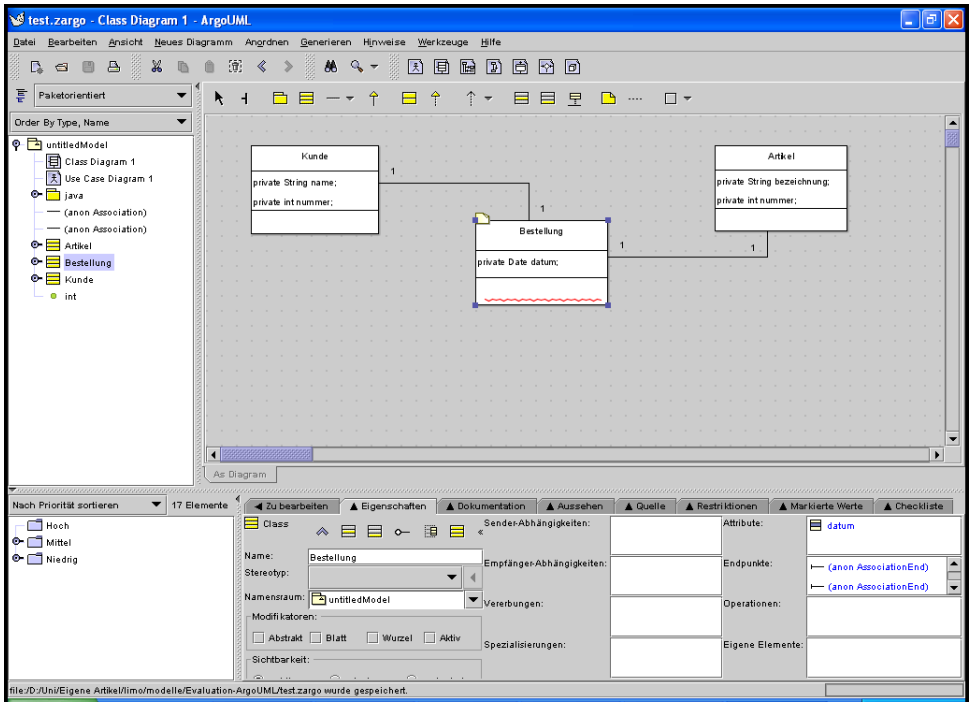



Abbildung 3.3: Benutzungsschnittstelle von ArgoUML

Bereich für verschiedene hierarchische Darstellungen des aktuellen Modells, ein Bereich zur Dokumentation der Modellelemente und eine Übersichtskarte des Modells. Die Steuerung des graphischen Editors geschieht auf weitgehend intuitive Weise mithilfe der Maus. Zur Bearbeitung der Eigenschaften eines Modellelements muss – ähnlich wie bereits bei Together – ein spezieller Dialog geöffnet werden. Dieser kann aber hier nicht dauerhaft sichtbar bleiben, was sich bei der Modellierung als hinderlich erweist.

### 3.4.3 ArgoUML 0.18

Das Open-Source-Projekt ArgoUML [Arg06] ist ebenfalls eine CASE-Werkbank für Analyse- und Designtätigkeiten, fällt aber im Funktionsumfang schlichter aus als die beiden zuvor beschriebenen kommerziellen Produkte. In ArgoUML stehen sieben verschiedene Diagrammtypen der UML zur Verfügung, aber keine Diagramtypen aus anderen Bereichen. ArgoUML unterstützt das Generieren von Java-Quellcode aus einem strukturellen Modell, aber keine Rücküberführung von existierendem Code in ein Modell. Modelle können ins XMI-Format und in diverse Grafikformate exportiert werden.

Die Benutzungsschnittstelle von ArgoUML (siehe Abbildung 3.3) ist in erster Näherung ähnlich strukturiert wie die der beiden anderen Werkzeuge. Zentrale Elemente sind wieder

der graphische Editor und die hierarchische Sicht des Modells. Es existieren jedoch einige Besonderheiten:

- **Kognitive Unterstützung** – Modellelemente im Diagramm besitzen Anknüpfungspunkte. Diese ermöglichen das schnelle Erzeugen eines weiteren Modellelements, das mit dem aktuellen Element in Beziehung steht (etwa eine neue Ober- oder Unterklasse), ohne dass dafür die Werkzeugleiste bemüht werden muss [RR99].
- **Kritik-Mechanismus** – Das Modell wird ständig auf der Basis von Metriken auf die Verletzung von Design-Richtlinien analysiert (etwa namenlose Elemente oder zyklische Vererbung). Die Ergebnisse dieser Analyse werden zusammen mit Verbesserungsvorschlägen im unteren Bereich des Fensters angezeigt [RHR98].

Zur Bearbeitung der Eigenschaften eines Modellelements ist ein eigener, dauerhaft sichtbarer Bereich vorgesehen. Dieser ist jedoch etwas ungünstig unterhalb des Editors positioniert und nimmt relativ viel Platz in Anspruch. Trotzdem sind teilweise nicht alle Elemente sichtbar.

### 3.4.4 Fujaba 4.0

Auch das UML-Werkzeug From UML To Java and Back Again (Fujaba) [NNZ00] lässt sich als CASE-Werkbank für den Bereich Analyse und Entwurf klassifizieren. Wie der Name andeutet, unterstützt es die Generierung von Java-Quellcode aus einem Modell und umgekehrt. Im Unterschied zu den drei bisher vorgestellten Werkzeugen ist diese Fähigkeit nicht auf den strukturellen Teil des Modells beschränkt, sondern erstreckt sich auch auf dessen dynamische Anteile. Dazu unterstützt Fujaba drei Diagrammtypen der UML, die aber um spezielle Fähigkeiten erweitert wurden. Eine zentrale Rolle spielen hier die sogenannten Story-Diagramme [FNTZ98], eine um Graphtransformationen angereicherte Fassung von Aktivitätsdiagrammen, mit denen Änderungen der Objektstruktur eines Programms graphisch beschrieben werden können.

Die Benutzungsschnittstelle von Fujaba (siehe Abbildung 3.4) ist deutlich reduzierter als die der anderen drei Werkzeuge. Das Fenster wird im Wesentlichen durch den graphischen Editor und eine Übersicht der vorhandenen Diagramme gefüllt. Die Bedienung des Editors ist jedoch nur teilweise intuitiv und streckenweise eher unkomfortabel. So ist es zwar möglich, Position und Größe der Knoten des Diagramms (zum Beispiel Klassen oder Zustände) mit der Maus zu manipulieren. Zum Verändern der Eigenschaften von Kanten (also etwa Assoziationen oder Transitionen) muss jedoch ein spezieller Dialog geöffnet werden. Insbesondere ist es nicht möglich, die Enden einer solchen Kante mit der Maus direkt zu manipulieren.

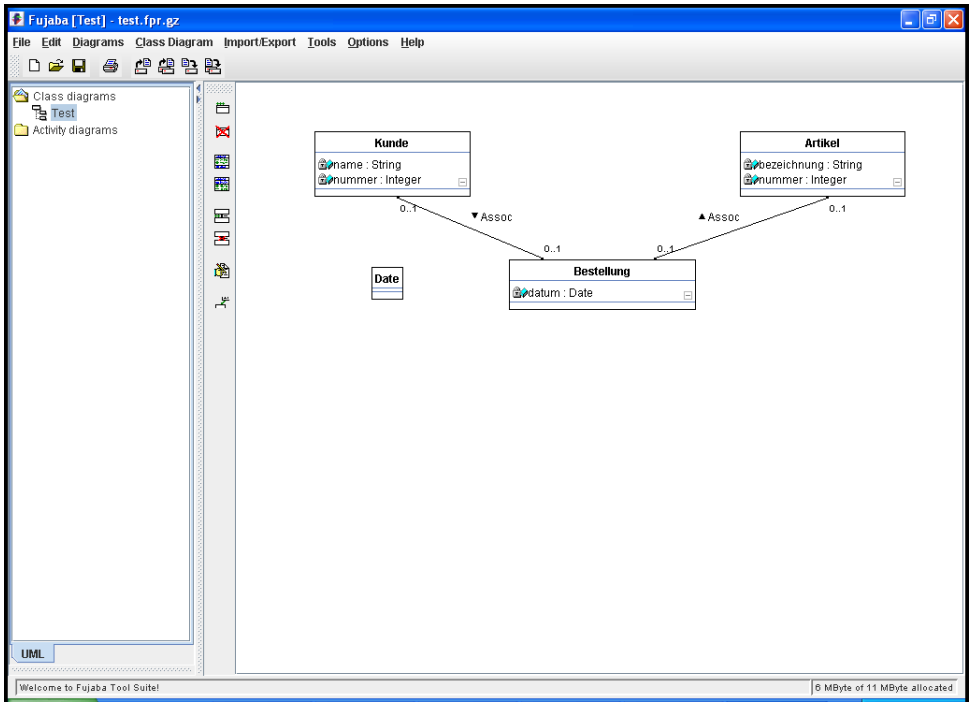



Abbildung 3.4: Benutzungsschnittstelle von Fujaba

## 3.5 Werkzeuge für die Programmierausbildung

Während dedizierte Werkzeuge zur Unterstützung der Modellierungsausbildung offensichtlich rar sind, hat die Entwicklung spezifischer Sprachen und Werkzeuge für die Programmierausbildung eine lange Tradition. Die Programmiersprachen Basic, Pascal und Logo sind ursprünglich zu Ausbildungszwecken entstanden. Programmierumgebungen wie Alice [CAB+00], Karel [Pat81] oder Stagecast [SCT00] sind geeignet, Schülern frühzeitig auf anschauliche und spielerische Weise rudimentäre Programmierkenntnisse zu vermitteln.

Kölling stellt in [Köl99a] einen Anforderungskatalog für lehretaugliche Werkzeuge auf, die er auch als Lernumgebungen bezeichnet. Einige der Anforderungen sind spezifisch für seinen Kontext der Einführung in die objektorientierte Programmierung. Die meisten lassen sich jedoch problemlos auf Modellierungswerkzeuge übertragen:

- **Einfache Bedienung** – Eine Lernumgebung muss so einfach zu bedienen sein, dass sie von den typischerweise unerfahrenen Studierenden niedriger Semester bereits nach einer kurzen Zeit produktiv eingesetzt werden kann.
- **Integration** – Eine Lernumgebung sollte verschiedene Werkzeuge unter einer einheitlichen Benutzungsschnittstelle integrieren und so einen Mehrwert gegenüber der

Nutzung der einzelnen Werkzeuge anbieten.

- **Unterstützung des Lernens** – Eine Lernumgebung muss Funktionalität beinhalten, welche die Lehre explizit unterstützt. Kölling schlägt zur Vermittlung von Java die Visualisierung von und die Interaktion mit Klassen- und Objektstrukturen vor [Köl99b].
- **Unterstützung von Gruppenarbeit** – Software-Entwicklung bedeutet Arbeit im Team. Eine Lernumgebung sollte die gemeinsame Erarbeitung und den Austausch von Ergebnissen in Übungs- und Praktikumsgruppen unterstützen.
- **Verfügbarkeit** – Die Anschaffungskosten professioneller Software sowie die Folgekosten für die benötigte Hardware (die aufgrund der Komplexität nötig wird) sind für Bildungseinrichtungen oft zu hoch. Eine Lernumgebung muss zu überschaubaren Kosten verfügbar sein und auf allgemein verfügbarer Hardware laufen.

Eine weitere Charakterisierung von lehretauglichen Werkzeugen, die von vielen Autoren angeführt wird und sich in Teilen mit dem obigen Katalog deckt, ist die der Leichtgewichtigkeit [HCB04, ACS02, MH03, SMM+03, BNT02]. Der Begriff hat mehrere Facetten:

- **Funktionalität** – Das Werkzeug sollte sich auf die für die Lehre essenziell notwendige Funktionalität beschränken, die natürlich – je nach Kontext – zunächst zu bestimmen ist.
- **Bedienung** – Das Werkzeug sollte bereits mit geringem Lernaufwand effektiv genutzt werden können.
- **Systemanforderungen** – Das Werkzeug sollte genügsam bezüglich der Anforderungen an die unterliegende Hard- und Software sein.
- **Implementierung** – Das Werkzeug sollte mit geringem Aufwand zu implementieren und zu warten sein.

Michiels et al. heben zusätzlich hervor, dass lehretaugliche Werkzeuge zwar möglichst leichtgewichtig sein sollen, aber nicht zu primitiv werden dürfen [MBBF03]. Auch Kölling betont, dass zu einfache Werkzeuge aus pädagogischer Sicht ähnlich ungeeignet sind wie zu komplexe [KQPR03]. Gleichzeitig müssen sie einen didaktischen und motivationalen Mehrwert besitzen: Jede Lernumgebung sollte den Spaß am Lernen durch Interaktivität und unmittelbares Feedback erhöhen [Köl99a].

| Produkt | Download | Festplatte | Hauptspeicher |
|---|---:|---:|---:|
| BlueJ | 2,7 MB | 4,5 MB | 47 MB |
| jGrasp | 2,6 MB | 4,5 MB | 48 MB |
| Dr. Java | 6,5 MB | 6,5 MB | 52 MB |
| Gild (+Eclipse) | 2 MB (+80 MB) | 3 MB (+140 MB) | 115 MB |

Tabelle 3.2: Systemanforderungen einiger Werkzeuge für die Lehre

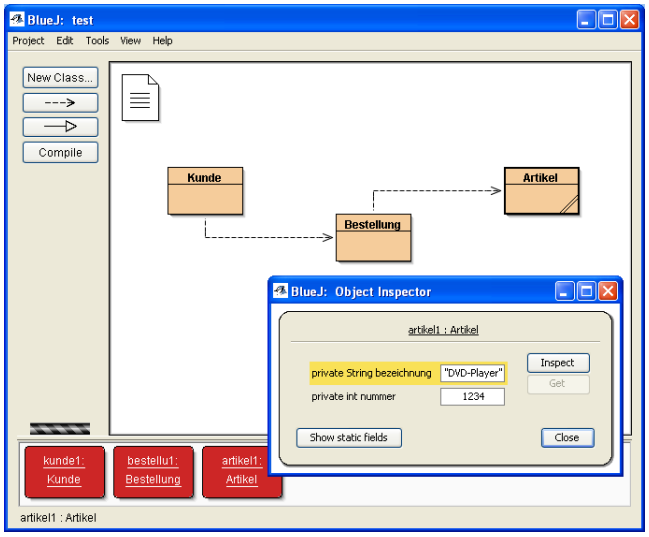



Abbildung 3.5: Benutzungsschnittstelle von BlueJ

## 3.6 Beispiele für Werkzeuge zur Programmierausbildung

Die folgenden Abschnitte stellen exemplarisch einige Werkzeuge vor, die der Lehre objektorientierter Programmierung mit Java dienen. Die ersten drei davon gehören zu den bekanntesten eigenständigen Arbeiten auf diesem Gebiet, während die letzten beiden Arbeiten eine professionelle Entwicklungsumgebung an den Ausbildungskontext anpassen. Alle Werkzeuge betonen den leichtgewichtigen Ansatz, der sie von professionellen Werkzeugen unterscheidet. Die Systemanforderungen der Werkzeuge liegen, wie Tabelle 3.2 zeigt, teilweise deutlich unter denen der zuvor diskutierten CASE-Werkzeuge .

### 3.6.1 BlueJ 2.1.3

BlueJ [KQPR03] ist eine integrierte graphische Java-Entwicklungsumgebung für Einsatzzwecke in der Lehre. Neben einer generellen Reduktion der Komplexität im Vergleich zu professionellen Werkzeugen ist ein wesentliches Merkmal von BlueJ die technische Unterstützung des „Objects First“-Ansatzes: Die Studierenden können unmittelbar Klassen instanziieren und mit den Objekten interagieren, ohne dass dafür ein komplettes Programm (inklusive einer `main()`-Methode) notwendig wäre. Es können gezielt Methoden aufgerufen werden, und die Eigenschaften eines Objekts vor und nach dem Aufruf können inspiziert werden. Die Studierenden erhalten so frühzeitiges Feedback und Einblicke in die Semantik von Klassen und Objekten. Insbesondere der Unterschied zwischen beiden Konzepten wird deutlich.

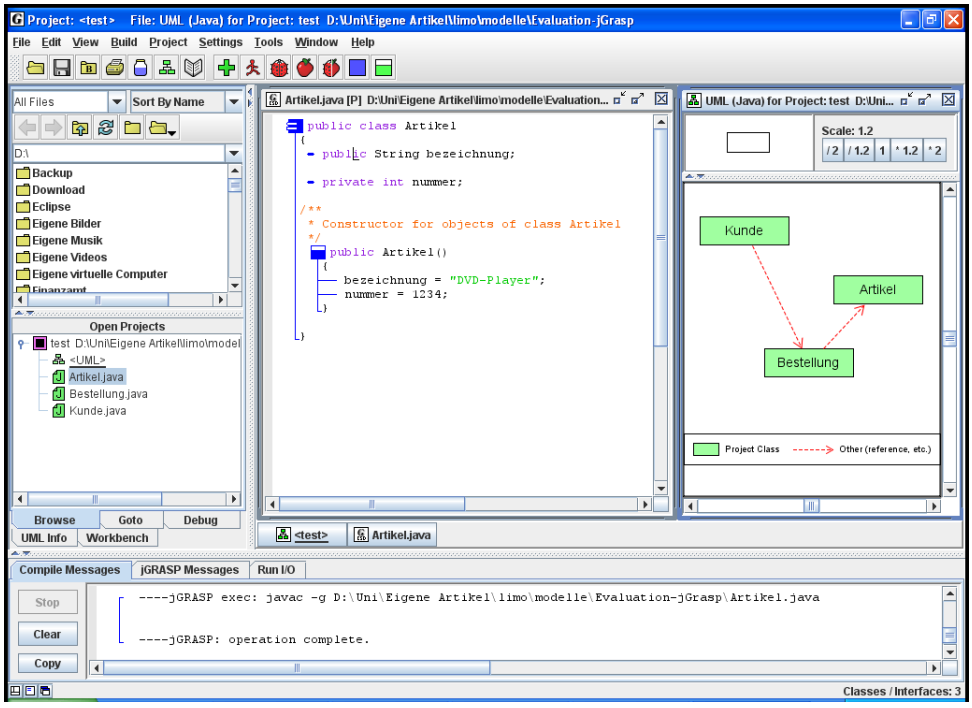



Abbildung 3.6: Benutzungsschnittstelle von jGrasp

Die Benutzungsschnittstelle von BlueJ (siehe Abbildung 3.5) weicht stark von der üblicher Programmierwerkzeuge ab. Der größte Teil des Fensters wird von einer Art einfachem Klassendiagramm des Projekts in Anspruch genommen. Jede dieser Klassen besitzt ein Kontextmenü, über das zum Beispiel ein Quelltexteditor geöffnet oder eine neue Instanz erzeugt werden kann. Unterhalb der Klassen sind die bereits existierenden Instanzen angeordnet. Deren Kontextmenüs enthalten unter anderen die jeweils aufrufbaren Methoden. Am linken Rand des Fensters befindet sich eine Werkzeugleiste mit Funktionen, die das gesamte Projekt betreffen.

### 3.6.2 jGrasp 1.8.4

Ähnliche Ziele wie BlueJ verfolgt auch die Programmierumgebung jGrasp [HCB04], die neben Java noch verschiedene andere Programmiersprachen unterstützt. Neben der gezielten Interaktion mit Klassen und Objekten liegt ein Schwerpunkt von jGrasp auf der Unterstützung des Programmverständnisses durch Visualisierung:

- Die syntaktische Struktur des Programms kann durch einen Kontrollflussgraphen verdeutlicht werden. Dieser entspricht grob dem Syntaxbaum und wird in den Java-Quellcode eingeblendet.

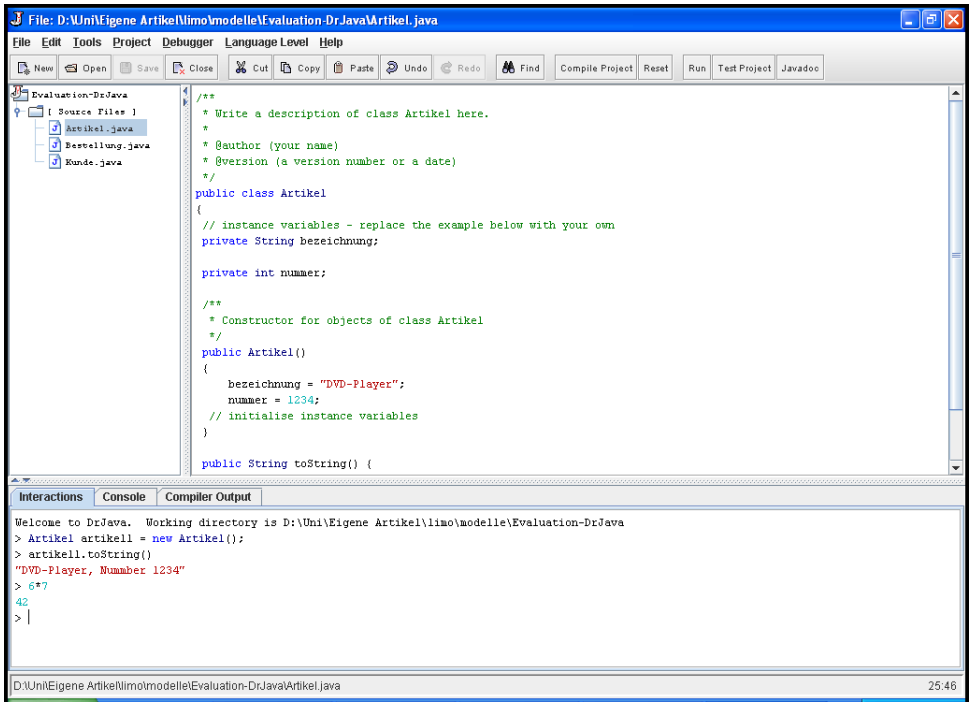



Abbildung 3.7: Benutzungsschnittstelle von Dr. Java

- Die Klassenstruktur des Projekts wird mit Hilfe eines stark vereinfachten UML-Klassendiagramms dargestellt. Dieses enthält neben den Klassen nur Vererbungs- und Benutzungsbeziehungen.
- Die Datenstrukturen eines Programms können anschaulich durch Graphen dargestellt werden. jGrasp enthält bereits Visualisierungen für übliche Datenstrukturen wie Listen und Bäume. Eine Plugin-Schnittstelle ermöglicht das Hinzufügen eigener Visualisierungen.

Die Benutzungsschnittstelle von jGrasp (siehe Abbildung 3.6) orientiert sich an der professioneller Werkzeuge, ist aber – abhängig davon, welche Funktionalität tatsächlich genutzt wird – bereits relativ komplex und verfehlt damit das Kriterium der Leichtgewichtigkeit. Auf der Ebene der Implementierung kann jGrasp aber durchaus als leichtgewichtig gelten.

### 3.6.3 Dr. Java 20061025-1556

Das Werkzeug Dr. Java [ACS02] ermöglicht ebenfalls die unmittelbare Interaktion mit Klassen und Objekten, verzichtet aber gänzlich auf eine graphische Darstellung. Stattdessen wird eine Endlosschleife aus Eingabe, Berechnung und Ausgabe (Read-Eval-Print

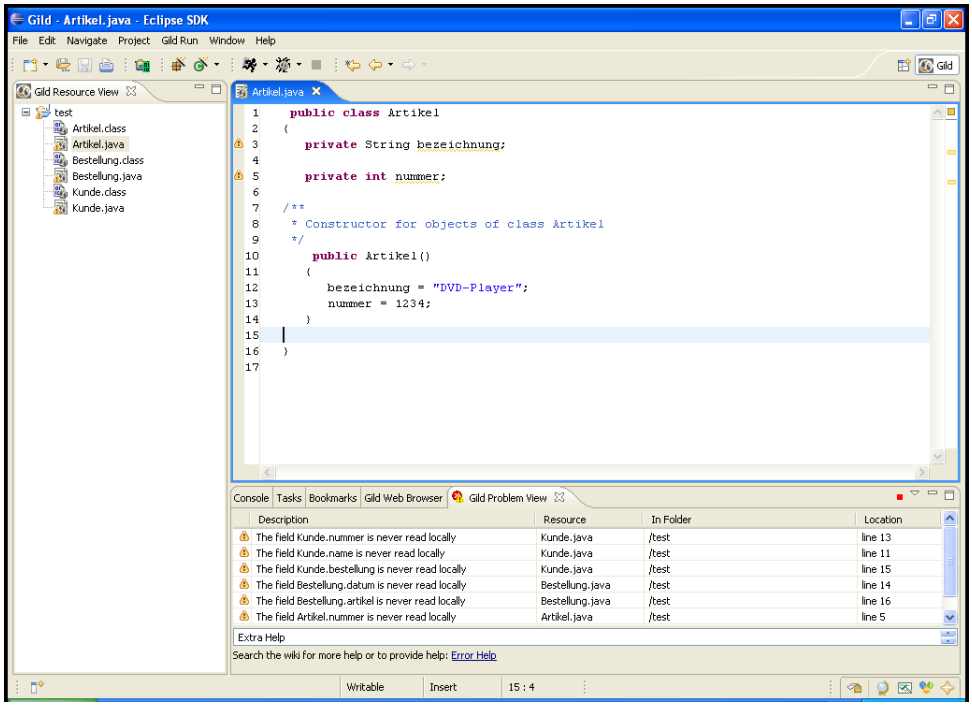



Abbildung 3.8: Benutzungsschnittstelle von Gild

Loop) angeboten, die funktionalen Programmiersprachen wie Lisp, Scheme oder ML entlehnt ist. Diese ermöglicht ein unkompliziertes, interaktives Experimentieren mit den Elementen der Sprache.

Die Benutzungsschnittstelle von Dr. Java (siehe Abbildung 3.7) ist sehr einfach strukturiert. Sie besteht nur aus zwei Bereichen:

- Ein Definitionsbereich (oberer Teil des Fensters) dient der Implementierung kompletter Klassen.
- Ein Interaktionsbereich (unterer Teil des Fensters) nimmt Eingaben des Benutzers entgegen und zeigt die Ergebnisse der Auswertung an.

Dr. Java erreicht Leichtgewichtigkeit sowohl auf der Ebene der Benutzungsschnittstelle als auch auf der Implementierungsebene.

### 3.6.4 Penumbra 1.4.3 / Gild 2.2.0

Penumbra [MH03] und Gild [SMM+03] sind keine eigenständigen Werkzeuge, sondern Anpassungen der professionellen Entwickungsumgebung Eclipse [Ecl06] an die Bedürfnisse der universitären Ausbildung. Beide Projekte stellen dazu jeweils ein Plugin bereit,

das die Benutzungsschnittstelle von Eclipse stark reduziert. Studierende werden nur noch mit einer einzigen Perspektive konfrontiert, die Möglichkeiten zum Bearbeiten, Übersetzen und Ausführen von Java-Code enthält. Dieser Kernfunktionalität fügen die Projekte nur wenig hinzu:

- Penumbra bietet eine einfache Möglichkeit der elektronischen Abgabe eines Projekts als Lösung zu einer Übungsaufgabe. Dies setzt das Vorhandensein einer speziellen Server-Infrastruktur voraus.
- Gild integriert eine Hypertext-Sicht, die sowohl Vorlesungsmaterial als auch Notizen zu Projekten anzeigen kann. Der Hypertext erlaubt die Integration von und die Verlinkung mit dem Quellcode.

Beide Projekte erreichen die Leichtgewichtigkeit nur auf der Ebene der Benutzungsschnittstelle (siehe Abbildung 3.8), nicht aber auf der Implementierungsebene. Die technische Komplexität von Eclipse ist noch vorhanden, wird aber verborgen. Der Vorteil, der sich daraus ergibt, ist der leichtere Übergang von der Lernumgebung zur vollständigen Entwicklungsumgebung – es muss nur die Perspektive gewechselt werden.

## 3.7 Werkzeuge aus Sicht der Lehre

Eine Unterstützung der Modellierungsausbildung durch Werkzeuge ist prinzipiell wünschenswert. Werkzeuge bedeuten für Studierende wie Tutoren eine Arbeitserleichterung. Studierende werden zudem bereits frühzeitig an deren Nutzung gewöhnt.

Zum Einsatz kommen vielfach kommerzielle CASE-Werkzeuge. Deren Komplexität, die bereits im professionellen Umfeld als problematisch gesehen wird, resultiert schnell darin, dass mehr Zeit in die Vermittlung des Werkzeugs als in die eigentliche Modellierungssprache investiert wird. Die hohen unmittelbaren und mittelbaren Kosten der Werkzeuge sind für Hochschulen mit vielen Studierenden ebenfalls ein Problem. Werkzeuge aus dem akademischen oder dem Open-Source-Umfeld scheinen eine Alternative zu sein, da sie meist weder mit der Komplexität noch mit den Kosten professioneller Werkzeuge belastet sind. Jedoch ist ihre Bedienbarkeit teilweise eher schlecht, was nachvollziehbar ist, da im Rahmen von Forschungsprojekten üblicherweise keine „polierten“ Werkzeuge entstehen.

Beiden Gruppen von Werkzeugen fehlt Funktionalität, die das Lernen einer Modellierungssprache gezielt unterstützt. Damit sind sie als Lernumgebung für Studierende niedriger Semester, etwa innerhalb einer Softwaretechnik-Veranstaltung, keine gute Wahl.

Für die Programmierausbildung wurde die Werkzeug-Problematik bereits früher erkannt. Eine Reihe von Ansätzen zeigt, wie Programmierwerkzeuge dediziert für die Lehre neu entwickelt werden können oder bestehende Werkzeuge an die Bedürfnisse der Lehre angepasst werden können. Dabei sind es zwei wesentliche Eigenschaften, die eine Lernumgebung auszeichnen und die sich in all diesen Ansätzen wiederfinden:

- **Leichtgewichtigkeit** – Die Werkzeuge müssen sich auf essenzielle Funktionalität beschränken und für unerfahrene Nutzer einfach zu bedienen sein. Idealerweise sind die Anforderungen an die umgebende Hard- und Software gering, und ihre Entwicklung und Wartung ist unaufwendig.
- **Didaktisch motivierte Funktionalität** – Die Werkzeuge müssen Funktionalität bieten, die das Lernen gezielt unterstützt. Zur Vermittlung der Programmiersprache Java haben sich die Visualisierung von Programm- und Laufzeitstrukturen sowie interaktive Experimente bewährt.

Da die Modellierungsausbildung streckenweise mit ähnlichen Problemen zu tun hat wie die Programmierausbildung – in beiden Fällen müssen Syntax und Semantik einer Sprache vermittelt werden – bietet es sich an, von diesen positiven Erfahrungen zu profitieren und sie auf Modellierungswerkzeuge zu übertragen. Das nächste Kapitel stellt ein entsprechendes Konzept für leichtgewichtige, didaktische Modellierungswerkzeuge vor.

# 4 Lösungsansatz

Das Ziel der vorliegenden Arbeit ist die Konzeption, prototypische Umsetzung und Evaluation von graphischen Modellierungswerkzeugen für die Lehre der Softwaretechnik. Dazu orientiert sich die Arbeit an den im vorangehenden Kapitel formulierten Ideen für und Anforderungen an lehretaugliche Werkzeuge, die sich den beiden zentralen Begriffen der Leichtgewichtigkeit und der Integration didaktisch motivierter Funktionalität unterordnen. Viele dieser Anforderungen lassen sich unmittelbar auf den Kontext der Modellierungswerkzeuge übertragen. So sind etwa die Kriterien für einfache Bedienbarkeit weitgehend unabhängig vom Kontext der Anwendung. Unterschiede bestehen aber zum Beispiel in der essenziellen Funktionalität, die für ein Modellierungswerkzeug erst noch zu bestimmen ist, und in der didaktischen Unterstützung, die ein solches Werkzeug leisten kann. Die folgenden Abschnitte geben einen ersten Überblick über den Lösungsansatz, der von dieser Arbeit vorgeschlagen und verfolgt wird. Die Gliederung folgt dabei den im vorangehenden Kapitel erarbeiteten Kriterien.

## 4.1 Überlegungen zur Leichtgewichtigkeit

Die erste Gruppe von Anforderungen aus Abschnitt 3.7 umfasste verschiedene Facetten des Begriffs der Leichtgewichtigkeit. Dazu zählten ein bewusst eingeschränkter Funktionsumfang, eine leichte Bedienbarkeit sowie geringe Systemanforderungen der Werkzeuge. Außerdem sollten die Werkzeuge mit begrenztem Aufwand entwickel- und wartbar sein.

### 4.1.1 Funktionalität

Wie in Abschnitt 3.5 diskutiert, ist eine wesentliche Eigenschaft leichtgewichtiger Werkzeuge deren Beschränkung auf die – für die Lehre – essenzielle Funktionalität. Die essenzielle Funktionalität eines Modellierungswerkzeugs ist ein syntaxgesteuerter graphischer Editor, der die Bearbeitung diagrammatischer Modelle erlaubt. Die Modelle müssen gespeichert, geladen und ausgedruckt werden können. Zudem ist eine Exportmöglichkeit wünschenswert, um die Modelle in andere Dokumente (Texte, Präsentationen etc.) einzubinden oder sie mit anderen CASE-Werkzeugen weiterzubearbeiten.

Auch wenn, wie in Abschnitt 2.5 gezeigt wurde, verschiedene Modellierungssprachen relevant für die Softwaretechnik sind, ist es nicht sinnvoll, diese alle in einem einzigen

Werkzeug zu unterstützen. Michiels et al. [MBBF03] plädieren stattdessen für kleine, spezialisierte Werkzeuge mit einem klar definierten Anwendungszweck. Die vorliegende Arbeit folgt diesem Ansatz und entwickelt eine Produktfamilie von graphischen Modellierungswerkzeugen, deren Mitglieder jeweils auf die Lehre einer Modellierungssprache spezialisiert sind.

### 4.1.2 Bedienbarkeit

Entscheidend für die Bedienbarkeit eines Werkzeugs ist die Gestaltung der Benutzungsschnittstelle. Hier stehen nach den Diskussionen aus Abschnitt 3.6 zwei grundsätzliche Varianten offen:

- Die Benutzungsschnittstelle ist minimalistisch und orientiert sich ausschließlich an didaktischen Gesichtspunkten. Das Resultat ist eine reine Lernumgebung, die nicht oder nur sehr eingeschränkt im Rahmen einer realen Software-Entwicklung genutzt werden kann. Die Programmierumgebungen BlueJ und Dr. Java gehen diesen Weg.
- Die Benutzungsschnittstelle orientiert sich an den üblichen Gepflogenheiten bei einschlägigen Werkzeugen, reduziert diese aber auf das Nötigste. Die Plugins Penumbra und Gild zur Anpassung von Eclipse verfolgen diese Strategie. Auch jGrasp besitzt eine für Programmierumgebungen übliche Schnittstelle, deren Komplexität jedoch immer noch recht hoch ist.

Aufgrund der im Kontext von BlueJ geschilderten potenziellen Schwierigkeiten beim Übergang von einer reinen Lernumgebung zu einem echten Werkzeug [KQPR03] verfolgt diese Arbeit die zweite Strategie. Die Benutzungsschnittstelle soll zudem üblichen Anforderungen an Bedienbarkeit genügen. Dazu zählen insbesondere Übersichtlichkeit, die Berücksichtigung der „7 bis 11 Elemente“-Regel, das Platzieren von zentralen und häufig benötigten Elementen an markante Stellen wie einer Werkzeugleiste, das Auslagern weniger zentraler Elemente ins Menü, Kontextabhängigkeit, große und intuitiv verständliche Symbole, etc.

Ein Teil dieser Anforderungen scheint trivial, wird aber, wie Abschnitt 3.4 zeigt, oft vernachlässigt. Gerade bei Forschungsprojekten steht die Gestaltung der Benutzungsschnittstelle meist im Hintergrund, da eine Implementierung mehr als Beweis für die Machbarkeit der entwickelten Konzepte dient. Die im Rahmen der Arbeit entstehenden Werkzeuge sind jedoch keine reinen Forschungsprojekte, sondern sollen innerhalb des Lehrbetriebs eingesetzt werden. Damit ist die Gestaltung der Schnittstelle relevant.

Die Arbeit widerspricht damit insbesondere der Forderung von Biddle und Noble nach möglichst unpolierten Werkzeugen („messy is good“) [NB04], bei denen die Hemmung zur Nutzung seitens der Studierenden am geringsten sei. Nach der Erfahrung des Autors sehen sich Studierende der Informatik sehr stark als Experten für Software-Produkte und betrachten deren Qualität entsprechend kritisch.

### 4.1.3 Systemanforderungen

Die Werkzeuge sollen geringe Anforderungen an die umliegende Hard- und Software stellen, um auf möglichst vielen universitären und studentischen Rechnern problemlos lauffähig zu sein. Auch wenn eine Nennung konkreter Werte an dieser Stelle weder möglich noch sinnvoll ist, sollen die Werkzeuge die in Abschnitt 3.4 erwähnten professionellen Modellierungswerkzeuge signifikant unterbieten, was die Download-Größe, den benötigten Plattenplatz und den Bedarf an Hauptspeicher betrifft. Wünschenswert sind hier Werte, die im Bereich dessen liegen, was die didaktischen Programmierwerkzeuge aus Abschnitt 3.6 (mit Ausnahme der beiden auf Eclipse basierenden Lösungen) voraussetzen.

Hinzu kommt die Tatsache, dass gerade Studierende der Informatik sehr experimentierfreudig bezüglich der Wahl ihrer Betriebssysteme sind. Verschiedene Versionen von Windows, Linux/Unix und MacOS sind eher die Regel als die Ausnahme. Damit die Werkzeuge von möglichst allen Studierenden genutzt werden können, müssen sie auch auf dieser heterogenen Rechnerlandschaft lauffähig, mithin also weitgehend plattformunabhängig sein. Dies läßt sich derzeit am besten durch den Einsatz von Java erreichen.

### 4.1.4 Entwicklung

Die Werkzeuge werden im universitären Umfeld entwickelt und kommen auch vornehmlich dort zum Einsatz. Sowohl wissenschaftliche Mitarbeiter als auch studentische Hilfskräfte sind aber meist nur in der Lage, einen geringen Teil ihrer Zeit in Entwicklungs- oder Wartungsarbeiten zu investieren. Insofern ist es wichtig, dass die Werkzeuge den begrenzten personellen Kapazitäten im universitären Umfeld Rechnung tragen. Die Entwicklung und Wartung muss also mit möglichst geringem Aufwand verbunden sein. Dazu bietet sich die Verwendung des Meta-CASE-Ansatzes [Ald91] an.

Der Kerngedanke von Meta-CASE ist das Vorhandensein eines generischen graphischen Editors, der an die abstrakte und die konkrete Syntax einer Modellierungssprache angepasst werden kann. Die Anpassung kann händisch oder automatisiert geschehen. Zumindestens im letzteren Fall wird eine formale Spezifikation der Modellierungsmethode (meist, wie in Abschnitt 2.6 beschrieben, auf der Basis von ER-Diagrammen oder Graphgrammatiken) und eventueller weiterer Details des Werkzeugs benötigt. Es existieren vier grundsätzliche Varianten von Meta-CASE, sie sich in der technischen Realisierung und ihren Vor- und Nachteilen unterscheiden:

- **Adaption** – Ein CASE-Werkzeug ist in begrenztem Umfang an Spezialisierungen einer Notation anpassbar. Die Elemente der konkreten Syntax können geändert werden. Die abstrakte Syntax kann durch zusätzliche Randbedingungen verfeinert werden.

- **Interpretation** – Ein generisches CASE-Werkzeug wird zur Startzeit oder zur Laufzeit mit einer Spezifikation der Modellierungsmethode parametrisiert. Die verschiedenen Komponenten des Werkzeugs passen sich automatisch an die Spezifikation an.
- **Generation** – Ein Meta-CASE-Werkzeug wird mit der Spezifikation einer Modellierungsmethode parametrisiert und erzeugt auf dieser Basis ein nutzbares CASE-Werkzeug (in Form von ausführbarem Code). Die Parametrisierung findet hier zur Entwicklungszeit statt.
- **Framework** – Ein (objektorientiertes) Meta-CASE-Framework kapselt die Grundfunktionalität eines CASE-Werkzeugs. Über definierte Erweiterungspunkte wird aus dem generischen Framework mit Hilfe von Vererbung und Delegation ein konkretes Werkzeug erzeugt.

Adaptierbare Werkzeuge sind für den Zweck dieser Arbeit nicht ausreichend, da damit nur leichte Varianten einer Notation, aber keine vollständig neuen Notationen unterstützt werden. Interpreter scheiden aufgrund des möglichen Performanzproblems aus. Sowohl bei Interpretern als auch bei Generatoren besteht die Gefahr der mangelnden Flexibilität, wenn sie ausschließlich über eine formale Spezifikation parametrisiert werden und die Integration zusätzlicher Funktionalität nicht oder nur aufwendig möglich ist. Damit scheint ein Framework der geeignetste Ansatz für diese Arbeit zu sein, zumal es möglich ist, dieses zu einem späteren Zeitpunkt durch einen Generator zu ergänzen.

## 4.2 Überlegungen zur didaktisch motivierten Funktionalität

Neben der Leichtgewichtigkeit war das zweite zentrale Element des Anforderungskatalogs aus Abschnitt 3.7 die gezielte Unterstützung der Lehre durch geeignete Funktionalität. Die dort beschriebenen Programmierumgebungen haben Visualisierungen von Programm- und Laufzeitstrukturen sowie interaktive Experimente verwendet, um Syntax und Semantik von Java zu lehren. Abschnitt 2.7 hat mit dem Verständnis von Syntax, Semantik, Pragmatik und Abstraktion insgesamt vier Problembereiche bei der Lehre von graphischen Modellierungssprachen aufgezeigt, die von entsprechender Funktionalität profitieren können. Die im Rahmen dieser Arbeit entwickelten Werkzeuge sollen diese wie folgt unterstützen:

- **Hypertext** – Die Werkzeuge verwenden Hypertext zur Integration von Lehrmaterial (Vorlesungsstoff, Übungsaufgaben, Dokumentation etc.). Dieser Hypertext wird insbesondere dazu genutzt, Syntax und Semantik der einzelnen Elemente einer Modellierungssprache sowie die Pragmatik des gesamten Ansatzes zu erläutern, was bei herkömmlichen CASE-Werkzeugen nicht der Fall ist. Eine enge Verzahnung von Modell, Werkzeug und Lehrmaterial bietet zudem Möglichkeiten zur Manipulation eines Modells durch den Hypertext.

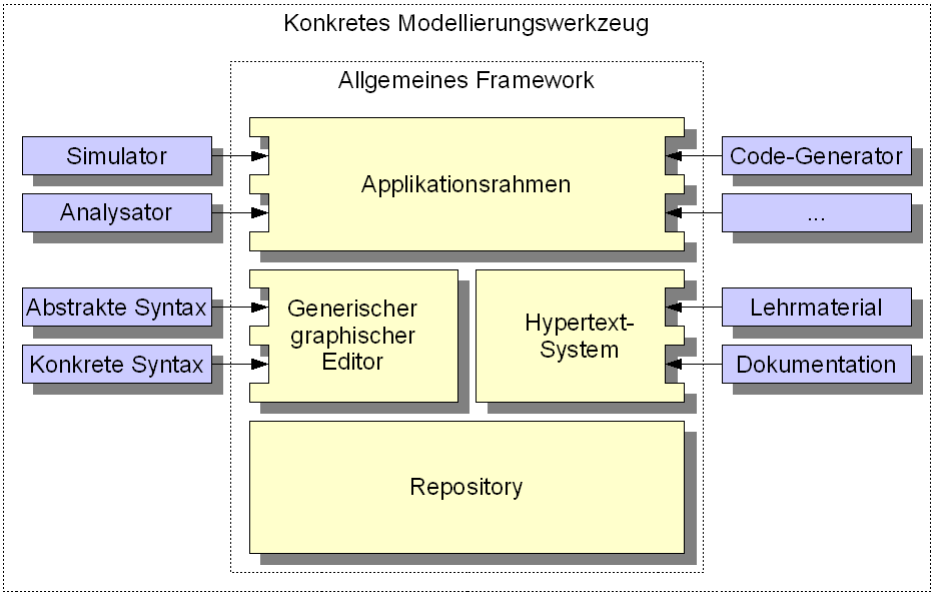



Abbildung 4.1: Überblick der Lösungsarchitektur

- **Simulation und Analyse** – Die Werkzeuge verwenden Simulationen und Analysen als interaktive Experimente zur Verdeutlichung der Sprachsemantik. Hierzu wird die Instanzebene (innerhalb der UML-Architektur die Ebene M0, siehe Abschnitt 2.6.3) zum Bestandteil des Werkzeugs, was bei herkömmlichen CASE-Werkzeugen nicht der Fall ist. Es sei angemerkt, dass sich Simulationen sowohl bei dynamischen als auch bei strukturellen Modellen anbieten. Im ersteren Fall werden konkrete Abläufe durchgespielt, im letzteren Fall erhält der Benutzer die Möglichkeit, konkrete Strukturen aufzubauen.
- **Visualisierung** – Zum Überwinden der Kluft zwischen dem abstrakten Modell und der Realität werden anschauliche Visualisierungen dieser Realität verwendet. Eine Visualisierung kann mit einem Modell verbunden werden, so dass die Studierenden während einer Simulation das Verhalten des Modells und das des realen Systems parallel studieren können. Visualisierungen bieten zudem eine gute technische Basis für Übungsaufgaben mit hohem Motivationswert.

## 4.3 Überlegungen zur Architektur

Abbildung 4.1 gibt einen groben Überblick über die Architektur der angestrebten Lösung und zeigt dabei insbesondere, welcher Teil durch das Framework und welcher durch einzelne Werkzeuge abgedeckt wird.

Den Kern des Frameworks bildet die Infrastruktur zur Verwaltung von Modellen, in anderen Ansätzen meist als Repository bezeichnet. Das Repository hält ein Modell im Speicher und ist in der Lage, es in eine XML-basierte Datei zu sichern oder aus einer solchen Datei zu laden. Auf dem Repository setzen die beiden zentralen Komponenten eines jeden Modellierungswerkzeugs auf:

- Ein syntaxgesteuerter graphischer Editor bietet die nötige Funktionalität zur Darstellung und Bearbeitung von diagrammatischen Modellen. Der Editor wird über die abstrakte und konkrete Syntax einer Modellierungssprache parametrisiert.
- Ein Hypertext-System bietet die Möglichkeit, ein Werkzeug mit Dokumentation oder Lehrmaterial zu füllen. Es ermöglicht außerdem die Verknüpfung eines Modells mit einzelnen Seiten dieses Hypertextes.

Editor und Hypertext-System sind in einen lauffähigen Applikationsrahmen integriert, der ebenfalls durch das Framework bereitgestellt wird. Dieser Rahmen enthält bereits eine weitgehend vollständige Benutzungsschnittstelle, die für ein gemeinsames Erscheinungsbild der einzelnen Werkzeuge sorgt und damit dem Gedanken der Produktfamilie und den sich daraus ergebenden Vorteilen Rechnung trägt. Zusätzliche Funktionalität, die spezifisch für ein Werkzeug oder eine Modellierungssprache ist, wird auf dieser Ebene hinzugefügt. Dabei kann es sich zum Beispiel um die in Abschnitt 4.2 erwähnten Simulations- oder Analysemöglichkeiten handeln, die der Verdeutlichung der Semantik einer Sprache dienen, oder Generatoren, die ein Modell in Quellcode transformieren.

Die Zweiteilung der Lösung hat zur Folge, dass einige der in den Abschnitten 4.1 und 4.2 formulierten Anforderungen bereits im generischen Framework Berücksichtigung finden, andere, insbesondere jene zur Unterstützung der Lehre, erst in konkreten Werkzeugen. Der folgende Teil II der Arbeit beschreibt zunächst den Aufbau des Frameworks. Ein Katalog konkreter Werkzeuge ist Bestandteil von Teil III der Arbeit. Eine kleine Vorschau auf die Werkzeuge bietet Abbildung 4.2. Diese zeigt einige Vertreter der Werkzeugfamilie, die in den späteren Kapiteln detailliert vorgestellt werden.

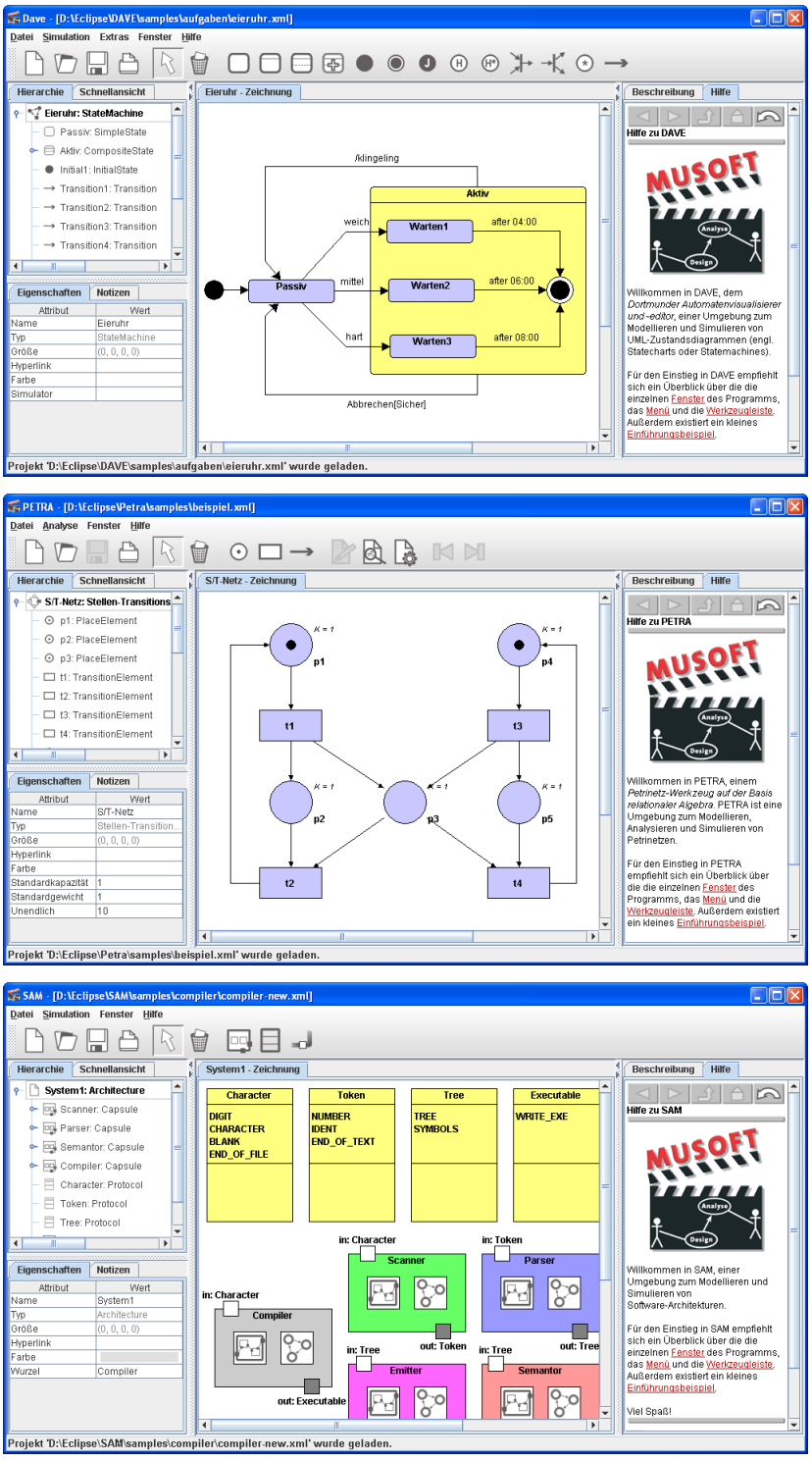



Abbildung 4.2: Einige der in der Arbeit entstandenen Werkzeuge

# Teil II

# Das LiMo-Framework

# 5 Überblick

Dieser Teil der Arbeit stellt das Lightweight Modeling Framework (LiMo) vor, ein in Java realisiertes Framework zur Entwicklung graphischer Modellierungswerkzeuge. Da das Framework dem Bau von CASE-Werkzeugen [Som04, Fug93, Nag93] dient, ist es selbst der Kategorie der Meta-CASE-Frameworks [Ald91] zuzurechnen. Der Name reflektiert die Zielsetzung, dass mit dem Framework keine vollwertigen professionellen Werkzeuge, sondern leichtgewichtige Werkzeuge für die softwaretechnische Lehre an Schulen und Hochschulen erstellt werden sollen.

## 5.1 Berücksichtigte Anforderungen

Das Framework orientiert sich an den Anforderungen, die in Teil I der Arbeit gesammelt wurden. Zu den wesentlichen funktionalen Anforderungen gehörte die Unterstützung der dort identifizierten, aus Sicht der Lehre essenziellen Untermenge des Funktionsumfangs professioneller Werkzeuge (im Wesentlichen das Anzeigen, Bearbeiten, Laden, Speichern und Drucken von Modellen) sowie die Möglichkeit der Integration von Werkzeug und (z.B. textuellem) Lehrmaterial.

Bei den nichtfunktionalen Anforderungen spielte die Bedienbarkeit eine zentrale Rolle, also die Gestaltung der Benutzungsschnittstelle derart, dass die aus dem Framework gewonnenen Werkzeuge für die Zielgruppe möglichst intuitiv bedienbar sind. Dies wird im wesentlichen durch eine in der Komplexität reduzierte Benutzungsschnittstelle erreicht. Gleichzeitig bleibt diese Benutzungsschnittstelle aber strukturell nahe an der professioneller Werkzeuge, um einen späteren Umstieg zu erleichtern.

Eine weitere nichtfunktionale Anforderung ergab sich aus der Tatsache, dass die auf dem Framework basierenden Werkzeuge mutmaßlich im akademischen Umfeld entstehen, in dem die personellen Kapazitäten zur Softwareentwicklung und -wartung naturgemäß begrenzter sind als in einem Unternehmen: die Entwicklung und Wartung der Werkzeuge sollte mit möglichst geringem Aufwand verbunden sein. Das Framework unterstützt dies, indem es bereits sehr umfangreiche (aber nicht überflüssige) generische Funktionalität bereitstellt, die von allen potenziellen Modellierungswerkzeugen benötigt wird.

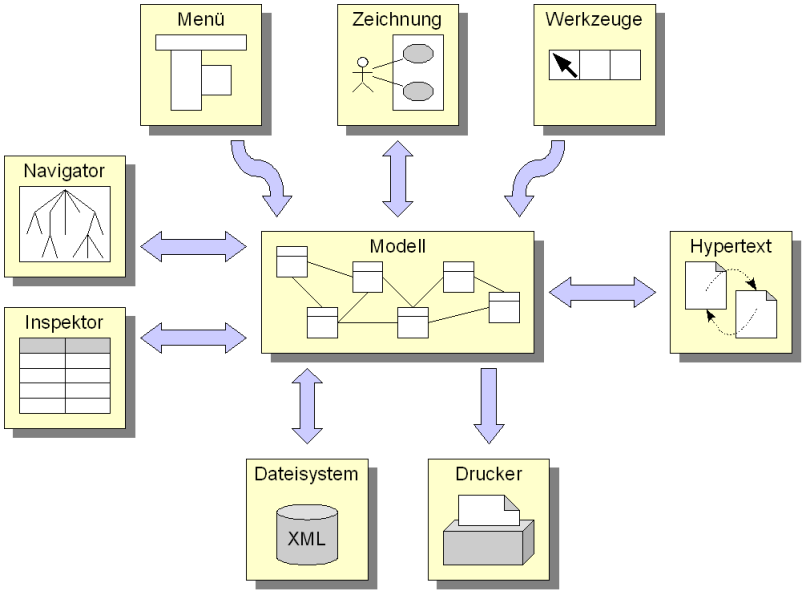



Abbildung 5.1: Funktionalität einer LiMo-Anwendung

## 5.2 Funktionalität einer LiMo-Anwendung

Eine LiMo-Anwendung setzt sich aus drei wesentlichen Komponenten zusammen: Dem eigentlichen Modell, das vom Benutzer erstellt wird und das gleichzeitig die zentrale Datenstruktur der Anwendung darstellt, einer Benutzungsschnittstelle, die verschiedene Sichten auf dieses Modell anbietet, und einem Hypertext-System, das mit Lehrstoff und Dokumentation gefüllt werden kann. Abbildung 5.1 verdeutlicht dies, wobei die Pfeile den Datenfluss zwischen den Komponenten repräsentieren.

Die primäre Sicht auf das Modell ist die Zeichnung, die eine diagrammatische Repräsentation des Modells enthält. Die Zeichnung besteht aus graphischen Figuren (meist geschlossenen Linienzügen mit Flächeninhalt, also z.B. Rechtecken, Ellipsen etc.) und Verbindungen (offenen Linienzügen zwischen zwei Figuren). Figuren und Verbindungen bilden einen Graphen. Die Knoten dieses Graphen können geschachtelt werden, d.h. Figuren können andere Figuren enthalten.

Der Benutzer manipuliert die Zeichnung im wesentlichen unter Zuhilfenahme der Maus. Die Zeichnung gibt dabei Feedback über die syntaktische Korrektheit von Änderungen, indem die beteiligten Figuren und Verbindungen farblich hervorgehoben werden. Änderungen, die zu syntaktisch inkorrekten Modellen führen würden, werden von der Benutzungsschnittstelle abgelehnt.

Weitere Sichten auf das Modell sind der Inspektor und der Navigator, die jeweils spezielle Aspekte des Modells betonen. Auch das Hypertext-System stellt in Teilen eine Sicht auf

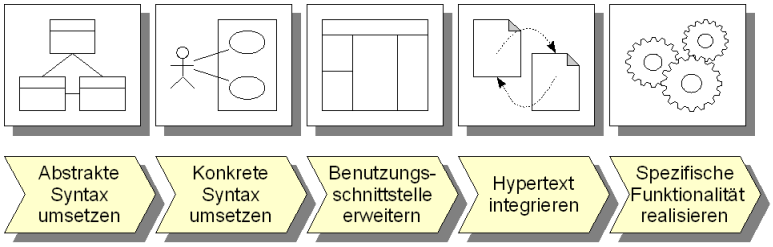



Abbildung 5.2: Methodologie beim Einsatz des LiMo-Frameworks

das Modell dar. Komplettiert wird die Benutzungsschnittstelle durch eine Werkzeugleiste und eine Menüleiste, welche dem Benutzer häufiger bzw. seltener benötigte Operationen zur Verfügung stellen. Die Werkzeugleiste enthält üblicherweise eine Palette der Modellelemente, die in der Zeichnung verwendet werden können.

Modelle können gespeichert und zu einem späteren Zeitpunkt wieder geladen werden, wobei ein einfaches, XML-basiertes Dateiformat zum Einsatz kommt. Modelle können außerdem mit wählbaren Skalierungsfaktoren und Ausrichtungen gedruckt werden. Die Übernahme von Modellen in andere Dokumente – etwa einen Text, eine Präsentation oder eine Webseite – wird von einer LiMo-Anwendung durch die Möglichkeit des Exports in verschiedene Grafikformate unterstützt.

## 5.3 Erweiterungspunkte und Methodologie des Frameworks

Ein objektorientiertes Framework ist ein auf eine bestimmte Problem- oder Anwendungsklasse spezialisierte Menge von abstrakten und konkreten Klassen sowie deren Schnittstellen [WBJ90]. Es besitzt nach [Pre97] eine Menge von klar definierten Punkten (sog. *hot spots*), an denen ein Entwickler das Framework mit Hilfe von Vererbung und Delegation erweitern muss, um daraus eine lauffähige Anwendung abzuleiten. Die beiden zentralen Erweiterungspunkte, die vom Entwickler zu füllen sind, um von dem LiMo-Framework zu einem lauffähigen Modellierungswerkzeug zu gelangen, ergeben sich naturgemäß aus dem wesentlichen Unterschied zwischen den einzelnen Werkzeugen, der graphischen Notation: Dem Framework müssen die abstrakte und die konkrete Syntax der Modellierungssprache bekanntgemacht werden. Hieraus ergibt sich auch gleichzeitig, wie Abbildung 5.2 andeutet, eine Art Methodologie für die Nutzung des Frameworks:

1. Die abstrakte Syntax der Sprache wird „bereitgestellt“. In dieser Arbeit wird Metamodellierung [Küh06] auf der Basis von UML-Klassendiagrammen zur Definition der abstrakten Syntax verwendet. Das Metamodell wird mithilfe einiger Java-Klassen implementiert, die Basisklassen des Frameworks erweitern.

2. Die konkrete Syntax der Sprache wird „bereitgestellt“. Jedes Element der konkreten Syntax wird in einer eigenen Java-Klasse implementiert. Zudem wird eine Abbildung von der abstrakten in die konkrete Syntax realisiert. Hier greift das Framework auf die Bibliothek JHotDraw [Kai04, JHD06] zurück, welche dem Bau von graphischen Editoren dient und über einen Fundus kombinierbarer Figuren (Rechtecke, Ellipsen etc.) verfügt.
3. Die für den Benutzer verfügbaren Elemente der Sprache werden in der Benutzungsschnittstelle bekannt gemacht. Es ist möglich, weitere Anpassungen vorzunehmen. Dies ist aber meist nicht nötig, da sich weite Teile der Benutzungsschnittstelle automatisch an das Metamodell der Sprache anpassen.
4. Der Lehrstoff – etwa eine Beschreibung der Sprachelemente oder Aufgaben, die mit dem Werkzeug zu lösen sind – wird integriert. LiMo verwendet Hypertext auf Basis der Hypertext Markup Language (HTML) [W3C99a] zur Beschreibung des Lehrstoffs und zu dessen Verknüpfung mit dem Modell und dessen Metamodell.
5. Zusätzliche Funktionalität, die gänzlich abhängig von der jeweiligen Modellierungssprache oder dem spezifischen Zweck des Werkzeugs ist, wird realisiert. Dabei kann es sich etwa um die in Abschnitt 4.2 diskutierten Simulations- oder Analysemöglichkeiten handeln, die dem Benutzer Experimente mit dem Modell erlauben.

Die Schritte 1 bis 3 sind für jedes Werkzeug erforderlich, während die Schritte 4 und 5 eher optionalen Charakter haben.

## 5.4 Weitere Gliederung

Die nachfolgenden Abschnitte 6, 7 und 8 beschreiben die drei zentralen Komponenten des LiMo-Frameworks – Metamodell, Benutzungsschnittstelle und Hypertext – im Detail. Das Metamodell setzt die abstrakte Syntax einer Sprache um. Die konkrete Syntax findet sich im graphischen Editor wieder, der zur Benutzungsschnittstelle zählt. Metamodell und Modell realisieren das Subsystem, das in der Architekturskizze in Abschnitt 4.3 als Repository bezeichnet wurde.

Als durchgängiges Beispiel zur Illustration dient ein einfaches Modellierungswerkzeug für eine Untermenge von UML-Anwendungsfalldiagrammen. Abschnitt 9 vergleicht das Framework mit anderen Arbeiten. Tatsächliche Werkzeuge, die mit LiMo realisiert wurden, finden in Teil III der Arbeit Berücksichtigung.

# 6 Metamodell

Das LiMo-Framework verwendet Metamodellierung zur Umsetzung der abstrakten Syntax einer Modellierungssprache. Die Syntax der Modellierungssprache wird also mit den Mitteln eines Modells beschrieben. Das Vorgehen setzt sich dabei aus zwei Schritten zusammen:

1. Das Metamodell wird auf der Basis eines UML-Klassendiagramms spezifiziert.
2. Das Metamodell wird durch eine Menge von Java-Klassen implementiert.

Für den ersten Schritt wurden verschiedene Alternativen in Erwägung gezogen. Jenseits der Metamodellierung hätte sich etwa die Verwendung von Graphgrammatiken und Graphtransformationen angeboten. Diese sind jedoch mit einer hohen Komplexität behaftet, die sich sowohl auf die Entwicklung als auch die Nutzung des Frameworks ausgewirkt hätte (siehe Abschnitt 2.6). Eine Diplomarbeit [Pet05] hat die Einsatzmöglichkeiten der (wohl bekanntesten) Graphtransformations-Maschine AGG [ERT99] als Basis des LiMo-Frameworks untersucht und bestätigt das Komplexitätsproblem. Gleichzeitig erwies sich AGG in der Diplomarbeit als zu fehlerbehaftet und instabil, um auf dieser Basis produktive Werkzeuge zu realisieren.

Ein alternativer Formalismus zur Metamodellierung wären ER-Diagramme oder eine Variante davon gewesen. Diese schieden jedoch aus rein pragmatischen Gründen aus: Der hohe Bekanntheitsgrad der UML und die Verfügbarkeit von entsprechenden Werkzeugen sprechen für Klassendiagramme, ebenso wie die Tatsache, dass auf Klassendiagrammen basierende Metamodelle für viele Sprachen – insbesondere die einzelnen Teilsprachen der UML [OMG03b] – bereits existieren und mit geringem Aufwand übernommen werden können.

Die im zweiten Schritt durchgeführte Implementierung geschieht derzeit händisch, ist aber ein sehr geradliniger Prozess. Eine wenigstens teilweise Unterstützung durch generative Werkzeuge wäre möglich, war aber nicht Ziel dieser Arbeit.

## 6.1 Spezifikation des Metamodells

Das Metamodell einer Modellierungssprache entsteht, indem die syntaktischen Konzepte der Sprache durch Klassen repräsentiert werden. Die (potenziellen) Zusammenhänge zwischen den Elementen werden durch Assoziationen und deren Multiplizitäten beschrieben. Attribute dienen – wie üblich – zur Aufnahme von Informationen mit eher „lokalem“

Charakter und primitivem Datentyp[1]. Konstrukte wie Vererbung und Abstraktion können zur Strukturierung des Metamodells genutzt werden. Zur Wiedergabe syntaktischer Einschränkungen, die das Klassendiagramm alleine nicht auszudrücken vermag, bietet sich zum Beispiel die OCL [WK95] an.

Vier spezielle Stereotypen dienen zur Aufteilung der auf diese Weise entstehenden Klassen in disjunkte Mengen und geben erste Hinweise auf die spätere Abbildung von der abstrakten in die konkrete Syntax. Der Stereotyp `«figure»` deutet an, dass eine Klasse durch eine Figur repräsentiert werden soll. Klassen mit dem Stereotyp `«connection»` werden auf eine Verbindung abgebildet. Der Stereotyp `«model»` zeichnet eine Klasse als Wurzel eines Modells aus. Er kann mehrfach verwendet werden, um auszudrücken, dass sich ein Modell aus verschiedenen Teilmodellen zusammensetzt. Mit dem Stereotyp `«nonvisual»` werden Klassen gekennzeichnet, die nicht in die konkrete Syntax abgebildet werden.

Über die Verwendung der Stereotypen hinaus existieren folgende Besonderheiten an dem Klassendiagramm, welches das Metamodell bildet:

- Komposition wird genutzt, um die Zugehörigkeiten von Modellelementen zu beschreiben. Dabei gilt die übliche Kompositionssemantik (Teil-Ganzes-Beziehung, Existenzabhängigkeit) mit den bekannten Einschränkungen (Azyklizität, Exklusivität). Der allgemeine Graph des Modells erhält so ein baumartiges Gerüst, das möglicherweise mit weiteren Assoziationen verfeinert wird.
- Die Sichtbarkeiten `private`, `protected` und `public` werden minimal anders interpretiert als üblich: `private` kennzeichnet eine Eigenschaft, die für den Benutzer nicht sichtbar ist. Eigenschaften mit der Sichtbarkeit `protected` sind für den Benutzer sichtbar, aber nicht änderbar. Eigenschaften mit der Sichtbarkeit `public` können vom Benutzer geändert werden[2].
- Die Liste der üblichen primitiven Datentypen wird für den gegebenen Kontext pragmatisch erweitert. Als primitive Datentypen stehen zur Verfügung: `Boolean`, `Integer`, `String`, `Float`, `Color` (ein RGB-Farbwert), `Date` (Datum und Uhrzeit) und `Resource` (ein Uniform Resource Identifier (URI) [IET05], also ein Verweis auf eine Datei, Webseite etc.). Außerdem können Aufzählungstypen verwendet werden.

Abbildung 6.1 zeigt ein entsprechendes Metamodell für die einfache Untermenge von UML-Anwendungsfalldiagrammen, die in diesem und den folgenden Abschnitten als durchgängiges Beispiel dient.

[1] Die Frage, wann ein syntaktisches Element auf eine Klasse abgebildet und wann ein Attribut verwendet wird, ist nicht immer eindeutig zu beantworten. Da aber die Metamodellierung letztlich auf eine Art Problembereichsmodell der Modellierungssprache hinausläuft, ist es nicht verwunderlich, dass in ihrem Rahmen die gleichen Schwierigkeiten auftreten wie bei der Nutzung der UML zur „gewöhnlichen“ Softwareentwicklung.

[2] Die Nutzung neuer Sichtbarkeiten oder Stereotypen wäre an dieser Stelle sicher konsequenter gewesen, darunter hätte aber die Übersichtlichkeit des Klassendiagramms gelitten. Die Lösung stellt letztlich ein Zugeständnis an die (begrenzten) Möglichkeiten existierender CASE-Werkzeuge dar.

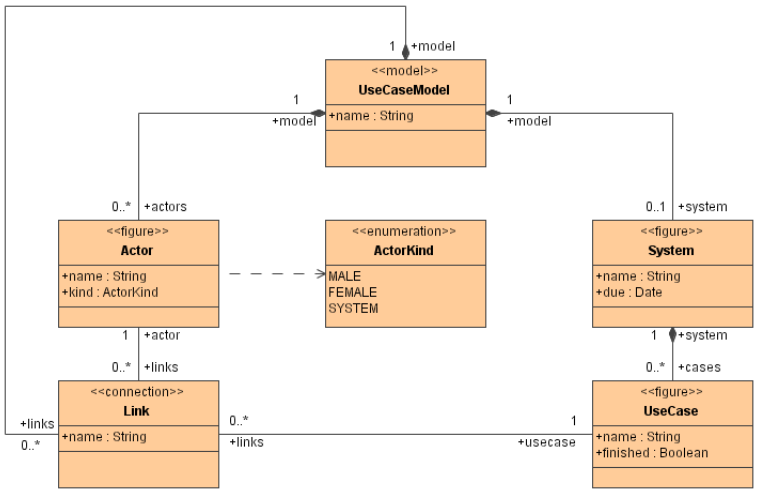



Abbildung 6.1: Metamodell für eine Untermenge von Anwendungsfalldiagrammen

Das Metamodell enthält zunächst eine Klasse zur Repräsentation der Wurzel des Modells (Klasse `UseCaseModel`) und eine weitere zur Repräsentation des eigentlichen Systems (Klasse `System`). Die restlichen Klassen repräsentieren Akteure (Klasse `Actor`), einzelne Anwendungsfälle (Klasse `UseCase`) und die Beziehung zwischen letzteren (Klasse `Link`). Alle Klassen verwenden die beschriebenen Stereotypen. Das System enthält die einzelnen Anwendungsfälle und ist selbst Teil des Modells. Akteure stehen außerhalb des Systems, sind also ebenfalls der Wurzel des Modells untergeordnet.

Die Klassen wurden zudem mit einigen (teilweise sinnfreien) Attributen versehen, die ausschließlich die Nutzung der verschiedenen Datentypen verdeutlichen sollen. Alle Klassen sehen einen Namen vor (Attribut `name`). Für das System kann festgelegt werden, bis zu welchem Zeitpunkt es fertiggestellt sein muss (Attribut `due`), für jeden Anwendungsfall, ob dieser bereits realisiert ist (Attribut `finished`). Akteure können in männliche und weibliche Nutzer sowie externe Systeme unterschieden werden (Attribut `kind`), wobei eine Aufzählung zur Spezifikation der möglichen Werte dient (Aufzählungstyp `ActorKind`).

## 6.2 Implementierung des Metamodells

Um das Metamodell einer Modellierungssprache im LiMo-Framework nutzen zu können, muss es zunächst auf Java-Klassen abgebildet – also implementiert – werden. Dabei tritt das Problem auf, dass nicht zu allen Elementen eines UML-Klassendiagramms unmittelbare Entsprechungen in Java existieren. Prominentes Beispiel sind hier die Assoziationen, die sich in Java zwar prinzipiell durch wechselseitige Objektreferenzen realisieren lassen,

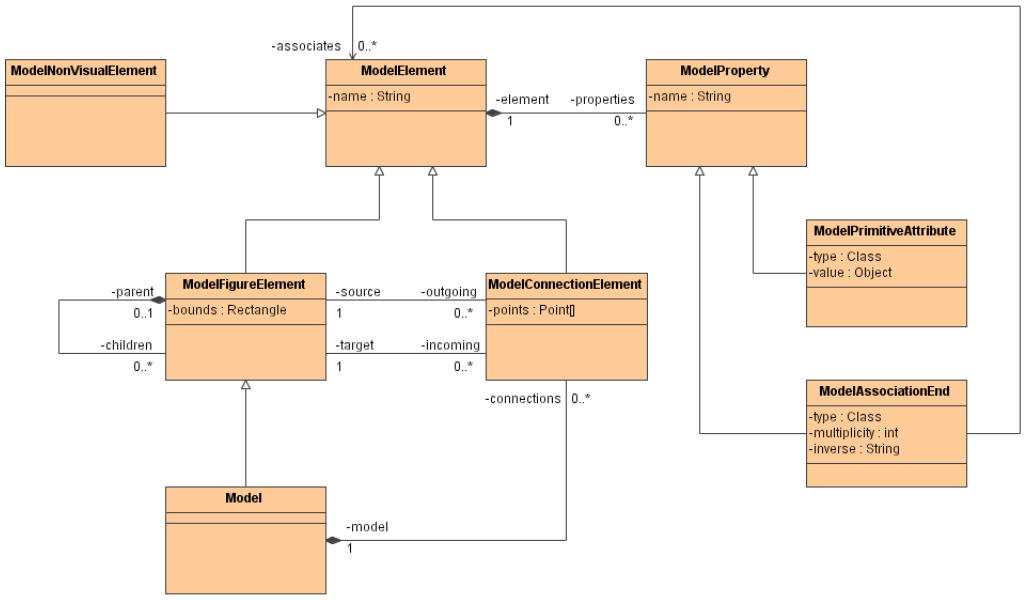



Abbildung 6.2: Klassen des Metamodellkerns von LiMo

deren Konsistenz und Einhaltung von Multiplizitäten aber nicht ohne zusätzlichen Code sicherzustellen ist.

Als Lösung bietet das Framework eine Menge von Java-Klassen zur Repräsentation von Metamodellen an, den sogenannten Metamodellkern. Anschaulich betrachtet bildet dieser Metamodellkern einen Baukasten von Elementen, die sich leicht zu anwendungsspezifischen Metamodellen kombinieren lassen. Konzeptionell verwendet er Ideen, die auch in der UML-Superstruktur [OMG03b], der Meta Object Facility (MOF) [OMG06] oder anderen Ansätzen zur Metamodellierung zu finden sind. Es ist jedoch weder eine Implementierung eines dieser Ansätze, noch handelt es sich dabei um ein Meta-Metamodell im strikten Sinne. Dies liegt daran, dass die Elemente eines spezifischen Metamodells von denen des Metamodellkerns abgeleitet werden statt diese zu instanziieren. Im Vergleich zur UML-Architektur (siehe Abschnitt 2.6.3) reduziert sich damit die Anzahl der Modellebenen in der LiMo-Architektur auf drei statt vier.

### 6.2.1 Umsetzung der Klassen

Abbildung 6.2 zeigt ein Klassendiagramm der Elemente des Metamodellkerns. Die Klasse `ModelElement` stellt die abstrakte Grundlage zur Repräsentation aller Klassen eines Metamodells dar. Sie besitzt ein Attribut zur Benennung jeder Klasse des Metamodells. Die Klasse `ModelFigureElement` bietet eine konkrete Basis für Figurelemente. Sie speichert zusätzlich die Position und die Abmessungen der Figur. Die Klasse `ModelConnectionElement`

stellt die Grundlage für Verbindungselemente bereit. Sie speichert zusätzlich ein Array von Punkten, die von der Verbindung berührt sind. Die Klasse `Model` repräsentiert die Wurzel eines Modells, und die Klasse `ModelNonVisualElement` wird für alle sonstigen Elemente des Metamodells verwendet.

Zwischen `ModelFigureElement` und `ModelConnectionElement` bestehen zwei Assoziationen, die Start und Ziel der einzelnen Verbindungen festlegen, also die Modellelemente zu einem Graphen vernetzen. Die Klasse `ModelFigureElement` steht zudem in einer Eltern-Kind-Beziehung zu sich selbst. Dies erlaubt die Schachtelung von Figuren und dient zur Realisierung der Kompositionsbeziehungen innerhalb des Metamodells. Da die Klasse `Model` von `ModelElement` erbt, können auch komplette Modelle innerhalb von Modellelementen enthalten sein. Dies erlaubt die Unterstrukturierung komplexer Modelle in Teilmodelle und wird z.B. innerhalb der Benutzungsschnittstelle ausgenutzt, um ein Modell in mehrere Diagramme aufzuteilen.

### 6.2.2 Umsetzung der Attribute und Assoziationen

Jedes Element des Metamodellkerns besitzt einen Namen sowie eine Liste von benannten Eigenschaften, die durch die abstrakte Klasse `ModelProperty` repräsentiert werden. Konkrete Ausprägungen davon sind die Klassen `ModelPrimitiveAttribute` zur Repräsentation von primitiven Attributen und `ModelAssociationEnd` zur Repräsentation von Assoziationsenden. Ein primitives Attribut besitzt einen der in Abschnitt 6.1 beschriebenen primitiven Datentypen (repräsentiert durch die jeweils gleichnamige Java-Klasse) und einen entsprechenden (möglicherweise leeren) Wert. Ein Assoziationsende besitzt als Typ die Klasse eines anderen Modellelements sowie als Wert eine (gegebenenfalls ebenfalls leere) Menge von assoziierten Objekten, deren maximaler Umfang sich nach der Multiplizität des Assoziationsendes richtet.

Die Basisklasse `ModelElement` erzeugt bereits derartige Repräsentationen dreier primitiver Attribute, die damit automatisch in allen Modellelementen zur Verfügung stehen:

- Das Attribut `Color` speichert die Farbe eines Modellelements. Dies wird in der Zeichnung als Füllfarbe (bei Figuren) bzw. als Linienfarbe (bei Verbindungen) genutzt.
- Das Attribut `Note` speichert Notizen zu einem Modellelement. Diese sollen zur Dokumentation des Modellelements dienen. Das Attribut wird vom Hypertext-System genutzt (siehe Abschnitt 8.2).
- Das Attribut `Hyperlink` speichert eine Referenz auf eine externe Ressource, die mit dem Modellelement verknüpft ist. Das Attribut wird ebenfalls vom Hypertext-System genutzt (siehe Abschnitt 8.2).

Weitere Attribute und Assoziationen eines konkreten Metamodells werden auf Instanzen von `ModelPrimitiveAttribute` und `ModelAssociationEnd` abgebildet, die jeweils im Konstruktor der Java-Klasse zu erzeugen sind. Die Abbildungen 6.3 und 6.4 zeigen die

```
import org.musoft.limo.model.*;

public class UseCaseModel extends Model {
    public UseCaseModel() {
        this(null, null);
    }

    public UseCaseModel(String name, Rectangle bounds) {
        super(name, bounds);

        createAttribute("finished", ModelPrimitiveAttribute.TYPE_BOOLEAN,
            ModelAttribute.ACCESS_READWRITE);
        createComposition("system", System.class, ModelAttribute.ACCESS_READ,
            0, 1, "model");
        createComposition("actors", Actor.class, ModelAttribute.ACCESS_READ,
            0, Integer.MAX_VALUE, "model");
        createComposition("links", Link.class, ModelAttribute.ACCESS_READ,
            0, Integer.MAX_VALUE, "model");
    }
}

public class System extends ModelFigureElement {
    public System() {
        this(null, null, null);
    }

    public System(String name, Rectangle bounds, UseCaseModel parent) {
        super(name, bounds, parent);

        createAttribute("due", ModelPrimitiveAttribute.TYPE_DATE,
            ModelAttribute.ACCESS_READWRITE);
        createAssociation("model", UseCaseModel.class, ModelAttribute.ACCESS_READ,
            0, 1, "system");
        createComposition("usecases", UseCase.class, ModelAttribute.ACCESS_READ,
            0, INTEGER.MAX_VALUE, "system");
    }
}

public class Actor extends ModelFigureElement {
    public Actor() {
        this(null, null, null);
    }

    public Actor(String name, Rectangle bounds, UseCaseModel parent) {
        super(name, bounds, parent);

        String[] values = new String[] {"MALE", "FEMALE", "SYSTEM"};

        createEnumeration("kind", ModelPrimitiveAttribute.TYPE_INTEGER,
            ModelAttribute.ACCESS_READWRITE, values);
        createAssociation("model", UseCaseModel.class, ModelAttribute.ACCESS_READ,
            0, 1, "actors");
        createAssociation("links", Link.class, ModelAttribute.ACCESS_READ,
            0, INTEGER.MAX_VALUE, "actor");
    }
}
```

Abbildung 6.3: Implementierung des Beispiel-Metamodells (Teil 1)

```
public class UseCase extends ModelFigureElement {
    public UseCase() {
        this(null, null, null);
    }

    public UseCase(String name, Rectangle bounds, System parent) {
        super(name, bounds, parent);

        createAttribute("finished", ModelPrimitiveAttribute.TYPE_BOOLEAN,
            ModelAttribute.ACCESS_READWRITE);
        createAssociation("system", System.class, ModelAttribute.ACCESS_READ,
            0, 1, "usecases");
        createAssociation("links", Link.class, ModelAttribute.ACCESS_READ,
            0, Integer.MAX_VALUE, "usecase");
    }
}

public class Link extends ModelConnectionElement {
    public Link() {
        this(null, null);
    }

    public Link(String name, UseCaseModel model) {
        super(name, model);

        createAssociation("model", UseCaseModel.class, ModelAttribute.ACCESS_READ,
            1, 1, "links");
        createAssociation("actor", Actor.class, ModelAttribute.ACCESS_READ,
            1, 1, "links");
        createAssociation("usecase", UseCase.class, ModelAttribute.ACCESS_READ,
            1, 1, "links");
    }
}
```

Abbildung 6.4: Implementierung des Beispiel-Metamodells (Teil 2)

Implementierung des Metamodells für das Beispiel der Anwendungsfalldiagramme. Die Klassen besitzen jeweils einen parametrisierten Konstruktor, der die Objekte zur Repräsentation der Attribute und Assoziationen erzeugt. Für Attribute werden Name, Typ und Sichtbarkeit übergeben. Bei Aufzählungen kommt ein Array der legalen Werte hinzu, bei Assoziationen die Multiplizitäten und der Name der inversen Assoziation. Der zweite, parameterlose Konstruktor wird benötigt, damit die Klassen über den Java-Reflektionsmechanismus (und die Methode `newInstance()`) instanziiert werden können.

## 6.3 Infrastruktur für Syntaxprüfung

Die Elemente des Metamodellkerns stellen bereits verschiedene Methoden bereit, die dem Framework das Überprüfen der Zulässigkeit einer Änderung des Modells erlauben, bevor diese tatsächlich durchgeführt wird. Sowohl `ModelFigureElement` als auch `ModelConnectionElement` besitzen zum Beispiel eine Methode `canConnect()`, die prüft, ob eine gegebene Verbindung zwischen zwei gegebenen Figuren zulässig ist. Erst, wenn

alle drei beteiligten Elemente dieser Änderung zustimmen, wird die Benutzungsschnittstelle diese auch durchführen. Die Standard-Implementierung von `canConnect()` zieht nur die Typen und Multiplizitäten der Assoziationsenden in Betracht. Die Methode kann überschrieben werden, um speziellere Einschränkungen zu berücksichtigen. Das Beispiel-Metamodell für Anwendungsfalldiagramme macht davon jedoch keinen Gebrauch.

## 6.4 Infrastruktur für Beobachtung

Um die verschiedenen Sichten der Benutzungsschnittstelle oder andere Komponenten von Änderungen des Modells zu notifizieren, bietet sich die Nutzung des Beobachtermusters [GHJV95] an. Der Metamodellkern enthält deshalb entsprechende Infrastruktur in Form verschiedener Schnittstellen zur Beobachtung der verschiedenen Arten von Modellelementen. Klassen, die eine oder mehrere dieser Schnittstellen implementieren, können sich bei (Instanzen) der Klasse `ModelElement` (und deren Subklassen) als Beobachter registrieren und werden anschließend von Änderungen unterrichtet.

## 6.5 Infrastruktur für Introspektion

Die Klassen des generischen Metamodells besitzen außerdem zusätzliche Methoden zur Introspektion. Diese geben Auskunft über die Implementierung des Metamodells (auf einem höheren Abstraktionsniveau als es der Reflektionsmechanismus von Java vermocht hätte). Für jedes Modellelement kann die Menge der zugehörigen Attribute und Assoziationsenden erfragt werden. Attribute und Assoziationsenden sind zudem in der Lage, Auskunft über ihre zulässigen Werte zu geben. Diese Methoden werden innerhalb des LiMo-Frameworks zur automatischen Anpassung von Teilen der Benutzungsschnittstelle (siehe auch Abschnitt 7.5) an das Metamodell einer Anwendung genutzt.

## 6.6 Infrastruktur für Persistenz

Die Methoden zur Introspektion bilden auch die Basis für eine generische Persistenz von Modellen. Zum Einsatz kommt ein einfaches, XML-basiertes Dateiformat, dessen Schema sich unmittelbar aus dem jeweiligen Metamodell ableiten lässt: Die Namen der verwendeten XML-Elemente entsprechen denen der Klassen. Die Schachtelungsstruktur ergibt sich aus den Kompositionsbeziehungen. Attribute und Assoziationen des Metamodells werden durch ein Element `<Property>` ausgedrückt, wobei im mehrwertigen Fall ein Unterelement `<Item>` zum Einsatz kommt. Informationen zum Layout von Figuren werden direkt mithilfe von XML-Attributen in die entsprechenden Elemente integriert. Die Stützpunkte von Verbindungen werden durch ein zusätzliches Element `<Point>` ausgedrückt. Das Format ist so beschaffen, dass es für Menschen einfach zu lesen und für mögliche weitere Anwendungen leicht zu verarbeiten ist.

# 7 Benutzungsschnittstelle

Das LiMo-Framework stellt eine fast vollständige Benutzungsschnittstelle bereit, die nur an wenigen Stellen händisch an eine konkrete Anwendung angepasst werden muss. Die Benutzungsschnittstelle orientiert sich strukturell an professionellen Modellierungswerkzeugen, um den Studierenden einen späteren Übergang zu diesen Werkzeugen zu erleichtern. Mit Hinblick auf einen Einsatz im Rahmen der Lehre wurde jedoch Wert darauf gelegt, den Umfang zu begrenzen, um die Benutzungsschnittstelle einfach, intuitiv und möglichst wenig ablenkend zu gestalten.

Abbildung 7.1 zeigt diese Benutzungsschnittstelle am Beispiel des Werkzeugs für Anwendungsfalldiagramme. Der Löwenanteil des Bildschirms wird von der Zeichenfläche mit der graphischen Darstellung des Modells in Anspruch genommen, da dies der eigentliche Fokus der Applikation ist. Sämtliche Aspekte, die unmittelbar die Arbeit am Modell betreffen – etwa das Verschieben, Vergrößern, Verkleinern oder Verbinden von Modellelementen – werden in diesem Bereich und mit Hilfe der Maus gehandhabt. Im linken Drittel der Benutzungsschnittstelle befinden sich Navigator und Inspektor. Ersterer dient zur Orientierung innerhalb größerer Modelle. Letzterer ermöglicht die Anzeige und Bearbeitung der Eigenschaften einzelner Modellelemente. Beide können bei Bedarf ausgeblendet werden, etwa wenn innerhalb einer Vorlesung nur ein Modell präsentiert werden soll.

Zeichenfläche, Navigator und Inspektor stellen Sichten auf das Modell dar, innerhalb derer jeweils unterschiedliche Aspekte betont werden. Die weitere für den Benutzer erreichbare Funktionalität des Frameworks verteilt sich auf eine Werkzeugleiste, die oberhalb der Zeichenfläche angeordnet ist, und eine Menüleiste, die sich am oberen Rand des Fensters befindet. Die folgenden Abschnitte stellen die einzelnen Komponenten im Detail vor.

## 7.1 Die Zeichenfläche

Die Zeichenfläche ist die zentrale Komponente der Benutzungsschnittstelle. Sie dient zum Anzeigen und Bearbeiten einer oder mehrerer Zeichnungen, wobei unter einer Zeichnung im Folgenden eine zweidimensionale, vektororientierte Darstellung eines Modells oder eines Teils eines Modells verstanden werden soll. Eine Zeichnung setzt sich primär aus zwei Arten von Elementen zusammen:

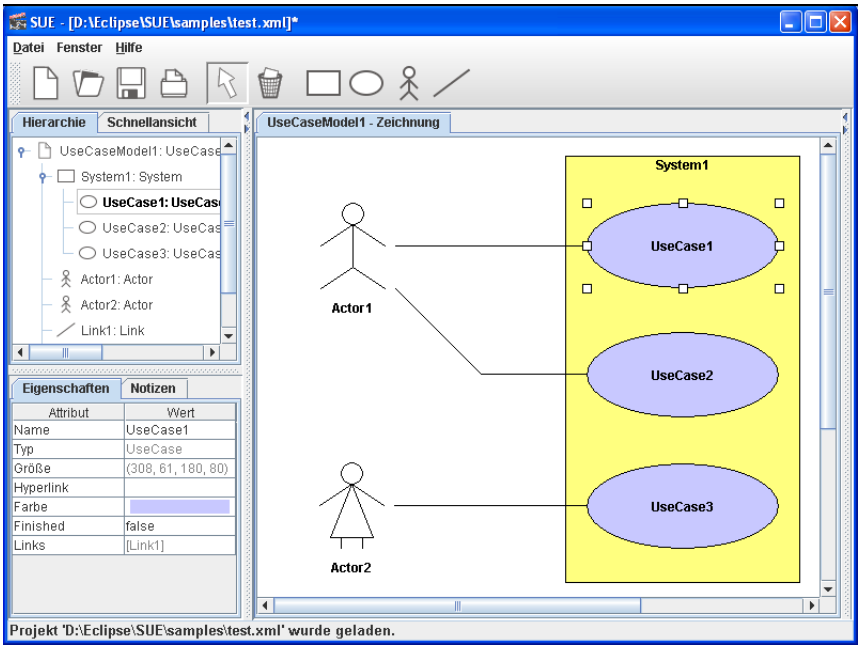



Abbildung 7.1: Ein Werkzeug für Anwendungsfalldiagramme

- **Figuren** – Figuren sind (meist) geschlossene geometrische Figuren mit Flächeninhalt, d.h. Rechtecke, Ellipsen etc. (oder beliebige Kombinationen davon). Es sind jedoch auch komplexere, ikonische Darstellungen auf der Basis von Pixel-Grafiken denkbar.
- **Verbindungen** – Verbindungen sind offene Linienzüge, mit deren Hilfe zwei Figuren in Beziehung gesetzt werden. Eine Verbindung besitzt einen Start- und einen Endpunkt sowie eine – möglicherweise leere – Menge von zusätzlichen Stützpunkten.

Die Figuren und Verbindungen, die innerhalb einer Zeichnung zum Einsatz kommen können, bilden die konkrete Syntax einer Modellierungssprache. Sie sind damit üblicherweise spezifisch für ein Werkzeug. Neben der abstrakten Syntax, die durch das Metamodell der Sprache beschrieben wird, ist die konkrete Syntax der zweite wesentliche Erweiterungspunkt, der zu füllen ist, um aus dem generischen LiMo-Framework eine vollwertige Anwendung zu entwickeln. Für den Entwickler einer LiMo-Anwendung bedeutet das konkret, dass er eine Menge von Java-Klassen bereitstellen muss, welche die einzelnen Figuren implementieren.

Zusätzlich zu abstrakter und konkreter Syntax wird eine Abbildung benötigt, welche die beiden geeignet in Beziehung setzt. Dies geschieht innerhalb des LiMo-Frameworks auf zwei Ebenen:

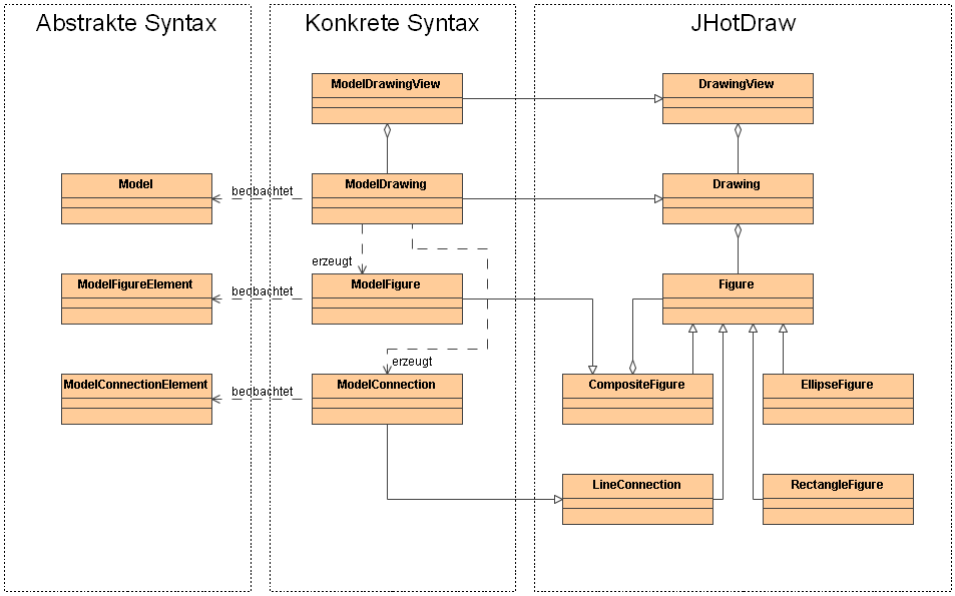



Abbildung 7.2: Zusammenspiel von abstrakter und konkreter Syntax

- Zwischen den Elementen der Zeichnung und den zugehörigen Modellelementen besteht eine Beobachter-Beziehung [GHJV95]. Jede Figur und jede Verbindung wird zum Beobachter des Modellelements, das sie repräsentiert. Bei Änderungen des Modellelements (Name, Position, Größe, Attribute oder Assoziationen) wird die Figur oder Verbindung benachrichtigt und kann ihre graphische Darstellung entsprechend anpassen. Wird das Modellelement zerstört, reagiert die Figur oder Verbindung entsprechend.
- Zwischen der Zeichnung und dem zugehörigen (Teil-) Modell besteht ebenfalls eine Beobachter-Beziehung. Die Zeichnung wird zum Beobachter des Modells. Wird ein neues Element in das Modell aufgenommen, dann erzeugt die Zeichnung die entsprechende Figur. Wird eine Zeichnung für ein bestehendes Modell geöffnet, dann traversiert die neue Zeichnung das Modell und erzeugt für jedes Element die passende Figur. Der Zeichung kommt damit gleichzeitig die Rolle einer Fabrik [GHJV95] für Figuren zu.

Aus der wesentlichen Rolle, die die Zeichnung beim Zusammenspiel von abstrakter und konkreter Syntax spielt, ergibt sich, dass eine LiMo-Anwendung zusätzlich zu den einzelnen Figuren üblicherweise auch eine spezialisierte Zeichnungsklasse verwendet.

Bei der Implementierung der Zeichenfläche war die Verwendung von JHotDraw [JHD06, Kai04] hilfreich, einer Klassenbibliothek zur Entwicklung von vektororientierten graphischen Editoren, die auf eine entsprechende Arbeit im Smalltalk-Umfeld zurückgeht [Joh92]. JHotDraw baut – wie das LiMo-Framework – wesentlich auf dem Begriff einer

```
package org.musoft.sue.figures;

import java.awt.Point;

import org.jhotdraw.figures.EllipseFigure;
import org.musoft.limo.drawing.ModelDrawing;
import org.musoft.limo.drawing.ModelFigure;
import org.musoft.limo.model.ModelElement;
import org.musoft.sue.model.UseCase;

public class UseCaseFigure extends ModelFigure {

    public UseCaseFigure(UseCase element, ModelDrawing drawing) {
        super(element, drawing);

        Point origin = getElement().getPosition();
        Point corner = getElement().getCorner();
        setPresentationFigure(new EllipseFigure(origin, corner));

        setLabelPosition(LABEL_INSIDE | LABEL_CENTER);
        setLabel(getElement().getName());
    }

    public void onSetName(ModelElement sender) {
        super.onSetName(sender);
        setLabel(sender.getName());
    }

}
```

Abbildung 7.3: Implementierung einer Figur für Anwendungsfälle

Zeichnung auf, die sich aus Figuren und Verbindungen zusammensetzt[1]. Um die Klassen der Bibliothek für dem Rest des LiMo-Frameworks nutzbar zu machen, wurde eine zusätzliche Schicht eingefügt, die sich grob als Adapter [GHJV95] klassifizieren lässt und deren zentrale Aufgabe in der Verbindung zwischen abstrakter und konkreter Syntax über das Beobachtermuster besteht (JHotDraw selbst sollte unverändert verwendet werden, obwohl der Quellcode zur Verfügung stand).

Abbildung 7.2 gibt einen groben Überblick über die Klassenstruktur. Ein Teil der Klassen ist bereits diskutiert worden. Zur Realisierung der konkreten Syntax einer Modellierungssprache werden üblicherweise spezialisierte Nachkommen von `ModelDrawing`, `ModelFigure` und `ModelConnection` benötigt. `ModelFigure` ist Nachkomme einer JHotDraw-Klasse zur Repräsentation (möglicherweise) rekursiv zusammengesetzter Figuren, so dass sich die konkrete Syntax an dieser Stelle einer Vielzahl existierender Figuren (Rechtecke, Ellipsen etc.) bedienen kann, die von JHotDraw bereitgestellt werden. `ModelConnection` erweitert die entsprechende Verbindungsklasse `ConnectionFigure` von JHotDraw, für die neben Linienstil und -farbe verschiedene Dekorationen der Enden (Pfeilspitzen etc.) festgelegt werden können. Sowohl für Figuren als auch für Verbindungen besteht die Möglichkeit, diese komplett selbst zu zeichnen, wenn die Anwendung es verlangt.

[1] Tatsächlich wurde die Terminologie teilweise von dort übernommen.

Abbildung 7.3 zeigt die Implementierung der Figur für Anwendungsfälle innerhalb des durchgehenden Beispiels. Der Konstruktor erzeugt eine graphische Darstellung bestehend aus einer beschrifteten Ellipse. Die Methode `onSetName()` ist Teil der Beobachterschnittstelle. Sie passt die Beschriftung an, wenn sich der Name der Figur ändert. Für weitere Details zur Realisierung von Figuren sei auf die Dokumentation von JHotDraw [JHD06] verwiesen.

## 7.2 Die Werkzeugleiste

Die Werkzeugleiste befindet sich unmittelbar oberhalb der Zeichenfläche. Ihr Zweck ist der schnelle Zugriff auf die am häufigsten benötigte Funktionalität, wobei es dem Entwickler einer LiMo-Anwendung obliegt, diese zu definieren.

Die vom Framework vordefinierte Werkzeugleiste enthält bereits Schalter für den grundlegenden Umgang mit Dateien (Erzeugen, Öffnen, Speichern, Drucken) sowie zum Aktivieren des Auswahlwerkzeugs der Zeichenfläche. Eine konkrete LiMo-Anwendung sollte der Werkzeugleiste weitere Schalter, etwa zum Erzeugen der einzelnen Modellelemente, hinzufügen. Jeder Schalter kann (und sollte) mit einem großem, intuitiv verständlichen Symbol versehen werden und kann außerdem einen Erläuterungstext tragen. Die Schalter der Werkzeugleiste können zur Laufzeit einzeln oder in Gruppen aktiviert und deaktiviert werden, wenn der Zustand der Anwendung dies verlangt.

## 7.3 Die Menüleiste

Die Menüleiste, die sich – wie üblich – am oberen Rand des Anwendungsfensters befindet, ermöglicht den Zugriff auf weniger häufig benötigte Teile der Funktionalität des Werkzeugs. Sie enthält zunächst drei Menüs, die allen LiMo-Werkzeugen gemein sind:

- Das Menü „Datei“ bietet die kanonische Funktionalität zum Umgang mit Dateien (Erzeugen, Öffnen, Speichern etc.). Ein Teil dieser Funktionalität ist bereits in der Werkzeugleiste enthalten, wird jedoch aus Gründen der Konsistenz hier dupliziert. Über das Dateimenü kann ein Modell außerdem gedruckt und in verschiedenen Grafikformaten exportiert werden.
- Das Menü „Fenster“ ermöglicht die Änderung einiger Einstellungen, die das Anwendungsfenster und die Zeichnung betreffen: Ein Fangraster kann aktiviert und deaktiviert werden. Eine Anpassung des Erscheinungsbilds der Anwendung an verschiedene Betriebssysteme ist möglich. Außerdem kann die gesamte Anwendung in einen Präsentationsmodus versetzt werden, in dem nur die Zeichnung sichtbar ist.
- Das Menü „Hilfe“ sammelt diverse unterstützende Funktionalität. Hier kann die Einstiegsseite des in einem Werkzeug enthaltenen Hypertextes (siehe Abschnitt 8.1)

aufgerufen werden. Außerdem befindet sich hier der übliche Dialog mit Informationen über die jeweilige Anwendung. Dieser kann neben Name und Versionsnummer des Werkzeugs Hyperlinks zu einer Homepage oder Kontaktadresse enthalten.

Entwickler von spezifischen Modellierungswerkzeugen können diese Standard-Menüleiste sowohl um einzelne Einträge als auch um komplette Menüs erweitern. Der Benutzer kann der Menüleiste zudem ohne Kenntnis von LiMo ein Menü „Extras“ hinzufügen, das den Aufruf externer Programme ermöglicht. Dieses Menü wird in einer XML-Datei definiert, die einem einfachen Schema folgt (siehe Anhang A) und die im gleichen Verzeichnis erwartet wird, in welchem LiMo seine Konfigurationsdateien ablegt (üblicherweise `.limo` im Heimatverzeichnis des Benutzers).

## 7.4 Der Navigator

Der Navigator unterstützt den Benutzer bei der Orientierung innerhalb von – speziell komplexeren – Modellen. Dazu stellt er zwei zusätzliche Sichten auf das gesamte Modell zur Verfügung:

- Eine baumartige Hierarchie des Modells. Diese basiert auf der Schachtelung der Modellelemente, die sich wiederum aus den Kompositionen zwischen Elementen im Metamodell ergibt (vergleiche Abschnitt 6.1). Diese Ansicht ermöglicht eine schnelle Auswahl einzelner Elemente mit dem Effekt, dass diese Elemente auch in der Zeichenfläche zentriert angezeigt und ausgewählt werden. Gegebenenfalls wird zuvor eine neue Zeichnung geöffnet.
- Eine Übersicht des Modells in der Art einer Landkarte. Diese zeigt das gesamte zur aktuellen Zeichnung gehörige (Teil-) Modell aus der Vogelperspektive und gibt einen Überblick über die Positionierung aller Modellelemente. Die Übersicht ist immer dann hilfreich, wenn der Umfang eines Modells einen einzelnen Bildschirm sprengt. Der sichtbare Bereich der Zeichnung wird durch ein rotes Rechteck hervorgehoben, das auch mit der Maus manipuliert werden kann.

## 7.5 Der Inspektor

Der Inspektor zeigt die Eigenschaften (Attribute und Assoziationen) des aktuell vom Benutzer ausgewählten Modellelementes an und erlaubt ebenso deren Bearbeitung. Der Inspektor kommt immer dann ins Spiel, wenn Eigenschaften von Elementen keine unmittelbare graphische Repräsentation besitzen oder aus anderen Gründen nicht in der Zeichnung bearbeitet werden können oder sollen. Der Inspektor ist in der Benutzungsschnittstelle in zwei Karteireiter aufgeteilt.

Der erste Reiter stellt die Eigenschaften eines Modellelements mit den Sichtbarkeiten `protected` und `public` in einer Tabelle zusammen. Jede Zeile enthält ein Paar aus Name

und Wert einer Eigenschaft. Durch Klicken auf den Wert kann dieser bearbeitet werden, wenn das Metamodell dies vorsieht, die Sichtbarkeit also `public` ist. Wie das geschieht, hängt vom Typ der entsprechenden Eigenschaft ab:

- Attribute vom Typ `Integer`, `Float`, `String`, `Date` und `Resource` werden unmittelbar in der Tabelle eingegeben. Es findet eine vom Datentyp abhängige Syntaxprüfung statt.
- Für Attribute vom Typ `Boolean` sowie für solche, deren Typ eine Aufzählung ist, wird eine entsprechende Auswahlliste mit den möglichen Werten angezeigt.
- Bei Attributen vom Typ `Color` wird ein zusätzlicher Dialog zur Bearbeitung angezeigt.
- Assoziationen mit einer oberen Multiplizität von 1 werden innerhalb der Tabelle bearbeitet. Es wird eine Auswahlliste angezeigt, die alle im Modell vorkommenden Instanzen des entsprechenden Typs enthält.
- Assoziationen mit einer oberen Multiplizität größer als 1 werden in einem zusätzlichen Dialog bearbeitet, der das Hinzufügen und Entfernen einzelner Instanzen ermöglicht.

Der zweite Reiter des Inspektors bietet die Möglichkeit, die textuelle Beschreibung eines Modellelements einzusehen oder zu bearbeiten. Diese wird unter anderem innerhalb des Hypertext-Systems genutzt (siehe Abschnitt 8.2) und kann deshalb einfache HTML-Elemente verwenden.

# 8 Hypertext

Das LiMo-Framework sieht die Integration von Hypertext vor. Dieser bietet im weitesten Sinne die Möglichkeit, ein Werkzeug mit Lehrmaterial – etwa Vorlesungsinhalten oder Aufgaben – zu verbinden.

Der Hypertext gliedert sich in einen fixen und einen variablen Teil. Der fixe Teil ist abhängig von Metamodell und Werkzeug, bietet somit Platz für Vorlesungsinhalte (etwa eine Beschreibung der jeweiligen Modellierungssprache) und allgemeine Hilfestellungen zur Verwendung des Werkzeugs. Der variable Teil ist abhängig vom jeweils bearbeiteten Modell und bietet sich unter anderem zum Stellen und Lösen von Aufgaben an. Beide Teile können über spezielle Hyperlinks miteinander interagieren, aber auch mit jeweils aktiven Modell sowie dem gesamten Werkzeug.

Die Benutzungsschnittstelle des Hypertextes befindet sich im rechten Drittel des Anwendungsfensters. Sie gliedert sich in zwei Karteireiter, die Zugriff auf die beiden Teile bieten (siehe Abbildung 8.1). Wird der Hypertext nicht benötigt, kann der Bereich – analog zu Inspektor und Navigator – ausgeblendet werden.

Die folgenden Abschnitte beschreiben diese Möglichkeiten im Detail.

## 8.1 Fixer Teil des Hypertextes

Der fixe Teil des Hypertextes bietet die Möglichkeit, zusammen mit einem Werkzeug eine feste Menge von Hypertext-Seiten auszuliefern, die Vorlesungsinhalte sowie Hilfestellungen zur Verwendung des Werkzeugs enthalten.

Zur Beschreibung des Hypertextes wird aus pragmatischen Gründen die Hypertext Markup Language (HTML) [W3C99a] verwendet. Aufgrund der hohen Verbreitung von HTML und der Einfachheit der Sprache ist davon auszugehen, dass die potenziellen Nutzer des Frameworks – Lehrende der Informatik – entsprechende Kenntnisse besitzen. Die HTML-Dateien sind prinzipiell in Anzahl, Struktur und Inhalt nicht beschränkt, so dass auch bereits existierender Lehrstoff in ein neues Werkzeug aufgenommen werden kann. Die übliche Startseite `index.html` wird als Einstiegspunkt für die Hilfefunktion beim Aufruf aus dem Menü verwendet und gleichzeitig zur Begrüßung neuer Benutzer beim ersten Start des Werkzeugs angezeigt.

Über die Startseite hinaus wird eine einfache Namenskonvention für den Teil des Hypertextes verwendet, in welchem die jeweilige Modellierungssprache beschrieben ist. Wird


Abbildung 8.1: Integration des Hypertext-Bereichs in die Benutzungsschnittstelle

diese eingehalten, dann kann das Werkzeug dem Benutzer eine „aktive“ Form der Hilfe bieten, die sich speziell an neue Nutzer richtet (und im Menü aktiviert bzw. deaktiviert werden kann): Bei Aktionen des Benutzers, die im Zusammenhang mit Modellelementen stehen, wird jeweils eine passende Seite des Hypertextes angezeigt. Hierbei findet das jeweilige Metamodell Berücksichtigung. So wird zum Beispiel beim Bearbeiten eines Attributs $A$ eines Modellelements $M$ im Inspektor die Hilfeseite zum jeweiligen Typ $T$ von $M$ (also zur entsprechenden Klasse des Metamodells) aufgerufen, wenn eine solche Seite existiert.

Diese Seite wird zunächst in einer Datei gesucht, welche den gleichen Namen wie $T$ trägt (zuzüglich der Endung `.html`). Wird sie nicht gefunden, dann wird $T$ sukzessive durch die Superklasse von $T$ ersetzt, und es wird erneut gesucht. Wird eine passende Seite gefunden, dann wird diese angezeigt, und es wird versucht, innerhalb der Seite zu einem lokalen Anker zu springen, der den Namen von $A$ trägt (letzteres bleibt ohne Effekt, wenn der lokale Anker nicht existiert). Bleibt die Suche nach einer passenden Seite bis zur Wurzelklasse des Metamodells erfolglos, wird keine Hilfe angezeigt.

Die Zuordnung von Hilfeseiten zu Modellelementen folgt damit grob den üblichen objektorientierten Gepflogenheiten für Sichtbarkeiten von Bezeichnern. Insbesondere findet die Vererbungshierarchie des Metamodells Berücksichtigung. Aus Entwicklersicht ergibt sich daraus die Möglichkeit, den fixen Teil des Hypertextes anhand des Metamodells zu strukturieren und Redundanz zu vermeiden, indem gemeinsame Eigenschaften von Modellelementen auf der Ebene von Oberklassen beschrieben werden.

## 8.2 Variabler Teil des Hypertextes

Der variable Teil des Hypertextes entsteht dynamisch aus dem jeweils aktiven Modell. Zu jedem Modellelement existiert eine eigene Hypertext-Seite, die immer dann angezeigt wird, wenn dieses Modellelement in der Zeichnung selektiert ist.

Der Inhalt der Seite wird im wesentlichen den Notizen des Modellelements entnommen (siehe Abschnitt 6.2). Diese werden im Normalfall als einfacher Text interpretiert. Es besteht jedoch auch hier die Möglichkeit der Verwendung von HTML – dazu muss die Notiz in ein `<html>`-Element eingeschlossen sein. Zusätzlich kann auf eine externe HTML-Datei verwiesen werden, deren Inhalt in die Hypertext-Seite aufgenommen werden soll (siehe ebenfalls Abschnitt 6.2).

Die mögliche Struktur des variablen Hypertextes ergibt sich aus den Assoziationen zwischen Modellelementen (auf der Ebene des Metamodells). Die tatsächliche Struktur ergibt sich aus dem Modell: Für jede Verbindung zwischen zwei konkreten Elementen (auf der Modellebene) wird ein spezieller Hyperlink erzeugt, der das entsprechende Modellelement in der Zeichnung auswählt (mit dem Effekt, dass anschließend dessen Hypertext-Seite angezeigt wird).

Es ist offensichtlich, dass der variable Teil des Hypertextes dem Benutzer ein Traversieren des kompletten Modells ermöglichen. Er kann damit als zusätzliche Sicht auf das Modell betrachtet werden, ähnlich wie Zeichenfläche, Inspektor und Navigator, jedoch stehen hier eher die Dokumentation und die Struktur des Modells im Vordergrund. Aus der Verbindung von Modell und Hypertext ergibt sich eine Reihe von didaktisch interessanten Einsatzmöglichkeiten:

- Den Studierenden können komplexe, mit Hypertext-Erläuterungen versehene Beispielmodelle zur Verfügung gestellt werden, die im Sinne explorativen Lernens (vgl. [Ker01, Kap.5]) zu erkunden sind.
- Aufgaben können elektronisch gestellt werden, indem den Studierenden ein leeres Modell zum Download bereitgestellt wird. Dessen Dokumentation enthält die Aufgabenstellung.
- Lösungen zu Aufgaben können auf einfache Weise dokumentiert und elektronisch abgegeben werden, da sich Modell und Dokumentation nur aus einer Datei zusammensetzen.

Hinzu kommt die Tatsache, dass auch die Studierenden vom Hypertext als einer zusätzlichen Sicht auf das Modell unmittelbar profitieren und damit zur Dokumentation ihrer Modelle – sonst eher eine unbeliebte Tätigkeit – motiviert werden. Dies ist bei existierenden Modellierungswerkzeugen meist nicht der Fall, da die Dokumentation der Elemente dort vielfach „versteckt“ wird und erst beim Generieren von Quellcode oder beim Ausdrucken des Modells zum Tragen kommt (siehe Abschnitt 3.4).

## 8.3 Interpretation von Hyperlinks

Auch wenn die beiden Teile des Hypertextes in getrennten Bereichen des Anwendungsfensters dargestellt werden, bilden sie dennoch einen gemeinsamen Hypertext. Ein Indiz dafür ist der im vorangehenden Abschnitt erwähnte Hyperlink, der die Beschreibung eines Modellelements mit der Dokumentation der entsprechenden Klasse des Metamodells verbindet. Ähnliche Hyperlinks vom variablen Teil in den fixen Teil des Hypertextes und umgekehrt können auch vom Benutzer bzw. Entwickler eines LiMo-Werkzeugs verwendet werden. Darüber hinaus kann der Hypertext auf Dokumente aus externen Quellen verweisen.

In allen Fällen werden Hyperlinks mithilfe des üblichen HTML-Elements `<a href=...>` gebildet. Wie das Framework mit dem Ziel eines Hyperlinks umgeht, hängt von dem URI ab, der das Ziel des Hyperlinks festlegt. Entscheidend ist der Protokoll-Präfix; in einigen Fällen spielt zudem die Endung des Dateinamens eine Rolle. Details hierzu finden sich in Anhang A. Es sei angemerkt, dass Hyperlinks ohne Angabe eines Präfixes auf ein Ziel im gleichen Teil des Hypertextes (und damit im gleichen Fenster) verweisen, in dem sich ihr Quellanker befindet. Hyperlinks mit leerem Ziel werden nicht verfolgt.

## 8.4 Unterstützung von Skripten

Dem Medium Hypertext wohnt bereits ein gewisses Maß an Interaktivität inne. Diese beschränkt sich jedoch mit dem gezielten Verfolgen von Hyperlinks und der Möglichkeit des nichtlinearen Lesens zunächst auf das eigentliche Hyperdokument. Das Framework erweitert diese Interaktivität bereits etwas, indem es Metamodell, Modell und Hypertext über Hyperlinks verbindet. Für einige Lernszenarien sind jedoch möglicherweise weitergehende Interaktionsmöglichkeiten wünschenswert.

Man stelle sich hierzu hypertextuelles Lehrmaterial im Sinne einer „Guided Tour“ vor, das eine Modellierungssprache anhand eines schrittweise wachsenden Beispiels erläutert oder verschiedene Modellierungsvarianten des gleichen Sachverhalts diskutiert. Jeder Schritt der Tour – also näherungsweise jede Seite des Hypertextes – soll durch ein entsprechendes Modell begleitet werden. Mit den bisher vorgestellten Möglichkeiten ist ein solches Lernszenario realisierbar, aber nur auf sehr umständliche Weise:

- Es wird für jeden Schritt ein einzelnes Modell benötigt, dessen Dokumentation den entsprechenden Abschnitt des Hypertextes enthält.
- Hyperlinks sorgen für das Laden des jeweils „nächsten“ Modells oder eine Rückkehr zum vorangehenden Schritt.

Dieses Vorgehen ist in der Erstellung jedoch sehr aufwendig – ähnelt es doch stark der Erstellung eines Daumenkinos – und zudem in seinen Möglichkeiten begrenzt, da die Modelle eher als Abbildungen fungieren, die komplett ausgetauscht werden. Wünschenswert wäre hingegen eine Möglichkeit, das Modell aus dem Hypertext heraus zu modifizieren, während der Benutzer den Hypertext traversiert. Eine minimale Menge hierfür notwendiger Operationen umfasst das Hinzufügen und Löschen von Elementen sowie das Ändern einzelner Attribute (insbesondere Name, Position und Größe) und Assoziationen. Diese ließen sich mit geringem Aufwand durch zusätzliche spezialisierte Hyperlinks realisieren. Da man sich jedoch leicht weitergehende Operationen vorstellen kann, die diese Menge sprengen (zum Beispiel die Hervorhebung und Animation eines oder mehrerer Elemente) ist eine flexiblere und konsequentere Lösung sinnvoller: die Integration einer Skriptsprache in den Hypertext-Bereich des Frameworks und damit die Ausführung quasi beliebigen Codes innerhalb der bzw. durch die HTML-Seiten.

Die Problematik ähnelt dem dynamischen Verändern von HTML-Seiten, wie es im World Wide Web (WWW) praktiziert wird. Dort wird üblicherweise der Code einer Skriptsprache wie JavaScript geeignet in eine HTML-Seite integriert und durch den Web-Browser interpretiert, bevor und während die Seite angezeigt wird. Der abstrakte Syntaxbaum der HTML-Seite ist dem Skript über ein Document Object Model (DOM) [W3C04] zugänglich. Auch wenn sich die Dynamik in diesem Fall meist auf das Verändern des eigentlichen Hyperperdokuments beschränkt – ein Pendant zu einem zusätzlichen LiMo-Modell existiert nicht – kann sich die innerhalb von LiMo verwendete Realisierung an dieser Architektur orientieren:

- Als Skriptsprache wird der Einfachheit halber Java verwendet. Dies erspart die aufwendige Bereitstellung von Schnittstellen zwischen dem in Java implementierten Framework bzw. seinen Anwendungen und der Skriptsprache. Zur Ausführung der Skripte kommt der Java-Interpreter BeanShell [Bea06] zum Einsatz.
- Vor der Ausführung eines Skripts werden Variablen definiert, die unter anderem den Zugriff auf das Modell, aber auch auf die gesamte Anwendung – und damit auf jeden Teil von LiMo – ermöglichen (siehe Anhang A). Eine Variable `root` repräsentiert zum Beispiel die Wurzel des Modells.
- Zur Einbettung der Skripte in Hypertext-Seiten wird eine Untermenge der Attribute verwendet, die der HTML-Standard zu diesem Zweck vorsieht (siehe Anhang A). Damit ist es möglich, Skripte an ganze HTML-Seiten und an einzelne Hyperlinks zu heften.

Es ist durchaus denkbar, dass mehrere Skripte gemeinsamen Code oder gemeinsame Variablen benötigen oder dass ein Skript so komplex wird, dass dessen Definition innerhalb

eines HTML-Attributs aufgrund mangelnder Lesbarkeit nicht mehr praktikabel ist. Deshalb unterstützt das Framework zusätzlich eine Möglichkeit, umfangreicheren Code an eine beliebige Stelle der Seite auszulagern, etwa in Form einer Funktion. HTML sieht zu diesem Zweck das Element `<script>` vor, das folglich auch innerhalb von LiMo Verwendung findet[1]. Der innerhalb eines oder mehrerer `<script>`-Elemente definierte Code wird bereits beim Parsen der Seite (und damit noch vor deren Anzeige) ausgeführt. Eine dort definierte Funktion kann also innerhalb der HTML-Attribute aus Tabelle A.3 aufgerufen werden.

Abbildung 8.2 demonstriert anhand eines einfachen Beispiels die Skriptfähigkeiten von LiMo. Das Skript baut schrittweise ein Anwendungsfalldiagramm bestehend aus einem System, einem Akteur, einem Anwendungsfall und einer Assoziation auf. Der erste Teil des Skripts besteht im wesentlichen aus ausgelagertem Java-Code. Er deklariert Variablen zur Aufnahme der Modellelemente und Funktionen zu deren Erzeugung. Der zweite Teil des Skripts enthält darstellbaren HTML-Code. Die darin vorkommenden Hyperlinks haben kein Ziel (verzweigen also nicht zu einer anderen Seite), führen jedoch beim Anklicken jeweils eine der zuvor deklarierten Java-Funktionen aus. Ein derartiges Beispiel könnte in der Hilfe eines Modellierungswerkzeugs dazu dienen, die Studierenden schrittweise mit den Modellelementen vertraut zu machen.

## 8.5 Erzeugung des Hypertextes

Auch wenn es möglich ist, die HTML-Dateien für den fixen Teil des Hypertextes komplett händisch zu erstellen und zu verwalten, ist dies – speziell mit Hinblick auf spätere Umstrukturierungen – ab einer gewissen Größe nicht mehr praktikabel. Es entstehen die ähnliche Wartungsprobleme wie bei anderen Arten von Hypertext – etwa im WWW.

Zur Unterstützung der Erstellung und Wartung des Hypertextes wurde daher ein einfacher Hypertext-Compiler entwickelt, der eine Spezifikation des gesamten Hypertextes einliest, diese auf Konsistenz (etwa der Hyperlinks) überprüft und sie anschließend in eine Menge von HTML-Dateien übersetzt, die den in Abschnitt 8.1 beschriebenen Anforderungen genügen. Der Compiler sorgt dabei zudem für ein einheitliches Layout aller Seiten und fügt in jede Seite generische Navigationsmöglichkeiten (Seite vor- bzw. zurück, Liste untergeordneter Themen etc.) ein. Die Eingabe des Compilers besteht aus einer XML-Datei, die neben den üblichen HTML-Elementen einige zusätzliche Elemente zur Strukturierung des Hypertextes in einzelne Seiten enthält.

Für den Übersetzungsprozess werden XSL-Transformationen [W3C99b] verwendet, die auf der Basis einer Menge von deklarativ spezifizierten Regeln ein XML-Dokument in

[1] Aus technischen Gründen muss das `<script>`-Element in einen HTML-Kommentar eingeschlossen sein, ähnlich wie dies die HTML-Spezifikation zur Kompatibilität mit älteren Browsern empfiehlt. Verantwortlich dafür ist das HTML-Anzeigeelement von Java/Swing, das in seinen technischen Möglichkeiten leider einem solchen älteren Browser entspricht.

```
<!--
<script>
    import java.awt.Rectangle;
    import org.musoft.sue.model.*;

    System system;
    UseCase usecase;
    Actor actor;
    Association link;

    void addSystem() {
        if (system == null) {
            system = new System("System1", new Rectangle(280, 20, 220, 160), model);
        }
    }

    void addActor() {
        if (actor == null) {
            actor = new Actor("Actor1", new Rectangle(50, 100, 80, 100), model);
        }
    }

    void addCase() {
        if (system != null && usecase == null) {
            usecase = new UseCase("UseCase1", new Rectangle(300, 60, 180, 80), system);
        }
    }

    void addLink() {
        if (actor != null && usecase != null && link == null) {
            link = new Association("Link1", model);
            link.setStart(actor);
            link.setEnd(usecase);
        }
    }
</script>
-->

<html>
    Das Anwendungsfalldiagramm eines Systems entsteht in den
    folgenden Schritten:

    <ol>
        <li>Das <a href="" onclick="addSystem();">System</a> selbst wird
            durch ein Rechteck repräsentiert.
        <li>Die <a href="" onclick="addActor();">Akteure</a>, befinden
            sich außerhalb des Systems.
        <li>Jeder <a href="" onclick="addCase();">Anwendungsfall</a> des
            Systems wird durch ein Oval dargestellt.
        <li>Eine <a href="" onclick="addLink();">Verbindung</a> gibt
            wieder, dass ein Akteur einen Anwendungsfall ausführen kann.
    </ol>
</html>
```

Abbildung 8.2: Beispielhaftes Skript zur Modifikation eines Modells

eine beliebiges Zielformat überführen. Die Regeln werden in einem Stylesheet zusammengefasst, das leicht an neue Anforderungen oder Zielformate angepasst werden kann. So war es im Fall von LiMo zum Beispiel mit geringem Aufwand möglich, ein weiteres Stylesheet zu definieren, das anstelle der HTML-Seiten ein PDF-Dokument erzeugt, welches den kompletten fixen Teil des Hypertextes in einer druckbaren Form enthält.

# 9 Vergleich mit anderen Arbeiten

Die Entwicklung von CASE-Werkzeugen auf der Basis des Meta-CASE-Ansatzes ist bereits seit den 90er Jahren Gegenstand der Forschung. Die Ergebnisse haben auch Einfluss auf die industrielle Entwicklung solcher Werkzeuge genommen. Im Folgenden werden einige der bekannteren Ansätze vorgestellt und mit dem LiMo-Framework verglichen.

**Kogge.** Der Koblenz Generator for Graphical Design Environments (KOGGE) [ESU97] verwendet Metamodellierung auf der Basis von EER-Modellen zur Beschreibung der abstrakten Syntax. Zusätzliche syntaktische Randbedingungen, die ein EER-Modell nicht auszudrücken vermag, werden mithilfe der Z-basierten Sprache GRAL [Fra97] formuliert. Die Beschreibung der Sprache wird kombiniert mit einer graphbasierten Spezifikation der Menüs des gewünschten Werkzeugs und einer Spezifikation von dessen reaktivem Verhalten durch ein Zustandsdiagramm. Die gesamte Spezifikation wird von einem festen Basissystem interpretiert. Zur Erstellung einer Werkzeugspezifikation wird die Ur-KOGGE verwendet, die ebenfalls auf KOGGE basiert („bootstrapping“).

**GenGEd.** Der Generische Graphische Editor (GenGEd) [Bar98] erzeugt syntaxgesteuerte Editoren für visuelle Sprachen. Als formale Basis des Ansatzes kommen algebraische Graphgrammatiken zum Einsatz [CEH+97]. GenGEd enthält Editoren für die verschiedenen Aspekte einer visuellen Sprache: Das Alphabet einer Sprache wird beschrieben, indem vordefinierte graphische Primitive (Rechtecke, Ellipsen etc.) zu neuen Symbolen kombiniert und mit Randbedingungen (etwa bezüglich des Layouts) versehen werden. Die Grammatik einer Sprache wird mithilfe von Graphtransformationen spezifiziert. Das generierte Werkzeug erlaubt dem Benutzer dann die Anwendung der einzelnen Regeln der Grammatik. Eine direkte Manipulation des Modells ist nicht vorgesehen, so dass GenGEd aus Gesichtspunkten der Bedienbarkeit nicht zu empfehlen ist. Sowohl GenGEd als auch die mit GenGEd erzeugten Werkzeuge sind in Java realisiert und damit prinzipiell plattformunabhängig. Die in [Bar98] beschriebene Realisierung verwendet jedoch eine nativ implementierte Server-Komponente zur Auswertung der Randbedingungen.

**DiaGen.** Ein weiterer generativer Ansatz zum Bau graphischer Editoren wird von DiaGen [MV95] verfolgt. Als formale Basis zur Repräsenation des Modells werden hier Hypergraphen [Min00] verwendet, deren Hyperkanten beliebig viele Knoten verbinden können. Die abstrakte Syntax einer visuellen Sprache wird durch kontextfreie Hypergraph-Grammatiken spezifiziert. Zur Darstellung wird der Hypergraph mit zusätzlichen Layout-Einschränkungen versehen und auf die konkrete Syntax abgebildet. Mit DiaGen erzeugte Werkzeuge erlaubten zunächst nur dann eine direkte Manipulation des Modells, wenn sich Aktionen des Benutzers auf Regeln der Hypergraph-Grammatik abbilden ließen

(ähnlich wie GenGEd). Später wurde DiaGen um einen Parser erweitert, der die syntaktische Analyse beliebiger, vom Benutzer erstellter Modelle unterstützt. DiaGen berücksichtigt ebenfalls die Animation von Modellen. Dazu wird die Sprachsemantik mit in die Hypergraph-Grammatik kodiert. Dieser Ansatz scheint jedoch nur tragfähig zu sein, wenn die Semantik (a) nicht zu komplex wird und (b) die Instanz eines Modells im wesentlichen diesem entspricht und es um wenige Annotationen erweitert (wie etwa im Fall von endlichen Automaten, bei denen nur die Auszeichnung für den aktiven Zustand hinzukommen).

**Ipsen.** Die Meta-Umgebung Ipsen [KS97] ist ein Ansatz zur Erzeugung von Software-Entwicklungsumgebungen. Ipsen berücksichtigt insbesondere die Unterstützung hybrider Sprachen, die sowohl graphische als auch textuelle Elemente besitzen. Textuelle Anteile werden mit Hilfe von kontextfreien Grammatiken spezifiziert. Zur Spezifikation der graphischen Anteile kommen programmierte Graphersetzungen zum Einsatz. Die Spezifikation von Werkzeugen wird durch ein (ebenfalls mit Ipsen erzeugtes) Werkzeug unterstützt. Der in [KS97] dargestellte Entwicklungsprozess ist jedoch vielschrittig und kompliziert. Mit der Graphersetzungs-Maschine PROGRES [SWZ95] und der Graphspeicherungs-Maschine GRAS [KSW95] werden zur Laufzeit sehr umfangreiche Komponenten benötigt, die zudem nur auf wenigen Plattformen verfügbar sind, so dass Ipsen-Werkzeuge den Anforderungen der Plattformunabhängigkeit und Leichtgewichtigtkeit nicht genügen.

**MEG und WOG.** Der Modeling Language Environment Generator MEG [CDFG06] bietet Unterstützung bei der Definition und Erzeugung eines graphischen Modellierungswerkzeugs. Die abstrakte Syntax einer Sprache wird – wie im Fall von LiMo – mit Hilfe eines UML-Klassendiagramms spezifiziert. Die Elemente der konkreten Syntax werden graphisch entworfen, und es wird eine Abbildung von der abstrakten in die konkrete Syntax formuliert. Aus dem Klassendiagramm wird automatisch eine eXtended Positional Grammar (XPG) [CP00] erzeugt, die nachträglich verfeinert werden kann, um zusätzliche Randbedingungen zu berücksichtigen. Die zentrale Datenstruktur einer mit MEG erzeugten Anwendung ist ein getypter Graph. Dieser wird mit Hilfe der Graph Exchange Language (GXL) [HWS00] persistent gemacht. Der Workbench Generator (WOG) integriert mehrere MEG-Werkzeuge zu einer CASE-Werkbank, wobei die gemeinsame Datenhaltung durch das GXL-Format erleichtert wird.

**PI-SET.** Die an der Universität Siegen entstandene Werkzeugsammlung für den Softwaretechnik-Unterricht (PI-SET) [Mon03, MK00] basiert auf einem Meta-CASE-Ansatz, der dem der vorliegenden Arbeit ähnelt. Das Metamodell einer Modellierungssprache wird dort als Datenbankschema beschrieben. Eine Menge von generischen Werkzeugkomponenten passt sich automatisch an dieses Metamodell an und kann zusätzlich über Vererbung verfeinert werden. Der Ansatz legt großen Wert auf die Mehrbenutzerfähigkeit der Werkzeuge. Zu diesem Zweck wird eine Variante des Datenbanksystems PCTE [BGMT98] als zentrales Repository für Modelle verwendet, wodurch die (prinzipiell in Java implementierten) Werkzeuge nicht mehr vollständig Plattformunabhängig sind.

**Eclipse.** Im Umfeld des Open-Source-Projektes Eclipse [Ecl06] existieren einige Subprojekte, die vergleichbare Zielsetzungen verfolgen wie LiMo:

- Das Eclipse Modeling Framework (EMF) [Gri06] ist eine Kombination aus Framework und Code-Generator zum Bau von Werkzeugen, die auf einem strukturierten Modell basieren. Der Generator akzeptiert als Eingabe wahlweise ein UML-Klassendiagramm in Form einer XMI-Datei, eine Menge von speziell annotierten Java-Klassen oder ein XML-Schema. Als Ausgabe wird eine Menge von Java-Klassen mit Zugriffsmethoden für die einzelnen Attribute und Assoziationen, einer Beobachterschnittstelle und Möglichkeiten zur (De-) Serialisierung erzeugt.
- Das Graphical Editing Framework (GEF) [Ani06] ist ein Framework zum Bau von graphischen, vektororientierten Editoren. GEF setzt keine bestimmte Repräsentation eines Modells voraus („model agnostic“), arbeitet aber zum Beispiel mit EMF-Modellen zusammen. GEF besitzt viele der Eigenschaften und Fähigkeiten von JHotDraw, was nicht verwunderlich ist, da ein Teil der Entwickler von JHotDraw – insbesondere Erich Gamma – inzwischen im Eclipse-Umfeld tätig sind. GEF kann somit als Nachfolger von JHotDraw auf Eclipse-Basis betrachtet werden.
- Das Graphical Modeling Framework (GMF) [Pla06] ist eine Kombination aus Framework und Code-Generator zum Bau von graphischen Modellierungswerkzeugen. GMF ist ein relativ junges Projekt, das die Lücke zwischen EMF und GEF schließt und gleichzeitig Zusatzfunktionalität bereitstellt, die von vielen Modellierungswerkzeugen benötigt wird (etwa Möglichkeiten zum Drucken). GMF definiert für die Erstellung der Werkzeuge einen Workflow, der ähnlich der in Abschnitt 5.3 beschriebenen Methodologie ist.

Die Summe dieser Projekte weist starke Gemeinsamkeiten mit Teilen von LiMo auf. Speziell GMF ist jedoch zu spät entstanden, um Einfluss auf LiMo zu nehmen. EMF und GEF hingegen sind etwa parallel zu LiMo entstanden und wären eine geeignete Basis für das Framework gewesen. Dagegen sprach aber die Tatsache, dass auf EMF/GEF basierende Werkzeuge zu diesem Zeitpunkt immer die gesamte Eclipse-Umgebung vorausgesetzt hätten, was dem Ziel der Leichtgewichtigkeit widersprochen hätte. Zudem war das Standard Widget Toolkit (SWT) [NW04], das die Basis der Benutzungsschnittstelle von Eclipse bildet, lange nur auf wenigen Plattformen verfügbar, so dass die Werkzeuge trotz der Verwendung von Java nicht plattformunabhängig gewesen wären. Diese Situation hat sich etwas entschärft, seit mehr Portierungen von SWT existieren und die Rich Client Platform (RCP) [ML05] den Bau schlankerer Werkzeuge auf Eclipse-Basis ermöglicht.

Keiner der vorgestellten Ansätze hat den Bau von Werkzeugen für die Lehre zum Ziel oder berücksichtigt diesen explizit. Dementsprechend spielen viele der in Teil I der Arbeit formulierten Anforderungen darin keine Rolle. Insbesondere bietet keiner der Ansätze eine Möglichkeit zur Integration von (Meta-) Modell und Hypertext, die der von LiMo ähnelt.

# 10 Zwischenresümee

Im zweiten Teil der Arbeit wurde das LiMo-Framework beschrieben. LiMo stellt eine Infrastruktur bereit, aus der sich mit geringem Aufwand konkrete Modellierungswerkzeuge für die softwaretechnische Lehre entwickeln lassen. Im Unterschied zu anderen Ansätzen zum Bau von CASE-Werkzeugen berücksichtigt LiMo explizit die Belange der Lehre und damit die Anforderungen, die in Teil I der Arbeit aufgestellt wurden:

- LiMo stellt eine geeignete Basis für leichtgewichtige Werkzeuge dar. Es berücksichtigt alle Facetten des Begriffs, die in Teil I diskutiert wurden (essenzielle Funktionalität, gute Bedienbarkeit, geringe Systemanforderungen, leichte Entwicklung und Wartung).
- LiMo beinhaltet ein Hypertext-System, das neben der reinen Integration von Lehrmaterial auch dessen Verknüpfung und Interaktion mit einem Modell erlaubt. Diese didaktisch motivierte Funktionalität geht über den Funktionskanon existierender CASE-Werkzeuge hinaus.

Als Beispiel wurde an verschiedenen Stellen ein Modellierungswerkzeug für Anwendungsfalldiagramme verwendet. Dieses Werkzeug ist zwar funktionstüchtig, diente aber – insbesondere aufgrund seiner geringen Komplexität – primär zur Illustration der Verwendung von LiMo. Der nun folgende dritte Teil der Arbeit beschäftigt sich mit einer Reihe von Werkzeugen, die auf der Basis von LiMo entstanden sind und innerhalb der softwaretechnischen Lehre eingesetzt werden. In diesem Teil wird auch deutlich, wie Modellierungswerkzeuge um didaktisch motivierte Funktionalität erweitert werden können, die das Lehren und Lernen einer spezifischen Modellierungssprache unterstützt.

# Teil III

# Die LiMo-Werkzeugfamilie

# 11 Überblick

In diesem Teil der Arbeit werden einige konkrete Werkzeuge beschrieben, die auf Basis des in Teil II entwickelten Frameworks entstanden sind. Die Werkzeuge decken sowohl verschiedene Modellierungssprachen als auch unterschiedliche Anwendungsgebiete ab. Neben strukturellen und dynamischen Aspekten von Softwaresystemen existieren zum Beispiel Werkzeuge zur Prozessmodellierung oder zum Data Mining.

Die Beschreibung der Werkzeuge folgt einem einheitlichen Schema. Neben einem groben Überblick über den Zweck des Werkzeugs wird jeweils diskutiert, wie das jeweilige Werkzeug die beiden zentralen Erweiterungspunkte des Frameworks ausfüllt – die abstrakte und die konkrete Syntax der Sprache. Durch das Füllen dieser beiden Erweiterungpunkte wird aus dem generischen Anwendungsgerüst von LiMo ein vollwertiges Modellierungswerkzeug.

Zudem besitzt jedes Werkzeug didaktisch motivierte Funktionalität, die über die Grundfunktionalität von LiMo hinausgeht und das Lehren und Lernen einer graphischen Sprache unterstützt. Diese Funktionalität ist abhängig von der jeweiligen Sprache, also spezifisch für jedes Werkzeug. Sie orientiert sich an den in Abschnitt 2.7 diskutierten Problemen bei der Lehre graphischer Modellierungssprachen und trägt insbesondere dazu bei, das Verständnis von Semantik, Pragmatik und Abstraktion einer Modellierungssprache bzw. eines konkreten Modells zu erleichtern.

# 12 Das Werkzeug DAVE

Als erstes Beispiel für eine graphische Modellierungssprache, deren Lehre von geeigneten Werkzeugen profitieren kann, sollen Zustandsdiagramme [Har87] dienen, eine Teilsprache der UML, die das diskrete Verhalten eines Systems in Form von Zuständen und Übergängen zwischen diesen beschreibt. Zustandsdiagramme können zum Beispiel das Verhalten einer Klasse auf einem hohen Abstraktionsniveau oder die erlaubten Aufrufreihenfolgen der Methoden einer Schnittstelle spezifizieren. Im Bereich eingebetteter Systeme, wo Aktualisierungen oder Korrekturen der Software nach dem Ausliefern des Systems nur schwer möglich (weil sich die Software in einem nur lesbaren Speicher befindet) oder sehr teuer (weil damit eine Rückrufaktion verbunden ist) sind, dienen Zustandsdiagramme zur rigorosen Spezifikation und Verifikation des Systemverhaltens vor dessen Implementierung.

Durch ihre vielfältigen Einsatzmöglichkeiten sind Zustandsdiagramme Teil praktisch aller Vorlesungen, die sich mit Softwaretechnik im Allgemeinen oder mit UML im Besonderen beschäftigen. Ihre Vermittlung in der Lehre ist jedoch mit einer Reihe von Problemen behaftet, die unter anderem in der komplexen Laufzeitsemantik und dem hohen Abstraktionsgrad begründet sind, der Zustandsdiagrammen innewohnt. Das Modellieren mit Zustandsdiagrammen wird zwar von gängigen UML-Werkzeugen wie Together [Tog06], Magic Draw [Mag06] oder ArgoUML [Arg06] unterstützt. Jedoch richten sich diese Werkzeuge primär an professionelle Software-Entwickler und bieten dementsprechend keinerlei Funktionalität zum Lehren oder Lernen der einzelnen Diagrammarten.

Um Unterstützung beim Lehren von Zustandsdiagrammen anzubieten, wurde der Dortmunder Automatenvisualisierer und -editor, kurz DAVE[1], entwickelt. DAVE ist ein Modellierungswerkzeug für Zustandsdiagramme, das primär für den Einsatz innerhalb des Übungsbetriebs gedacht ist, aber auch in Vorlesungen oder zum Selbststudium Verwendung finden kann. Die folgenden Abschnitte beschreiben das Werkzeug näher.

## 12.1 Abstrakte und konkrete Syntax

Abbildung 12.1 zeigt das Metamodell für Zustandsdiagramme, wie sie von DAVE unterstützt werden. Es bedient sich dabei der Notation, die in Teil II der Arbeit vereinbart wurde.

[1] Die namentliche Verwandtschaft von DAVE mit David Harel, dem Erfinder der „klassischen“ Zustandsdiagramme, ist dabei durchaus beabsichtigt.

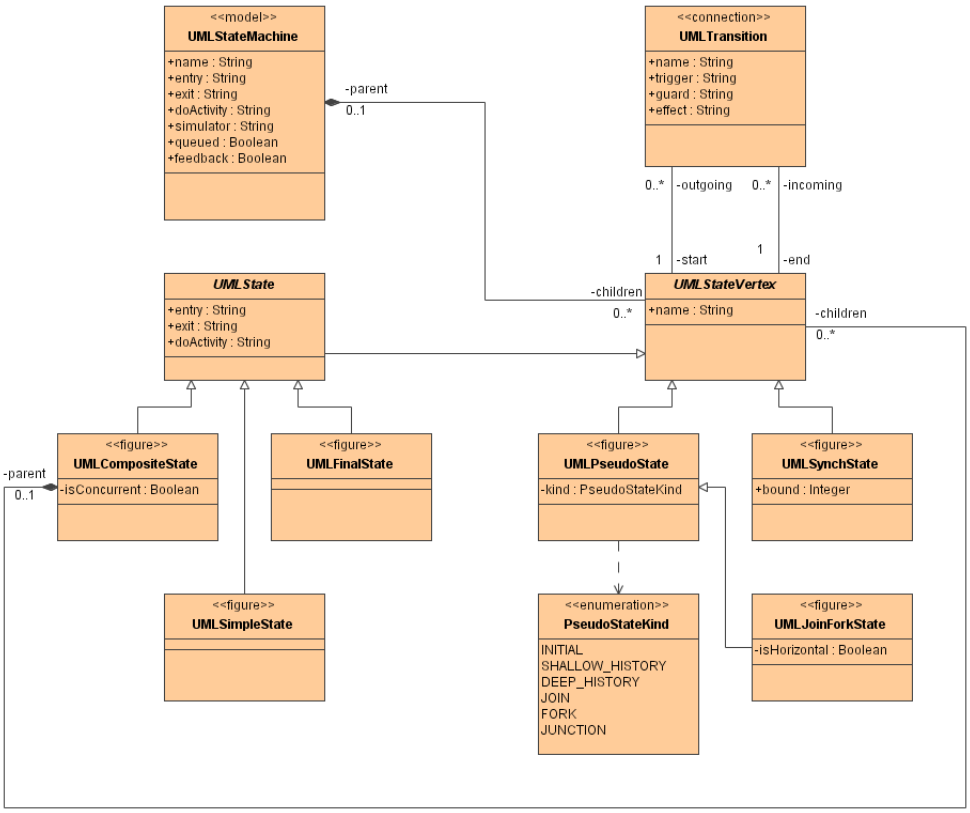



Abbildung 12.1: Metamodell für Zustandsdiagramme in DAVE

Die Klasse `UMLStateMachine` repräsentiert die Wurzel des Modells und damit das Zustandsdiagramm selbst. Die beiden abstrakten Klassen `UMLState` und `UMLStateVertex` sind Hilfskonstrukte, die gemeinsame Eigenschaften verschiedener konkreter Klassen des Metamodells definieren.

`UMLStateVertex` stellt die Abstraktion eines beliebigen Knotens in einem Zustandsdiagramm dar. Neben einem Namen verfügt jeder dieser Knoten über eine Menge von eingehenden und eine Menge von ausgehenden Transitionen. Die Klasse `UMLPseudoState` dient zur Repräsentation der verschiedenen sogenannten „Pseudozustände“, also solcher Zustände, die während des Schaltens einer Transition besucht werden, aber nicht längerfristig aktiv sein können. Dazu zählen der Initialzustand, die beiden Arten der Historie, die Vereinigung, die Gabelung und die Kreuzung. Die Art des Pseudozustands wird über das Attribut `kind` festgelegt. Vereinigung und Gabelung werden zudem über eine eigene Subklasse `UMLJoinForkState` repräsentiert, da sie ein zusätzliches Attribut `isHorizontal` zur Festlegung der Ausrichtung auf dem Bildschirm besitzen. Weiterhin existiert die Klasse `UMLSynchState` zur Repräsentation von Synchronisationszuständen in nebenläufigen Systemen.

Die Klasse `UMLState` repräsentiert „eigentliche“ Zustände innerhalb des Zustandsdiagramms, also solche, in denen das modellierte System während seiner Ausführung über die Grenze eines Verarbeitungsschritts hinaus verweilen kann. Diese Zustände besitzen Aktionen, die bei ihrem Betreten und Verlassen (Attribute `entry` und `exit`) ausgeführt werden sowie eine Aktivität, die mit dem Zustand verknüpft ist (Attribut `doActivity`). `UMLSimpleState` und `UMLFinalState` sind konkrete Ausprägungen von `UMLState`, die einfache Zustände und Endzustände repräsentieren. Die Klasse `UMLCompositeState` realisiert Dekompositionen – ob es sich um einen nebenläufigen Zustand handelt oder nicht, wird über das Attribut `isConcurrent` festgelegt. Eine Dekomposition enthält zudem wieder eine Menge von Instanzen von `UMLStateVertex`, so dass beliebige Schachtelungen von Zuständen möglich sind.

Die Klasse `UMLTransition` repräsentiert Transitionen. Jede Transition besitzt einen Namen. Sie kann zudem mit einem Auslöser (Attribut `trigger`), einer Bedingung (Attribut `guard`) und einem Effekt (Attribut `effect`) annotiert werden.

Das Metamodell orientiert sich in Struktur und Namensgebung am entsprechenden Ausschnitt des Metamodells von UML 1.5 [OMG03b], ist aber etwas einfacher gehalten. Die Unterschiede sind meist pragmatisch begründet:

- Auslöser, Bedingungen und Effekte werden nicht über eigene Klassen (in UML `Event`, `Guard` und `Action`) repräsentiert, sondern jeweils über einen String. Grund dafür ist zum einen die Tatsache, dass der Nachbau der entsprechenden Teile der UML-Infrastruktur zu aufwendig gewesen wäre. Zum anderen soll der Benutzer des Werkzeugs diese Attribute ohnehin in textueller Form eingeben.
- Einige fortgeschrittenere Konzepte von Zustandsdiagrammen, die in der Lehre üblicherweise nicht vorkommen, werden nicht unterstützt. Dazu zählen das Referenzieren anderer Zustandsdiagramme (in UML durch `SubmachineState` und `StubState` realisiert), „aufgeschobene“ Ereignisse und interne Transitionen.
- Aus dem gleichen Grund wird der Pseudozustandstyp `DYNAMIC_CHOICE` (der syntaktisch weitgehend der Kreuzung entspricht, jedoch eine minimal andere Semantik besitzt) nicht unterstützt.
- Das Zustandsdiagramm selbst referenziert nicht einen ausgezeichneten Wurzelzustand (der in UML immer vom Typ `UMLCompositeState` sein muss), sondern enthält eine Menge von `UMLStateVertex`-Instanzen, die auf oberster Ebene liegen. Die Eigenschaften `entry`, `exit` und `doActivity` der Klasse `UMLState` wurden aber hier dupliziert, so dass die Modellierungsmöglichkeiten denen des UML-Metamodells entsprechen.
- Neu sind außerdem die Attribute `feedback`, `queued` und `simulator`, die Eigenschaften der Simulation festlegen (siehe Abschnitt 12.2 bzw. Abschnitt 12.3).

Die zahlreichen zusätzlichen Randbedingungen für die Wohlgeformtheit eines Zustandsdiagramms, die in der UML-Spezifikation durch OCL-Ausdrücke formuliert sind, werden durch Nutzung der Infrastruktur zur Syntaxprüfung umgesetzt (siehe Abschnitt 6.3).

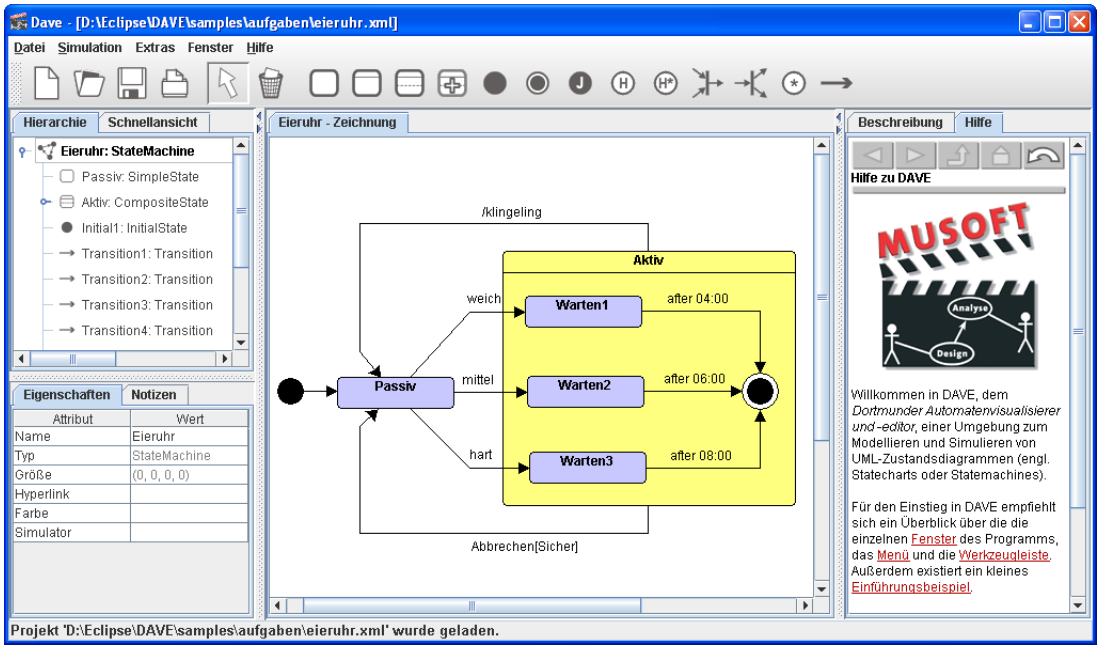



Abbildung 12.2: Bearbeiten eines Zustandsdiagramms mit DAVE

Die konkrete Syntax von DAVE orientiert sich an den Vorgaben der UML-Spezifikation und entspricht damit den üblichen Gepflogenheiten von UML-Werkzeugen. Einzig einige der Pseudozustände (Vereinigung, Gabelung und Kreuzung) wurden zusätzlich mit ihrem jeweiligen Anfangsbuchstaben annotiert, um leichter (voneinander sowie vom Startzustand) unterscheidbar zu sein.

Abbildung 12.2 zeigt ein Bildschirmfoto der Bearbeitung eines Zustandsdiagramms in DAVE. Man erkennt die bereits aus Teil II bekannte Benutzungsschnittstelle. Werkzeugleiste und Zeichnung spiegeln die Notation der Zustandsdiagramme wider. Der Rest bleibt unverändert.

## 12.2 Simulation des Modells

Zustandsdiagramme besitzen eine komplexe Laufzeitsemantik, die nicht ohne Grund lange Gegenstand der wissenschaftlichen Diskussion war [HN96, vdB94]. Die Feinheiten dieser Semantik sind in Vorlesungen schwer zu vermitteln und werden von den Studierenden oft entsprechend schlecht verstanden. Im Rahmen von Übungsaufgaben konstruierte Modelle sollen helfen, den Formalismus einzuüben. Die semantische Korrektheit eines komplexeren Zustandsdiagramms in Bezug auf gegebene Anforderungen ist aber sowohl für die Studierenden als auch für die beteiligten Tutoren durch bloßes Hinschauen oder gedankliches Durchspielen einzelner Eingaben schwer zu entscheiden. Durch große

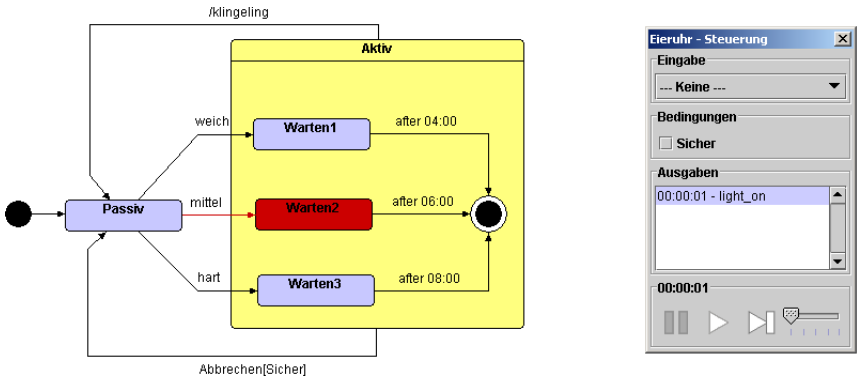



Abbildung 12.3: Simulieren eines Zustandsdiagramms mit DAVE

Studierendenzahlen wird das Problem der Korrektur solcher Aufgaben noch verschärft. Als Ergebnis prüfen Übungsaufgaben für Zustandsdiagramme vielfach schwerpunktmäßig deren Syntax ab und betrachten die – ungleich wichtigere – Semantik nur oberflächlich.

Um eine Brücke zwischen Syntax und Semantik von Zustandsdiagrammen zu schlagen und den Studierenden einen besseren Einblick in deren Laufzeitverhalten zu ermöglichen, unterstützt DAVE die interaktive Simulation des erstellten Modells. Basis für diese Simulation ist die in der UML-Spezifikation enthaltene Semantikbeschreibung für Zustandsdiagramme.

Zur Kontrolle der Simulation erscheint ein zweites, kleineres Fenster, dessen unterer Teil der Steuerung einer Stereoanlage ähnelt (siehe Abbildung 12.3). Die einzelnen Schalter dienen zum Pausieren der Simulation, zum erneuten Starten und zum Ausführen eines einzelnen Schrittes. Zudem kann die Geschwindigkeit der Simulation festgelegt werden. Für jeden Schritt der Simulation kann im oberen Teil des Fensters ein Ereignis ausgewählt werden, das vom Zustandsdiagramm zu verarbeiten ist. Die beiden Abschnitte in der Mitte des Fensters dienen zur Kontrolle der Überwachungsbedingungen und zur Anzeige von Ausgabeereignissen, die während der Simulation erzeugt werden. Aktive Zustände und schaltende Transitionen werden während der Simulation im Zustandsdiagramm durch farbliche Hervorhebungen animiert. Das Diagramm selbst wird während der Simulation in einen nur lesbaren Modus versetzt, damit keine Inkonsistenzen entstehen können. Während der Simulation wird deren gesamter Verlauf – inklusive aller aktiven Zustände sowie Ein- und Ausgabeereignisse – in einer Tabelle aufgezeichnet. Dies ermöglicht die Analyse und den Vergleich von kompletten Simulationsläufen *post mortem*.

## 12.3 Visualisierung der Simulation

Zustandsdiagramme beschreiben Verhalten auf eine sehr abstrakte Weise. Anwendungen von Zustandsdiagrammen in konkreten Systemen können – gerade bei der klassischen

Durchführung des Übungsbetriebs mit Stift und Papier, aber auch bei der Verwendung von CASE-Werkzeugen – bestenfalls als Ausgangspunkt von Übungsaufgaben dienen. Anschließend bewegen sich die Studierenden ausschließlich auf der Ebene des abstrakten Modells. Ein Rückschritt in die Realität und damit eine Verbindung von abstraktem Modell und konkreter Realität ist nicht möglich. Hinzu kommt speziell bei Zustandsdiagrammen die Tatsache, dass diese sich nicht – wie im Fall von Klassendiagrammen – direkt auf Quellcode in einer Programmiersprache abbilden lassen. Damit besitzen sie für die Studierenden, abgesehen von dem Erkenntnisgewinn, der sich (möglicherweise) bei der Modellierung einstellt, keinen unmittelbaren Nutzen. Als Ergebnis sind Zustandsdiagramme bei den Studierenden tendenziell eher unbeliebt.

Zum Überwinden der Kluft zwischen abstraktem Modell und realen Anwendungen bietet DAVE die Möglichkeit, die Simulation eines Zustandsdiagramms zu visualisieren. Unter einer Visualisierung soll im Folgenden eine die Realität nachahmende, anschauliche Darstellung eines Systems verstanden werden. Die Visualisierung soll mit der Simulation verknüpft werden können, so dass die Studierenden das Verhalten des realen Systems und des Modells parallel studieren können. Es existieren zwei verschiedene Arten von Visualisierungen, die sich sowohl in der Art der Verknüpfung mit dem Modell als auch ihren Anwendungsmöglichkeiten unterscheiden. Beide werden von DAVE unterstützt und im Folgenden anhand eines Beispiels diskutiert.

### 12.3.1 White-Box-Visualisierungen

Gegeben sei eine Modellierung eines einfachen (zugegebenermaßen etwas altmodischen) Fernsehers mithilfe eines Zustandsdiagramms (siehe Abbildung 12.4, obere Hälfte). Das Gerät kann über eine Taste ein- bzw. ausgeschaltet werden. Im eingeschalteten Zustand wird immer eines von drei Programmen angezeigt. Eine zweite Taste ermöglicht das zyklische Wechseln des Programms. Das zuletzt aktive Programm wird über das Ausschalten hinaus gespeichert.

Eine mögliche Visualisierung für dieses Modell besteht aus einem Bild des Fernsehers, das entweder einen scharzen Bildschirm oder eines der drei Programme anzeigt (siehe Abbildung 12.4, untere Hälfte). Die naheliegendste Art der Verknüpfung von Modell und Visualisierung stützt sich auf die aktuelle Zustandskonfiguration. Die Visualisierung beobachtet also die Simulation und ändert ihr Erscheinungsbild in Abhängigkeit der momentan aktiven bzw. inaktiven Zustände.

Damit dies funktioniert, muss der Visualisierung eine Teilmenge der Zustände des Modells namentlich bekannt sein – nämlich exakt jene Zustände, die sich auf das Erscheinungsbild der Visualisierung auswirken. Die Visualisierung muss also Einblick in die interne Struktur des Modells haben, so dass sich dieser Typ von Visualisierung als *White-Box*-Visualisierung klassifizieren lässt.

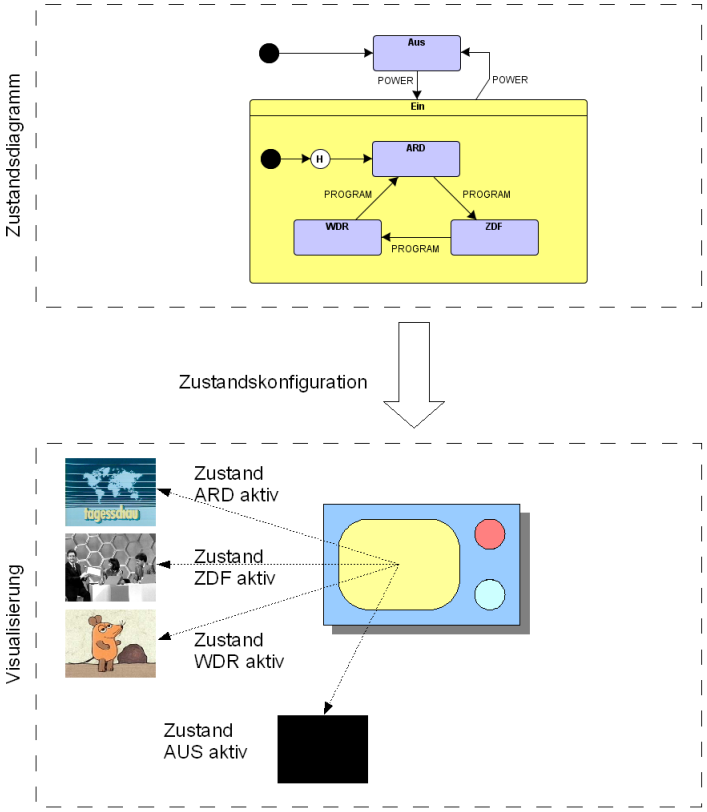



Abbildung 12.4: White-Box-Visualisierung eines Zustandsdiagramms in DAVE

Visualisierungen dieses Typs sind gut zu Demonstrationszwecken geeignet, etwa um im Rahmen einer Vorlesung die Semantik eines existierenden Modells zu verdeutlichen. Modell und Visualisierung bieten verschiedene Sichten auf das gleiche System. Die Möglichkeit, das Verhalten des Systems im abstrakten Modell und der realitätsnahen Visualisierung parallel zu beobachten, erlaubt es den Studierenden, die vorab getroffenen Modellierungsentscheidungen besser nachzuvollziehen.

White-Box-Visualisierungen sind allerdings weniger gut zu Übungszwecken geeignet. Man stelle sich hierzu eine Übungsaufgabe vor, in deren Rahmen die Studierenden das zuvor natürlichsprachlich formulierte Verhalten des Fernsehers unter Zuhilfenahme der Visualisierung in einem Zustandsdiagramm umsetzen sollen. Die Übungsaufgabe müsste die Menge der für die Visualisierung relevanten Zustände vorgeben, um deren Verknüpfung mit dem Modell zu gewährleisten. Damit würde aber bereits ein signifikanter Teil der Lösung vorgegeben, denn das Ermitteln der möglichen Systemzustände ist ein zentrales Element einer solchen Aufgabe. Die frei wählbaren Transitionen und eventuell zusätzliche Zustände ließen zwar immer noch einen gewissen Spielraum bei der Modellierung, aber die Aufgabe würde dennoch trivial.

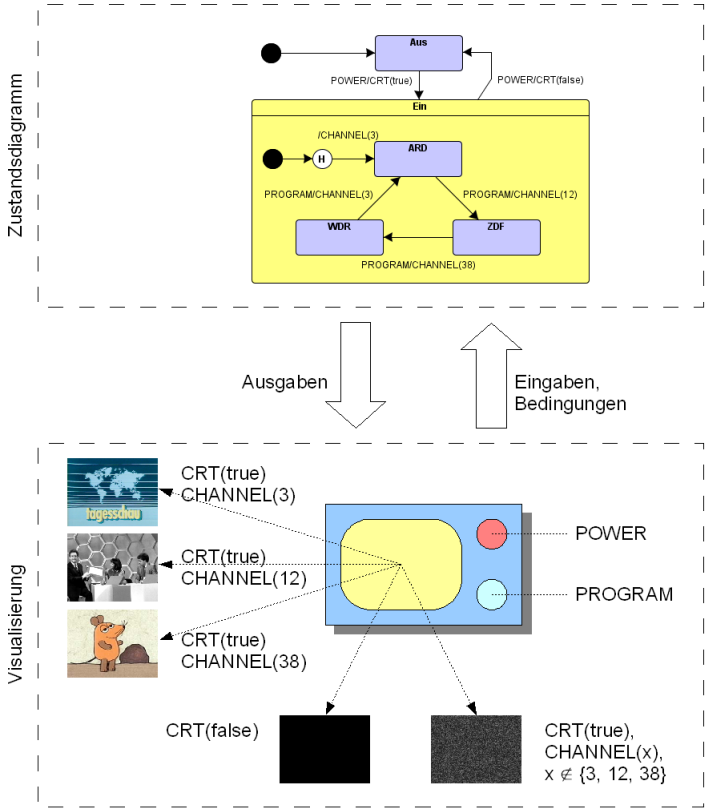



Abbildung 12.5: Black-Box-Visualisierung eines Zustandsdiagramms in DAVE

### 12.3.2 Black-Box-Visualisierungen

Eine alternative Form der Verknüpfung von Modell und Visualisierung, die sich bei Zustandsdiagrammen anbietet, verwendet die „natürliche“ Schnittstelle eines Zustandsdiagramms zu seiner Umwelt, nämlich dessen Ein- und Ausgaben sowie die Überwachungsbedingungen.

Abbildung 12.5 zeigt das entsprechend angepasste Zusammenspiel von Modell und Visualisierung. Die Visualisierung beobachtet die Ausgaben des Zustandsdiagramms und ändert gegebenenfalls ihr Erscheinungsbild. Die Ausgaben können zum Beispiel Ereignisse (im Sinne etwa einer Nachricht oder eines elektrischen Signals) oder Methodenaufrufe sein. Möglich wären auch Zuweisungen an Variablen, die für die Umgebung des Zustandsdiagramms sichtbar sind. Umgekehrt kann die Visualisierung verwendet werden, um Eingaben für das Zustandsdiagramm zu liefern (etwa über spezielle Schalter) oder dessen Überwachungsbedingungen zu kontrollieren.

Bei dieser Art der Visualisierung werden keinerlei Annahmen über die Struktur des Modells getroffen, was sie für Übungsaufgaben geeigneter macht. Modell und Visualisierung

kommunizieren über eine definierte Schnittstelle miteinander, so dass sich dieser Visualisierungstyp als *Black-Box*-Visualisierung klassifizieren lässt.

Es sei angemerkt, dass Zustandsdiagramm und Visualisierung hier nicht mehr unterschiedliche Sichten auf das gleiche System sind, sondern kommunizierende Subsysteme eines größeren Gesamtsystems. Die Visualisierung selbst kann dabei einen eigenen (möglicherweise sehr komplexen) Zustandsraum besitzen. Das korrekte Funktionieren des Gesamtsystems ergibt sich erst durch das Zusammenspiel beider Subsysteme.

Dieser Umstand lässt sich didaktisch ausnutzen, um den Studierenden mithilfe der Visualisierung Feedback zum Fortschritt oder der Qualität ihrer Lösung zu liefern. Zeigt der Fernseher beim Betätigen der entsprechenden Taste zyklisch alle drei Programme korrekt an, ist dies ein Indikator für eine korrekte Lösung. Ebenso ist es möglich, in der Visualisierung Zustände für zu erwartende Fehler in Modellen vorzusehen. Das Modell aus Abbildung 12.5 enthält einen solchen Fehler. Da die Ausgaben zur Auswahl des Kanals an die Transitionen gesetzt wurden, wird beim erneuten Einschalten des Geräts zwar die Bildröhre aktiviert, aber kein Sender gewählt[2]. Dieser Fall kann leicht durch ein „rauschendes“ Bild repräsentiert werden, um die Studierenden auf den Fehler hinzuweisen.

### 12.3.3 Beispiele für Visualisierungen

Neben dem bereits vorgestellen Beispiel des Fernsehers, das in beiden Varianten existiert, enthält DAVE verschiedene andere Visualisierungen:

- Eine Waschmaschine, bei der Eigenschaften wie Motorgeschwindigkeit sowie Zu- und Ablauf von Wasser kontrolliert werden können. Das Zustandsdiagramm der Studierenden soll einen kompletten Waschvorgang realisieren. Die Visualisierung gibt Hinweise auf die Korrektheit der Lösung: Zu Beginn der Simulation wird schmutzige Wäsche in die Maschine gelegt. Am Ende eines korrekten Waschvorgangs sollte diese Wäsche sauber sein. Fatale Fehler im Zustandsdiagramm enden zum Beispiel in einer Überschwemmung. Abbildung 12.6 zeigt einige Bilder der Waschmaschine, die verschiedenen Zuständen der Simulation entsprechen.
- Eine Kaffeemaschine, bei der das Zustandsdiagramm die Heizvorrichtung kontrolliert und korrekt auf die Einstellungen des Benutzers reagieren soll. Zudem müssen bestimmte Vorgaben wie eine automatische Abschaltung nach einer festen Zeitspanne berücksichtigt werden. Auch hier gibt die Visualisierung Hinweise auf die Korrektheit der Lösung: Erscheint Kaffee in der Kanne und schaltet sich die Maschine nach der geforderten Zeit automatisch ab, dann ist die Lösung korrekt – geht die Maschine in Flammen auf, dann ist sie verbesserungsbedürftig.

[2] Die Historie aktiviert die alte Zustandskonfiguration unmittelbar. Transitionen schalten nicht. Eine bessere Lösung wäre in diesem Fall gewesen, die *CHANNEL()*-Ausgaben als *entry*-Aktionen an die Zustände zu heften.

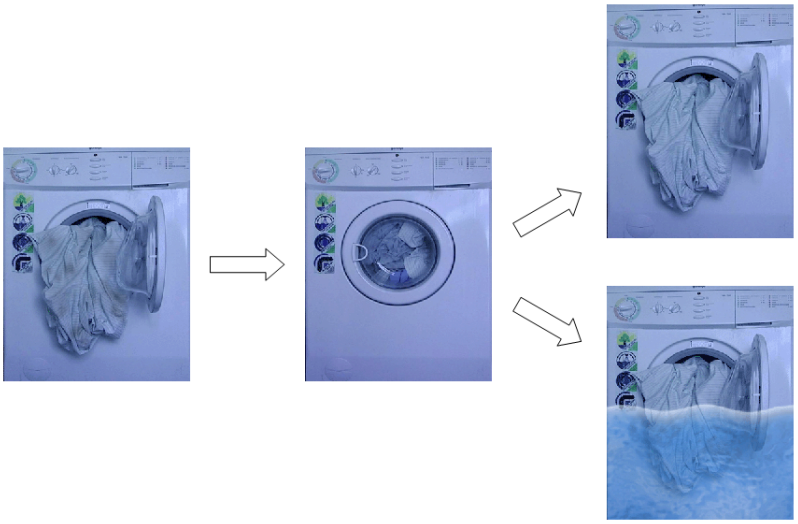


Abbildung 12.6: Visualisierung des Verhaltens einer Waschmaschine

- Einen Pfandautomaten, der eine beliebige Anzahl von Flaschen und Tassen akzeptiert und anschließend einen Bon über die Pfandsumme ausdruckt. Die Pfandgegenstände werden über Sensoren anhand ihrer Größe erkannt. Das Zustandsdiagramm muss ein Förderband ansteuern und im richtigen Moment eine Sortiervorrichtung aktivieren, damit die Gegenstände korrekt einsortiert werden. Bei falscher Modellierung kommen die eingegebenen Gegenstände unter Umständen in Form von Scherben wieder heraus[3].
- Eine digitale Armbanduhr, die zwischen verschiedenen Modi (Uhrzeit, Datum etc.) wechseln kann und für die mehrere Weckzeiten einstellbar sind. Diese Visualisierung ahmt das ursprüngliche Beispiel von Harel aus [Har87] nach, wenn auch nicht in allen Details (einige Eigenschaften von Harel Statecharts werden von UML-Zustandsdiagrammen nicht unterstützt, etwa das Löschen der Historie). Das Beispiel folgt dem White-Box-Ansatz und ist damit gut für eine Einführung von Zustandsdiagrammen im Rahmen einer Vorlesung oder zum Selbststudium geeignet.

Alle Beispiele wurden mit geringem Aufwand entweder digital fotografiert oder gezeichnet. Die Bilder wurden in „Kacheln“ unterteilt, die je nach aktuellem Zustand kombiniert werden können. Die Implementierung ist mit einem gewissen, von der Komplexität des gewünschten Verhaltens abhängigen Aufwand verbunden. Da jedoch auf der Basis jeder Visualisierung mehr als eine Übungsaufgabe gestellt werden kann, zahlt sich dieser Aufwand schnell aus. Neue Visualisierungen können dem Werkzeug über eine definierte Schnittstelle hinzugefügt werden.

[3] Dieses Beispiel ist für die Studierenden in Dortmund besonders interessant – der Automat ist in der dortigen Cafeteria zu finden und dementsprechend bestens bekannt.

## 12.4 Nutzung von Java-Code in Zustandsdiagrammen

Zum üblichen Vorwissen Studierender beim Besuch einer Softwaretechnik-Veranstaltung (in Dortmund typischerweise im dritten Semester) gehört die Programmierung in einer objektorientierten Programmiersprache wie Java. Um den Studierenden einen leichteren Bezug von Zustandsdiagrammen zu diesem Vorwissen zu ermöglichen und sie gleichzeitig Möglichkeiten der Nutzung von Zustandsdiagrammen erkennen zu lassen, erlaubt DAVE die Verwendung von Java-Code in einem Zustandsdiagramm. Dies stellt bei der Modellierung naheliegenderweise keine neuen Anforderungen an das Werkzeug, wirkt sich jedoch auf die Simulation aus. Hier sind Aktionen und Bedingungen geeignet zu interpretieren.

Während bei der Simulation der bisher diskutierten Modelle nur der Name eines Ausgabeereignisses eine Rolle spielte (es sich also im Sinne der UML um ein `SignalEvent` handelte), werden diese nun als Java-Anweisungen interpretiert. Gleiches gilt für die Überwachungsbedingungen, welche als Java-Ausdrücke angenommen werden, deren Auswertung in einem booleschen Wert resultiert. Zur Auswertung von Anweisungen und Ausdrücken kommt wieder der Java-Interpreter BeanShell [Bea06] zum Einsatz, der bereits im Hypertext-Bereich des LIMO-Frameworks Verwendung fand (siehe Abschnitt 8.4).

Aktiviert wird die erweiterte Simulationsmöglichkeit durch die Nutzung einer speziellen Visualisierung. Sie setzt also technisch auf der zuvor diskutierten Infrastruktur auf. Die Visualisierung zeigt jedoch kein Gerät, sondern eine stets aktuelle Liste von im Zustandsdiagramm verwendeten Variablen und deren Werten. Sie liefert also weniger Anschaulichkeit als vielmehr vertiefte Einblicke in das Zustandsdiagramm.

## 12.5 Code-Generierung für LEGO Mindstorms

In der Realität kommen Zustandsdiagramme oft im Bereich eingebetteter Systeme zum Einsatz. Die verschiedenen Visualisierungen von Alltagsgeräten leisten damit nicht nur einen Beitrag zur Anschaulichkeit von Aufgabenstellungen und zur Erhöhung der Motivation. Sie geben den Studierenden gleichzeitig ein Gefühl für typische Nutzungsszenarien von Zustandsdiagrammen. Die Visualisierungen sind aber letztlich nur ein Hilfsmittel, um die Realität nachzuahmen. Es stellt sich daher die Frage, ob es nicht möglich ist, die in DAVE modellierten Zustandsdiagramme zur Steuerung eines realen eingebetteten Systems zu nutzen. Der Einsatz realer Waschmaschinen innerhalb des Übungsbetriebs kommt aus naheliegenden Gründen nicht in Frage. Eine preiswerte und gefahrlose Alternative, die oft zu Lehrzwecken in Schulen und Hochschulen verwendet wird, ist das LEGO Mindstorms Robotics Invention System [LEG06].

Im Rahmen einer Diplomarbeit [Moc06] wurde erfolgreich der Versuch unternommen, DAVE mit dem Mindstorms-System zu verbinden. Die Diplomarbeit verwendet einen Ansatz zur Code-Generierung aus Zustandsdiagrammen, der sich an [vGB99] anlehnt, aber im Unterschied dazu alle relevanten Möglichkeiten von Zustandsdiagrammen (inklusive

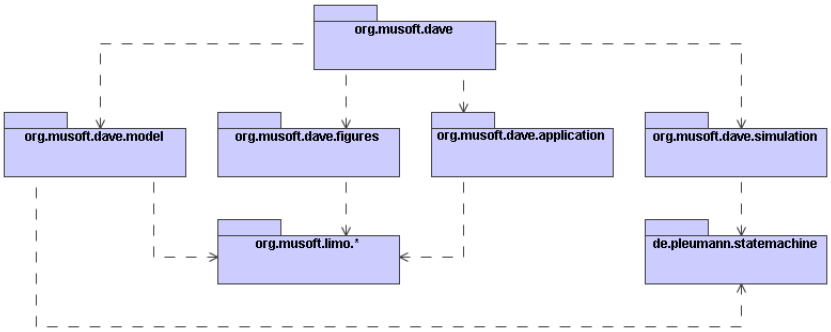



Abbildung 12.7: Grobe Architektur von DAVE

Nebenläufigkeit) unterstützt. Zur Ansteuerung der verschiedenen Arten von Sensoren und Aktoren des LEGO-Systems wird eine feste Menge von Ereignissen, Bedingungen und Aktionen vereinbart. Die Diplomarbeit enthält zudem verschiedene Beispiele für Modelle, Aufgaben und Roboter. Sie fügt der Arbeit mit DAVE eine spielerische Komponente hinzu, die das Werkzeug insbesondere für den Einsatz innerhalb des Informatik-Unterrichts an Schulen tauglich machen könnte.

## 12.6 Architektur von DAVE

Abbildung 12.7 zeigt die grobe Architektur von DAVE in Form eines Paketdiagramms mit den wichtigsten Abhängigkeiten. Der Modellierungsteil des Werkzeugs nutzt die Infrastruktur, die durch das in Teil II der Arbeit beschriebene Framework bereitgestellt wird. Die Implementierung des Metamodells (abstrakte Syntax) findet sich im Paket `model` wieder. Die Figuren zur graphischen Repräsentation der Elemente von Zustandsdiagrammen (konkrete Syntax) sind im Paket `figures` beheimatet. Das Paket `application` enthält die eigentliche Anwendung sowie an DAVE angepasste Menüs, Werkzeugleisten und eine Zeichenfläche, die als Bindeglied zwischen abstrakter und konkreter Syntax dient.

Die Funktionalität zum Simulieren von Zustandsdiagrammen findet sich im Java-Paket `simulation` wieder. Kernkomponente der Simulation ist eine Bibliothek zur Repräsentation und Ausführung von UML-Zustandsdiagrammen, die auf das Projekt „Modellieren in einer virtuellen Welt“ [WPD01] zurückgeht und vom gleichen Autor stammt. Die Bibliothek definiert eine Reihe von Schnittstellen für die verschiedenen Elemente von Zustandsdiagrammen und stellt eine Klasse `StateMachineRunner` bereit, welche die Simulation auf der Basis dieser Schnittstellen realisiert. Die Klassen des Metamodells von DAVE implementieren die vorgegebenen Schnittstellen.

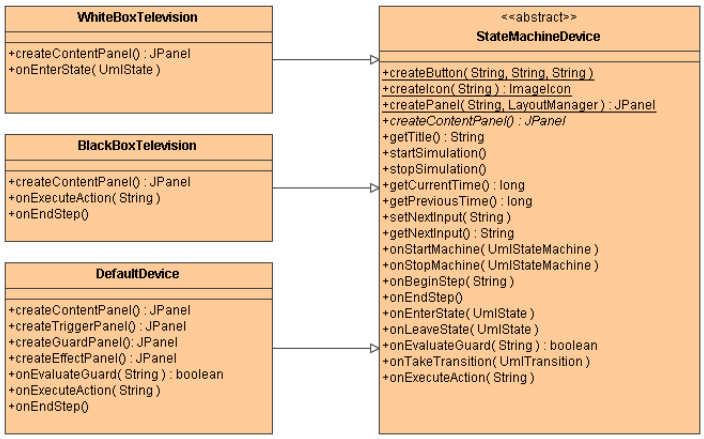



Abbildung 12.8: Hinzufügen eigener Visualisierungen zu Dave

### 12.6.1 Hinzufügen eigener Visualisierungen

Dave ist nicht auf die in Abschnitt 12.3.3 beschriebenen Visualisierungen beschränkt. Das Werkzeug besitzt einen Erweiterungspunkt, der Lehrenden die Möglichkeit bietet, eigene Visualisierungen hinzuzufügen. Die Implementierung einer solchen Visualisierung wird in Java durchgeführt. Um den hierfür benötigten Implementierungsaufwand sinnvoll zu begrenzen, existiert eine abstrakte Basisklasse `StateMachineDevice` für Visualisierungen, die sowohl eine einfach zu benutzende Fassade zur gesamten Simulation als auch verschiedene Hilfsmethoden bietet. Das Verständnis dieser einen Klasse ist im Allgemeinen ausreichend, um eine Visualisierung zu entwickeln.

Das Klassendiagramm in Abbildung 12.8 zeigt die Basisklasse sowie drei konkrete Visualisierungen, die `StateMachineDevice` erweitern (die Implementierungen befinden sich in Anhang A). Die Klasse `WhiteBoxTelevision` implementiert das zuvor beschriebene Beispiel des Fernsehers als White-Box. `BlackBoxTelevision` realisiert entsprechend die Black-Box-Variante. Beide überschreiben verschiedene vorgegebene Methoden, die auf Zustandsänderungen oder Ausgaben reagieren und die graphische Darstellung anpassen. Sämtliche in Abschnitt 12.3.3 beschriebenen Visualisierungen sind auf diese Weise implementiert. Tatsächlich nutzt sogar die durch die Klasse `DefaultDevice` realisierte Standardsteuerung für Simulationen (siehe Abschnitt 12.2) den Erweiterungspunkt, ohne eine Visualisierung im beschriebenen Sinne zu sein.

## 12.7 Vergleich mit anderen Arbeiten

Harel beschreibt in [HM03] einen Ansatz zur Anforderungsanalyse, der auf Szenarios basiert. Über einen Prototypen der Benutzungsschnittstelle interagiert der Kunde mit

einer partiellen Spezifikation des Systems. Einzelne Anwendungsfälle werden in das System „eingespielt“ und erweitern die Spezifikation. Analog können bereits beschriebene Anwendungsfälle „abgespielt“ und in der Benutzungsschnittstelle beobachtet werden. Der Ansatz basiert auf einer speziellen Art von Sequenzdiagrammen, sogenannten Live Sequence Charts, und ist somit nicht geeignet, die Semantik von Zustandsdiagrammen zu verdeutlichen.

Die existierende Werkzeuglandschaft für Zustandsdiagramme setzt sich im wesentlichen aus CASE-Werkzeugen zusammen, die neben Zustandsdiagrammen auch die weiteren Teilsprachen der UML unterstützen. Die kommerziellen Produkte Rational Rose [IBM96], Together [Tog06], MagicDraw [Mag06] und Poseidon [Pos06] sind typische Vertreter dieser Gruppe, ebenso das frei verfügbare Werkzeug ArgoUML [Arg06]. Sie alle fallen bedingt durch den großen Funktionsumfang unter das in Teil I der Arbeit beschriebene Komplexitätsproblem. In der Bedienung etwas zugänglicher, da weniger überladen, sind Zeichenwerkzeuge wie Visio [Vis06] oder Dia [Dia06]. Keines der Werkzeuge besitzt jedoch Funktionalität, die zur Unterstützung der Lehre von Zustandsdiagrammen eingesetzt werden könnte, etwa die von DAVE angebotenen Simulationen und Visualisierungen.

Werkzeuge die neben der Modellierung von Zustandsdiagrammen auch deren Ausführung in Form von generiertem Code unterstützen, sind Rose Real-Time [IBM96], iLogix Statemate [HLN+90] und iLogix Rhapsody [GHP02]. Real-Time Object-Oriented Modeling (ROOM) [SGW94] verwendet eine Variante von Zustandsdiagrammen, für die ebenfalls mithilfe der zugehörigen Werkzeuge Code generiert werden kann. All dies sind jedoch ebenfalls industrielle Werkzeuge, die unter dem Komplexitätsaspekt für die Lehre eher ungeeignet sind.

Das im akademischen Umfeld entstandene UML-Werkzeug Fujaba [NNZ00] arbeitet ebenfalls mit Code-Generierung, ist jedoch weniger komplex als die zuvor genannten und wird dementsprechend häufiger im Lehrbetrieb eingesetzt. Es enthält mit der Visualisierungskomponente Mr. Dobs eine Möglichkeit zur Veranschaulichung von Objektstrukturen, die ähnlich motiviert ist wie die in DAVE enthaltenen White-Box-Visualisierungen. Zustandsdiagramme können aber nicht visualisiert werden.

Mithilfe der auf Graphgrammatiken und Graphtransformationen basierenden Generatoren GenGed [Bar98] und DiaGen [MV95] wurden Werkzeuge zur Modellierung von Zustandsdiagrammen entwickelt [BE01, MH01], die eine interaktive Simulationen des Modells unterstützen. Beide Ansätze verwenden denselben Formalismus auch als Grundlage der Simulation, d.h. zur Beschreibung der Laufzeitsemantik. Visualisierungen werden nicht unterstützt.

# 13 Das Werkzeug PetRA

Ein weiteres in der Softwaretechnik neben Zustandsdiagrammen populäres Automatenmodell ist das der Petrinetze [Pet62]. Wo Zustandsdiagramme mit ihrer – auch im Unterschied zu einfachen endlichen Automaten – hohen Ausdrucksfähigkeit den konstruktiven Aspekt der Modellierung betonen, zeichnen sich die mit einer formalen Semantik unterlegten Petrinetze eher durch ihre Analysierbarkeit aus. Petrinetze besitzen eine lange Tradition in der Informatik. Dementsprechend umfangreich ist die Liste der existierenden Werkzeuge (einen Überblick bietet etwa die Petri Net World [Pet06]). Auch hier gilt jedoch, dass die Werkzeuge sich üblicherweise an Nutzer entweder aus der akademischen Forschung oder der industriellen Entwicklung richten. Zudem bauen viele Werkzeuge auf spezialisierten Netzklassen wie gefärbten Netzen [Jen96], Referenznetzen [Kum02], Funsoft-Netzen [EG95] oder anderen Highlevel-Netzen auf. Einfache Werkzeuge, die für zum Lehren der grundlegenden Netzklassen geeignet sind und gleichzeitig einen intuitiven Zugang sowie entsprechende didaktische Unterstützung bieten, sind rar. Gleichzeitig sind Petrinetze aber Bestandteil vieler Lehrveranstaltungen, in Dortmund etwa in den Bereichen Softwaretechnik und eingebettete Systeme, so dass Bedarf an solchen Werkzeugen besteht.

Das Werkzeug PetRA stellt einen Vorschlag dar, diese Lücke zu füllen. Es eignet sich zur Modellierung von Stellen-/Transitionsnetzen (und damit auch für den Spezialfall der Bedingungs-/Ereignisnetze). PetRA basiert in Teilen auf einem Petrinetz-Editor und -Simulator, der im Rahmen einer Diplomarbeit [Ami04] am Lehrstuhl für Software-Technologie der Universität Dortmund entwickelt wurde. Das Ziel dieser Diplomarbeit war die Übertragung der Erfahrungen mit dem Werkzeug Dave auf die Welt der Petrinetze. Insbesondere sollte das dort entwickelte Werkzeug ähnliche Möglichkeiten zur Simulation und Visualisierung bieten wie Dave, was auch prototypisch gelang. Im Anschluss an die Diplomarbeit wurde das Werkzeug dann um umfangreiche Analysemöglichkeiten erweitert [Fro04, Fro05, FP04, FP05a, FP05b, FP06]. Diese basieren formal und technisch auf Methoden der relationalen Algebra [SS93] (wenngleich dies für den Benutzer des Werkzeugs nicht relevant oder auch nur erkennbar ist), so dass sich der Name PetRA (Petrinetze und relationale Algebra) ergab.

## 13.1 Abstrakte und konkrete Syntax

Abbildung 13.1 zeigt das Metamodell für Stellen-/Transitionsnetze, das PetRA zugrundeliegt. Da diese Netzklasse nur über wenige syntaktische Elemente verfügt, fällt das

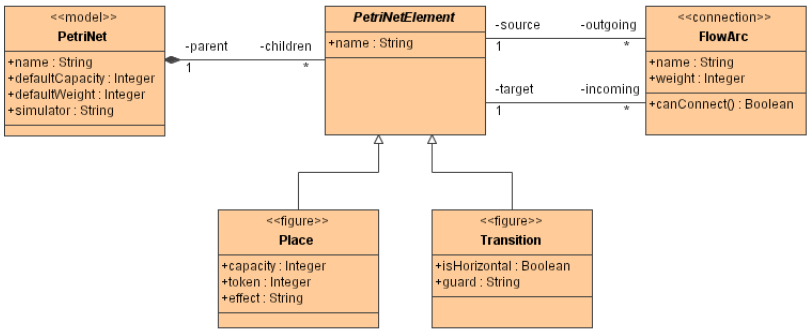



Abbildung 13.1: Metamodell der von PetRA unterstützten Petrinetze

Metamodell entsprechend einfach aus. Die Klasse `PetriNet` repräsentiert das Netz selbst und stellt die Wurzel des Modells dar. Die abstrakte Klasse `PetriNetElement` stellt die Gemeinsamkeit von Stellen und Transitionen dar, welche wiederum durch die Klassen `Place` und `Transition` repräsentiert werden. Ein Stellen-/Transitionsnetz ist im allgemeinen nicht hierarchisch, so dass Stellen und Transitionen immer unmittelbare Kinder der Modellwurzel selbst sind. Die Flussrelation wird über die Klasse `FlowArc` gebildet.

Für jede Stelle kann individuell eine Kapazität (Attribut `capacity`) und eine initiale Markenzahl (Attribut `token`) festgelegt werden. Der Wert -1 entspricht dabei einer unendlichen Kapazität. Jeder Flußkante kann ein Kantengewicht (Attribut `weight`) zugewiesen werden. Das Netz selbst bietet außerdem die Möglichkeit, Standardwerte für Kapazität und Gewicht anzugeben (Attribute `defaultCapacity` und `defaultWeight`). Diese gelten netzweit für alle Stellen und Flußkanten, für die keine individuellen Kapazitäten bzw. Gewichte festgelegt werden. Auch hier entspricht der Wert -1 einer unendlichen Kapazität. Durch Setzen einer Standardkapazität und eines Standardgewichts von jeweils 1 lässt sich das allgemeine Stellen-/Transitionsnetz sehr leicht auf ein Bedingungs-/Ereignisnetz einschränken. Für Transitionen existiert zudem ein Attribut `isHorizontal`, das zur Festlegung der Ausrichtung dient (ähnlich wie dies schon für die Synchronisationsbalken von Dave der Fall war). Das Attribut `simulator` des Netzes sowie die Attribute `input` und `output` der Transition dienen der Simulation. Sie werden in Abschnitt 13.4 erläutert.

Die einzige wesentliche syntaktische Randbedingung, die ein Petrinetz zu erfüllen hat, besteht darin, dass der Netzgraph bipartit sein muss. Flusskanten dürfen also nur Stellen mit Transitionen (oder umgekehrt) verbinden, nicht aber zwei Elemente gleichen Typs. Dies lässt sich sehr einfach durch Nutzung der Infrastruktur zur Syntaxprüfung (siehe Abschnitt 6.3) in der Klasse `FlowArc` umsetzen.

Die konkrete Syntax orientiert sich an den üblichen Gepflogenheiten bei Petrinetzen. Stellen werden durch Kreise dargestellt, Transitionen durch Rechtecke. Die Namen von Stellen und Transitionen werden in fetter Schrift neben bzw. in diesen Elementen angezeigt. Jede Stelle zeigt außerdem die Anzahl der vorhandenen Marken an. Bis zu sechs Marken werden der Anschaulichkeit halber in Form einzelner Würfelaugen dargestellt.

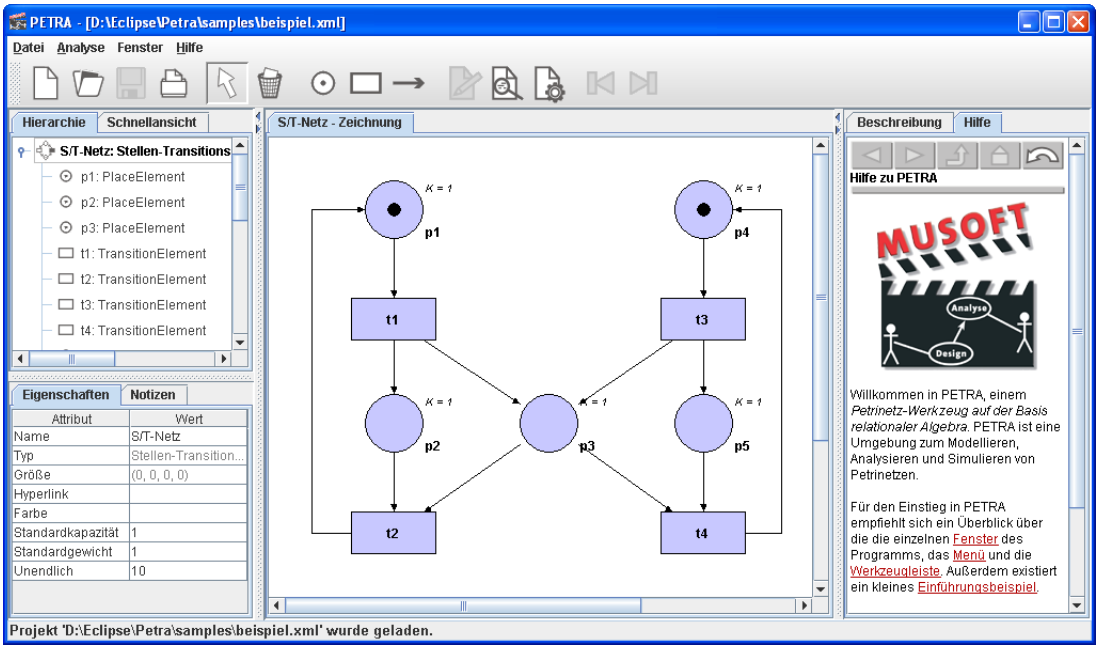



Abbildung 13.2: Bearbeiten eines Petrinetzes mit PETRA

Liegen mehr als sechs Marken auf der Stelle, dann wird die entsprechende Zahl ausgegeben. Die Kapazität einer Stelle wird nur dann angezeigt, wenn sie nicht unendlich ist, das Gewicht einer Kante nur dann, wenn es nicht gleich 1 ist. Da weder Stellen noch Transitionen andere Elemente enthalten können und für Stellen zudem eine gewisse Mindestgröße zur Anzeige der Marken benötigt wird, können beide nicht durch den Benutzer in der Größe verändert werden.

Abbildung 13.2 zeigt ein Bildschirmfoto der Bearbeitung eines einfachen Petrinetzes in PETRA. Die Benutzungsschnittstelle ergibt sich wieder weitgehend kanonisch aus dem LIMO-Framework. Unterschiede bei Werkzeugleiste und Zeichnung resultieren im wesentlichen aus der Notation. Die Werkzeugleiste enthält zusätzlich einige Schalter, um zwischen der Bearbeitung eines Netzes sowie dessen Analyse oder Simulation zu wechseln.

## 13.2 Analyse eines Petrinetzes

Da die Analysierbarkeit einer der zentralen Vorteile von Petrinetzen ist, unterstützt PETRA – wie die meisten anderen einschlägigen Werkzeuge auch – eine Reihe von Analysen statischer und dynamischer Eigenschaften eines gegebenen Netzes. Die Studierenden können diese Analysen nutzen, um häufig auftauchende Grundsituationen in S/T-Netzen,

etwa Konflikt und Kontakt, aufzudecken oder das Netz auf Deadlock-Freiheit zu überprüfen. Im einzelnen bietet PetRA nach dem Wechsel vom Bearbeitungsmodus in den Analysemodus die folgenden Analysen an (für Definitionen sei auf [Bau96] verwiesen):

- **Netzklassenanalyse** – Die Klasse eines Petrinetzes kann bestimmt werden. PetRA kennt das (Extended) Free-Choice-Netz, das Asymmetric-Choice-Netz, den (verallgemeinerten) Synchronisationsgraphen, die (verallgemeinerte) Zustandsmaschine sowie (trivialerweise) das einfache Netz.
- **Netzrand** – Der Vor- und Nachbereich einer Menge von Stellen oder Transitionen kann bestimmt werden, ebenso der Rand des Netzes sowie die Menge initialer oder terminaler Stellen oder Transitionen.
- **Kreisstrukturen** – Die im Netz enthaltenen einfachen, maximalen oder elementaren Kreise können bestimmt werden. Eine vorgegebene Menge von Elementen kann auf verschiedene Kreiseigenschaften geprüft werden.
- **Aktiviertheit** – Die Menge aller unter der aktuellen Markierung aktivierten Transitionen kann bestimmt werden, ebenso die Menge der nebenläufig aktivierten Transitionen. Außerdem können die Mengen von Stellen bzw. Transitionen berechnet werden, welche miteinander in Kontakt bzw. in Konflikt stehen.
- **Erreichbarkeit** – Die Nachfolgemarkierung für den Fall des Schaltens einer (zuvor auszuwählenden) Transition kann bestimmt werden, ebenso die Menge aller unmittelbar erreichbaren Nachfolgemarkierungen sowie der gesamte Erreichbarkeitsgraph.
- **Lebendigkeit** – Die starke bzw. schwache Lebendigkeit des modellierten Netzes kann geprüft werden. Des Weiteren können die Mengen toter, aktivierbarer oder lebendiger Transitionen sowie die Menge toter Markierungen bestimmt werden.
- **Fallen/Co-Fallen** – Die Mengen aller Fallen und Co-Fallen des Netzes können bestimmt werden. Zudem kann das Netz auf die Deadlock/Trap-Eigenschaft hin geprüft werden.

Während der Analyse wird das Netz in einen nur lesbaren Modus versetzt, damit keine Inkonsistenzen entstehen können. Ergebnisse der Analysen – auch geänderte Markierungen – werden auf dem Bildschirm angezeigt, wirken sich aber nicht auf das eigentliche Modell aus.

## 13.3 Simulation eines Petrinetzes

Mit der gleichen Motivation wie bei Dave – dem besseren Verständnis sowohl der allgemeinen Semantik von Petrinetzen als auch des konkreten modellierten Systems – wurde PetRA eine Simulationsmöglichkeit hinzugefügt. Diese ermöglicht das schrittweise Durchspielen des Netzes beginnend bei der im Modell vorgegebenen Anfangsmarkierung.

In der englischsprachigen Literatur wird dies gerne als „playing the token game“ bezeichnet.

Da Petrinetze im Unterschied zu Zustandsdiagrammen abgeschlossene Systeme sind, die nicht über Ein- und Ausgaben mit ihrer Umwelt kommunizieren, gestaltet sich die Steuerung der Simulation durch den Benutzer anders als bei DAVE. Das zusätzliche Steuerungsfenster wird zunächst nicht benötigt. Stattdessen interagiert der Benutzer im einfachsten Fall in jedem Schritt der Simulation unmittelbar mit der graphischen Darstellung des Petrinetzes (das auch hier für die Dauer der Simulation „eingefroren“ wird):

1. Die schaltbereiten Transitionen werden grün hervorgehoben. Der Benutzer wählt mit der Maus eine dieser Transitionen aus.
2. Die gewählte Transition schaltet. Konsumierte Marken werden aus dem Vorbereich entfernt. Produzierte Marken werden dem Nachbereich hinzugefügt.
3. Die neue Markierung wird angezeigt. Stellen, deren Markenzahlen sich im letzten Schritt der Simulation geändert haben, werden gelb hervorgehoben.

Das Wandern der Token wird auf Wunsch animiert dargestellt, um die Arbeitsweise der Schaltregel zu verdeutlichen. Ist diese Animation aktiviert, dann bewegen sich konsumierte Marken entlang der Flußkanten aus dem Vorbereich in die Transition. Produzierte Marken bewegen sich analog von der Transition zu deren Nachbereich. Ähnlich wie bei DAVE wird auch hier der Simulationsverlauf aufgezeichnet und ist in einer Tabelle einsehbar. Während der Simulation besteht zudem die Möglichkeit, beliebig viele Schritte zurückzunehmen (bis zur Anfangsmarkierung) oder erneut zu gehen (bis zur letzten erreichten Markierung) und so verschiedene Varianten von Abläufen zu untersuchen.

Die Simulation macht sich technisch das Vorhandensein der Analysemöglichkeiten zunutze. Im Prinzip werden hier lediglich wiederholt die Analysen zur Bestimmung der aktivierten Transitionen und der Nachfolgemarkierung angestoßen. Zudem ist jederzeit ein Wechsel zwischen Simulation und Analyse möglich, so dass die Eigenschaften einer bestimmten, während der Simulation erreichten Markierung untersucht werden können.

## 13.4 Visualisierung der Simulation

Auch für PETRA bestand der Wunsch, der Simulation des abstrakten Petrinetzes eine anschauliche Visualisierung eines modellierten oder durch das Modell gesteuerten Systems gegenüberstellen zu können. Ein erster Versuch einer Umsetzung wurde im Rahmen der zuvor erwähnten Diplomarbeit [Ami04] unternommen. Dort wurde prototypisch das aus DAVE stammende Beispiel der Waschmaschine an Petrinetze angepasst. Dabei zeigte sich, dass die Übertragung der Idee der White-Box-Visualisierungen unmittelbar möglich ist, während Black-Box-Visualisierungen zusätzliche Anforderungen stellen.

### 13.4.1 White-Box-Visualisierungen

White-Box-Beispiele, also solche zu rein demonstrativen Zwecken, sind auch mit Petrinetzen problemlos umzusetzen. Die Visualisierung kann die einzelnen Stellen und Transitionen des Netzes beobachten und entsprechend reagieren, etwa durch eine Änderung der Anzeige. Der Fall entspricht weitgehend dem bei Zustandsdiagrammen, so dass an dieser Stelle auf eine Abbildung verzichtet wird. Es sei jedoch angemerkt, dass die Schnittstelle zwischen Modell und Visualisierung erneut sehr umfangreich werden kann, da der Visualisierung im Extremfall sämtliche Stellen und Transitionen des Modells bekannt sein müssen.

Im Kontext von Petrinetzen bieten sich Visualisierungen des White-Box-Typs unter anderem zur Verdeutlichung grundlegender Problemstellungen wie der speisenden Philosophen oder Produzenten und Konsumenten an. Sie sind jedoch – aus den gleichen Gründen, die bereits im Kontext von Dave diskutiert wurden – weniger gut zum Stellen von Übungsaufgaben geeignet.

### 13.4.2 Black-Box-Visualisierungen

Black-Box-Beispiele sind im Kontext von Dave (siehe Abschnitt 12.3) als didaktisch interessanterer Beispieltyp herausgestellt worden, da sie mehr Freiheiten bei der Modellierung eröffnen und sich damit besser für Aufgaben eignen. Sie sind jedoch für Petrinetze nicht unmittelbar nutzbar, da sie eine Schnittstelle voraussetzen, über die Netz und Umwelt kommunizieren. Petrinetze – speziell die grundlegenden Netzklassen – werden jedoch meist als abgeschlossene Systeme betrachtet, denen eine solche Schnittstelle fehlt. Zur Unterstützung dieses Beispieltyps mussten die von PetRA unterstützten S/T-Netze also entsprechend erweitert werden. Dabei waren einige Randbedingungen zu beachten:

- Die Erweiterungen sollten sich möglichst unauffällig in die Welt der Petrinetze einfügen. Die Studierenden sollten nicht den Eindruck erhalten, dass die ausschließlich vom Werkzeug verwendeten Erweiterungen integraler Bestandteil von S/T-Netzen sind. Mit dieser Randbedingung schienen völlig neue Netzelemente nicht ratsam.
- Die Erweiterungen durften nichts an der Semantik des Petrinetzes ändern. Insbesondere die Schaltregel des S/T-Netzes musste weiterhin unverändert gelten. Damit verboten sich Lösungen, die etwa einzelne Stellen des Netzes zur Schnittstelle erklären, so dass die Außenwelt dort Marken entnehmen oder hinzufügen kann.

Innerhalb der Diplomarbeit [Ami04] wurde ein Ansatz gewählt, der diesen Randbedingungen genügte, aber für praktische Beispiele zu unflexibel war. Der Benutzer konnte den Verlauf der Simulation nur über das Ändern der Anfangsmarkierung bestimmen, jedoch während der Simulation keinen Einfluss mehr nehmen. Für PetRA wurde deshalb nach einer Alternative gesucht. Netzklassen mit Schnittstellen zur Außenwelt findet man etwa im Bereich der Automatisierungstechnik. Dort werden erweiterte Petrinetze

genutzt, um eingebettete Systeme zu beschreiben, zu analysieren und anschließend Code für Microcontroller zu generieren.

- Orth [Ort05] verwendet Stellen-/Transitionsnetze mit Ein-/Ausgabe (STIO) zur Modellierung. Das System kommuniziert mit seiner Umwelt über ausgezeichnete Stellen, deren Markenzahl von der Umgebung registriert bzw. geändert werden kann. Der Ansatz scheint für ein Werkzeug wie PetRA nicht geeignet zu sein. Willkürliche Änderungen der Markierung kollidieren mit der grundlegenden Netzsemantik, da durch sie Markierungen entstehen können, die nicht im Erreichbarkeitsgraphen des Netzes enthalten sind. In jedem Fall würde bei den studentischen Nutzern des Werkzeugs ein verfehlter Eindruck der Petrinetz-Semantik entstehen.
- Frey [Fre00, FM00] verwendet signal-interpretierte Petrinetze (SIPN), eine Erweiterung von Bedingungs-/Ereignisnetzen. Jeder Transition ist eine boolesche Funktion über einer Menge von binären Eingabesignalen zugeordnet. Eine schaltbereite Transition schaltet, wenn diese Funktion den Wert `true` liefert. Zudem wird jeder Stelle eine Funktion zugewiesen, die eine Menge von boolschen Ausgabesignalen belegt, solange die Stelle markiert ist. Der Ansatz steht im Einklang mit der grundlegenden Petrinetz-Semantik. Das Netz bleibt analysierbar. SIPN sind jedoch nur für B/E-Netze definiert, nicht für die allgemeineren S/T-Netze. Zudem ist die Beschreibung der Ausgabefunktionen aufwendig, da das Netz in jeder erreichbaren Markierung eine vollständige und eindeutige Belegung der Ausgabesignale liefern muss.

Der Ansatz, der schließlich für PetRA gewählt wurde, lehnt sich an SIPN an, modifiziert diese jedoch aus pragmatischen Gründen leicht:

- Jeder Transition wird eine Funktion über eine Menge von Eingabenvariablen zugewiesen, die *Bedingung* (im Metamodell das Attribut `guard`). Die Eingabevariablen hängen von der Visualisierung ab. Die Auswertung der Bedingung muss in einem booleschen Wert resultieren. Die leere Bedingung trifft immer zu.
- Jeder Stelle wird eine Belegung einer Menge von Ausgabevariablen zugewiesen, der *Effekt* (im Metamodell das Attribut `effect`). Die Ausgabevariablen hängen wieder wieder von der Visualisierung ab. Der Effekt kann sich über eine spezielle Variable `token` auf die aktuelle Markenzahl der Stelle beziehen.
- In jedem Schritt der Simulation des Petrinetzes wird unter allen schaltbereiten Transitionen, deren Bedingung zutrifft, eine Auswahl zum Schalten getroffen. Die genaue Schaltstrategie ist abhängig von der Visualisierung. Im allgemeinen Fall wird nichtdeterministisch eine der Transitionen gewählt.
- Bei jeder Änderung der Markenzahl einer Stelle wird deren Effekt produziert, d.h. die enthaltenen Wertzuweisungen an die Variablen werden ausgeführt. Dies schließt den Fall der Initialisierung des Petrinetzes mit der Startmarkierung ein.

Die Schnittstelle einer Black-Box-Variante des von Dave bekannten Fernseher-Beispiels reduziert sich damit auf vier Variablen. Zwei boolesche Eingabevariablen *POWER* und

*STATION* werden von den beiden entsprechenden Tasten des Geräts kontrolliert. Eine boolesche Ausgabevariable *CRT* kontrolliert die Bildröhre des Geräts. Eine ganzzahlige Ausgabevariable *CHANNEL* wählt einen Sender. Die Visualisierung ist vom Modell entkoppelt, wodurch verschiedene Lösungen möglich werden. Fehlerhaften Modellierungen kann ebenfalls Rechnung getragen werden, etwa indem wieder der Fall vorgesehen wird, dass das Gerät eingeschaltet ist, aber kein Sender gewählt wurde. Das Vorgehen ist insgesamt ähnlich dem bei Dave, so dass an dieser Stelle auf eine Abbildung verzichtet werden soll.

Es sei angemerkt, dass diese Erweiterungen – wie oben gefordert – keine Modifikation der Netzsemantik nach sich ziehen. Effekte haben trivialerweise keinerlei Auswirkung auf die Schaltregel, sondern dienen lediglich der Beobachtbarkeit der Simulation. Bedingungen haben ebenfalls keine Auswirkung auf die grundsätzliche Schaltregel, sondern lediglich auf die Auswahl der zu schaltenden Transition(en) unter allen schaltbereiten Transitionen. Damit bleibt der Zustandsraum des Netzes insgesamt unverändert. Dies gilt allerdings nicht mehr, wenn durch Bezüge zwischen Bedingungen und Effekten versteckte Abhängigkeiten erzeugt werden, die nicht Teil des Netzes sind.

Ein weiteres Problem, das in diesem Kontext auftritt, ist der fehlende Zeitbegriff von S/T-Netzen. Bei vielen praktischen Beispielen – etwa der bereits bekannten Waschmaschine – ist es hilfreich, wenn das Modell Elemente enthalten kann, die von der Zeit abhängig sind. Dies hätte jedoch für PetRA einen Wechsel auf eine höhere Netzklasse mit Zeitbegriff anstelle der grundlegenden S/T-Netze bedeutet und Konsequenzen insbesondere für die Analysierbarkeit des Netzes gehabt. Die Arbeit folgt daher für PetRA dem Vorschlag von [Bau96], den Zeitbegriff explizit durch Teilnetze zu modellieren, die mit einer festgelegten Zeitspanne identifizierte *Ticks* zählen.

## 13.5 Nutzung von Java-Code in Petrinetzen

Ähnlich wie bei Dave ist auch bei PetRA eine Möglichkeit wünschenswert, das Modell mit Java-Code zu verbinden, um stärker an das Vorwissen der Studierenden anzuknüpfen. Der Ansatz harmoniert gut mit den an S/T-Netzen vorgenommenen Erweiterungen um Bedingungen und Effekte. Bedingungen werden dazu als Java-Ausdrücke angenommen, die in einem boolschen Wert resultieren. In Effekten werden über reine Variablenzuweisungen hinaus beliebige Java-Anweisungen erlaubt, die ausgeführt werden. Zur Auswertung kann wieder der Java-Interpreter Beanshell [Bea06] verwendet werden.

## 13.6 Code-Generierung für LEGO Mindstorms

Auch die Idee der Generierung von Code für LEGO Mindstorms, die bereits bei Dave erfolgreich umgesetzt wurde, lässt sich auf PetRA übertragen. Benötigt wird zunächst eine Schnittstelle zwischen dem S/T-Netz und dem Mindstorms-System. Diese kann im

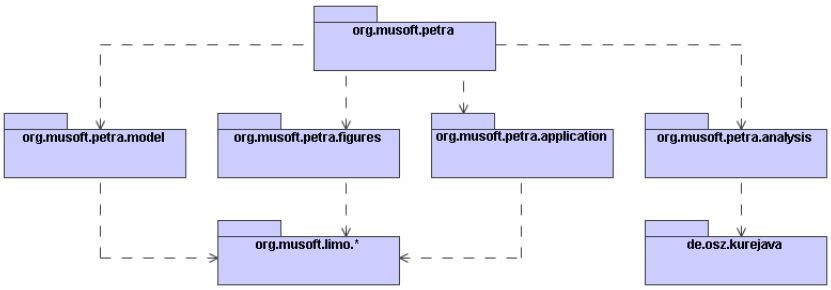



Abbildung 13.3: Grobe Architektur von PetRA

einfachsten Fall über drei Eingabevariablen für die Sensoren des Systems und drei Ausgabevariablen für die Aktoren bestehen. Ein mit entsprechenden Bedingungen und Effekten annotiertes Netz wird in Java-Quellcode umgewandelt. Zusätzlich wird – ähnlich wie in der Diplomarbeit [Moc06] – eine feste Ausführungsmaschine für S/T-Netze benötigt, die auf den Microcontroller des Mindstorms-Systems geladen wird. Sowohl die Ausführungsmaschine als auch der netzspezifische Code fallen deutlich einfacher aus als bei Zustandsdiagrammen, da Syntax und Semantik von S/T-Netzen weniger komplex sind.

## 13.7 Architektur von PetRA

Abbildung 13.3 zeigt die grobe Architektur von PetRA in Form eines Paketdiagramms mit den wichtigsten Abhängigkeiten. Der Modellierungsanteil weicht strukturell nicht signifikant von Dave ab. Die Funktionalität wurde – wie üblich – auf die Pakete `model`, `figures` und `application` aufgeteilt. Der Analyseteil von PetRA – und damit auch die Simulation – findet sich im Paket `analysis` wieder. Die folgenden Abschnitte beschreiben einige ausgewählte Details dieser Komponente.

### 13.7.1 Analyse und Simulation

Dieser Teil von PetRA basiert auf Methoden zur Analyse von Petrinetzen mit Mitteln der relationalen Algebra [BvKU96]. Beim Wechsel in den Analyse- oder Simulationsmodus wird das Petrinetz intern in eine relationale Repräsentation überführt. Anschließend können über das Menü die zuvor beschriebenen Analysen angestoßen werden. Jede Analyse besteht üblicherweise aus einem oder mehreren relationalen Termen, die ausgewertet werden. Das Ergebnis liegt zunächst wieder in relationaler Form vor, kann aber mit geringem Aufwand in die graphische Darstellung zurückübersetzt werden. In vielen Fällen

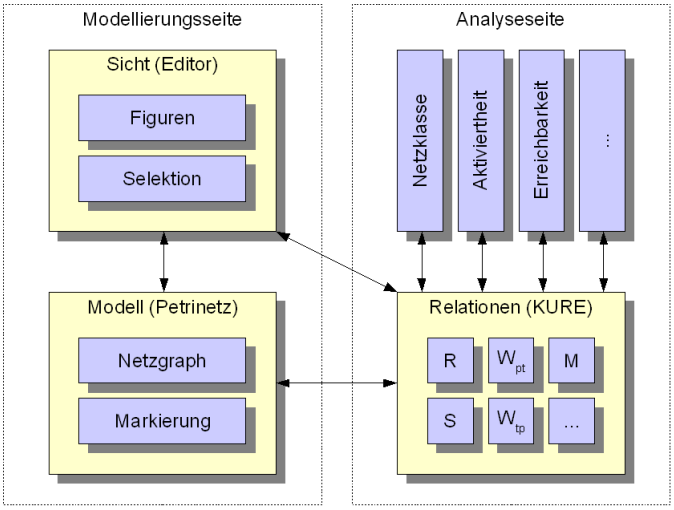



Abbildung 13.4: Zusammenspiel von Modellierung und Analyse in PetRA

handelt es sich etwa um Mengen von Stellen oder Transitionen, die farblich hervorgehoben werden, oder Markierungen, die in die Stellen eingetragen werden. In Einzelfällen werden Texte auf dem Bildschirm ausgegeben.

Abbildung 13.4 verdeutlicht das Zusammenspiel zwischen den einzelnen Komponenten von PetRA schematisch. Details zur relationalen Analyse von Petrinetzen finden sich in [Fro04, Fro05, FP04, FP05a, FP05b, FP06].

Mechanisiert wird das Rechnen mit Relationen durch die Java-Bibliothek KURE, die im Rahmen einer Diplomarbeit [Szy03] an der Universität Dortmund entwickelt wurde und die auf dem an der Christian-Albrechts-Universität zu Kiel entwickelten Relationenwerkzeugs RelView [BBMS01] basiert. KURE kommt damit im Rahmen von PetRA eine ähnliche Rolle zu wie der Ausführungskomponente für Zustandsdiagramme bei Dave.

### 13.7.2 Hinzufügen eigener Analysen

PetRA besitzt einen Erweiterungspunkt, der dem Benutzer das Hinzufügen neuer Analysen ermöglicht. Abbildung 13.5 zeigt dessen Struktur. Jede Analyse wird in einer Java-Klasse realisiert, welche die Schnittstelle `RelviewTask` implementiert. Deren Methode `execute()` setzt die eigentliche Berechnung um, kapselt also die Auswertung der jeweiligen relationalen Terme. Die Methode `report()` ist für die Rückübersetzung des Ergebnisses in die graphische Darstellung oder für textuelle Ausgaben zuständig. Der Zugriff auf die relationale Kodierung des Netzes und den Zustand der Simulation geschieht über die Klasse `RelviewSession`. Durch Hinzufügen der neuen Analyse zur Datei `analysis.xml`, die das Analysemenü beschreibt (nach den Vorgaben von Anhang A), wird diese dem Werkzeug bekanntgemacht.

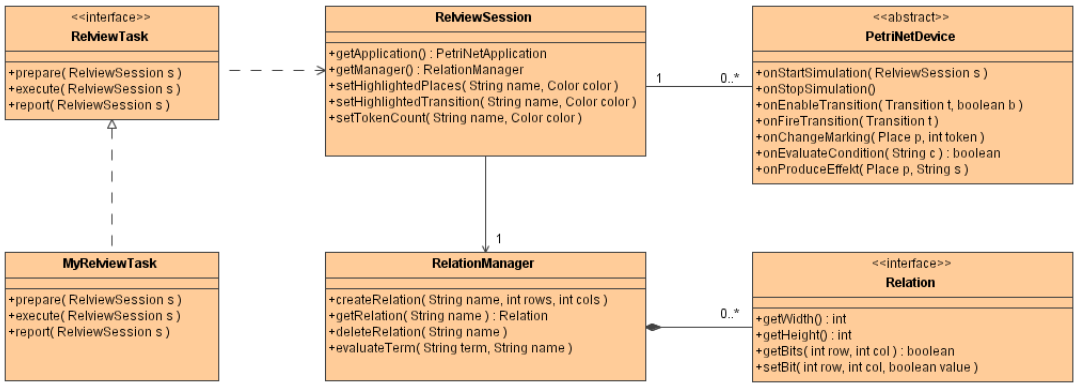



Abbildung 13.5: Schnittstelle für Analysen und Visualisierungen

### 13.7.3 Hinzufügen eigener Visualisierungen

Ein zweiter Erweiterungspunkt von PetRA bietet die Möglichkeit, dem Werkzeug neue Visualisierungen hinzuzufügen. Die Vorgehensweise entspricht weitgehend der bei Dave, so dass hier lediglich ein grober Überblick über die Struktur gegeben werden soll. Basis jeder Visualisierung ist die abstrakte Klasse `PetriNetDevice` (siehe Abbildung 13.5). Diese Klasse enthält eine Reihe von Methoden, die während der Simulation aufgerufen werden, um Änderungen am Zustand von Stellen oder Transitionen mitzuteilen. Die Namen der meisten Methoden sollten selbsterklärend sein, so dass an dieser Stelle ausschließlich `onEvaluateGuard` und `onExecuteAction` Erwähnung finden sollen. Diese beiden Methoden werden zur Auswertung einer Bedingung und zur Ausführung einer Aktion aufgerufen und realisieren damit die Schnittstelle zwischen einer Black-Box-Visualisierung und einem Petrinetz.

## 13.8 Vergleich mit anderen Werkzeugen

Wie zuvor erwähnt, ist die Werkzeuglandschaft für Petrinetze sehr reichhaltig. Die folgende Auswahl erhebt damit nicht den Anspruch, umfassend oder auch nur repräsentativ zu sein. Sie orientiert sich vielmehr an den in Teil I der Arbeit diskutierten Kriterien sowie daran, ob Publikationen zu einem Werkzeug existieren.

### 13.8.1 Petrinetz-Editoren und -Simulatoren

**CPN Tools.** CPN Tools [RWL+03] ist ein Werkzeug zur Modellierung, Analyse und Simulation hierarchischer, zeitbehafteter, gefärbter Petrinetze. CPN Tools ersetzt das

ältere, weit verbreitete Werkzeug Design/CPN [SCK97]. Es bietet in etwa dessen Funktionalität – neben der Modellierung und verschiedenen Simulationsmodi (interaktiv, automatisch) existieren Möglichkeiten zur Analyse des Zustandsraums. Ähnlich wie PetRA kann auch CPN Tools zwischen Analyse und Simulation wechseln, so dass interessante, während der Simulation erreichte Markierungen näher untersucht werden können. Im Unterschied zu seinem Vorgängerwerkzeug bricht CPN Tools weitgehend mit den Konventionen graphischer Benutzungsschnittstellen [BLMA+01]. So existieren etwa keine traditionellen Menüs, der Arbeitsplatz verwendet eine Metapher aus *Mappen* und *Blättern*, und für die Interaktion wird zusätzlich zur Maus entweder eine Tastatur oder ein anderes Eingabegerät – die Entwickler empfehlen einen Trackball – benötigt. Wenngleich die Bedienung interessante Ansätze aufweist und möglicherweise bei häufiger Nutzung des Werkzeugs eine Effizienzsteigerung bedeutet, stellt sich bei einem kurzzeitigen Einsatz doch die Frage nach dem benötigten Einarbeitungsaufwand.

**INA.** Der an der Humboldt-Universität zu Berlin entwickelte Integrated Net Analyzer (INA) [SR00, RS00] unterstützt die Analyse und Simulation von S/T-Netzen und gefärbten Netzen. Die Analysemöglichkeiten sind mit denen von PetRA vergleichbar. Da INA jedoch gänzlich ohne graphische Oberfläche auskommt und ausschließlich mit textuellen Kommandos bedient wird, scheint das Werkzeug unter dem Gesichtspunkt der Bedienbarkeit für die Lehre eher ungeeignet zu sein.

Sowohl CPN Tools als auch INA sind native Anwendungen und somit nur auf einer begrenzten Anzahl von Plattformen lauffähig. Die im Folgenden betrachteten Werkzeuge sind plattformunabhängig in Java implementiert.

**PIPE.** Der Platform Independent Petri Net Editor (PIPE) [Pip06] wurde im Rahmen zweier studentischer Projektgruppen am Imperial College in London entwickelt [BCC+03, BCC+04]. Mit Hilfe von PIPE können S/T-Netze sowie stochastische Petrinetze modelliert, analysiert und simuliert werden. Die Analysemöglichkeiten bleiben hinter denen von PetRA zurück. Untersucht werden können zum Beispiel der Netztyp, die verschiedenen Invarianten, aktivierte Transitionen sowie der Zustandsraum. Es existiert aber ein Erweiterungspunkt, über welchen dem Werkzeug neue Analysen in Form von Java-Klassen hinzugefügt werden können. Die Simulationsmöglichkeit entspricht der von PetRA, d.h. aktivierte Transitionen werden hervorgehoben und können zum Schalten ausgewählt werden. Es findet jedoch keine Animation der Token statt – nur das Ergebnis eines Simulationsschrittes wird angezeigt.

**RENEW.** Der Reference Net Workshop (RENEW) [Ren06, KWD+04] unterstützt primär die Modellierung und Simulation von Referenznetzen, einer speziellen Klasse von Netzen, die mit Java-Code annotiert werden können [Kum02]. Die offene Architektur des Werkzeugs erlaubt jedoch auch die Integration weiterer Netzklassen. Das Werkzeug bietet eine teilweise sehr fortgeschrittene Funktionalität, die für industrielle Systementwicklung in den Bereichen Workflow-Management und Multiagentensysteme interessant ist, etwa das Durchführen von langwierigen Simulationen auf einem Server und ohne eine graphische Oberfläche. Außerdem unterscheidet RENEW explizit zwischen modellierten Netzen und deren Instanzen, während PetRA immer exakt eine Instanz des Netzes

simuliert oder analysiert. RENEW unterstützt ausschließlich Simulationen, aber keine Analysen des Netzes, was möglicherweise auf die Komplexität der Netzklasse zurückzuführen ist. Interessanterweise verwendet RENEW ebenfalls die Bibliothek JHotDraw [JHD06, Kai04] als Basis des graphischen Editors.

**PNK.** Der Petri Net Kernel (PNK) [KW01] ist kein Werkzeug im eigentlichen Sinne, sondern eher ein Framework zur Entwicklung von Petrinetz-Werkzeugen. PNK stellt Infrastruktur zur Beschreibung der Syntax (Eigenschaften von Stellen, Transitionen und Flusskanten) und Semantik (Aktiviertheit, Schaltregel) von Netzklassen bereit. Aufbauend auf dieser Infrastruktur können mit geringem Aufwand eigene Netzklassen definiert und simuliert werden, wobei grundlegende Netzklassen wie S/T-Netze bereits implementiert sind. PNK enthält auch generische Anwendungsfunktionalität, etwa einen einfachen graphischen Editor, der jedoch in seinen Möglichkeiten und seiner Bedienbarkeit hinter PetRA zurückbleibt. PNK wurde ursprünglich in Python entwickelt und später auf Java portiert. Zu den Anwendungen, die auf der Basis von PNK entwickelt wurden, gehört ein Front-End für INA.

### 13.8.2 Ansätze zur Visualisierung

Keines der bislang diskutierten Werkzeuge bietet eine Möglichkeit zur Veranschaulichung der Simulation eines Petrinetzes. Jedoch existieren auch hierzu bereits verschiedene Ansätze. Allen ist das Ziel gemein, die Kluft zwischen dem abstrakten Modell und der Anwendungsdomäne eines Benutzers zu überbrücken. Das Mittel dazu ist jeweils die Abbildung von Netzteilen auf Elemente, die der Anwendungsdomäne entstammen.

Kindler und Páles beschreiben in [KP04] eine Methode zur dreidimensionalen Visualisierung von Petrinetzen. Sie identifizieren die Marken des Netzes mit Objekten der realen Welt und weisen ausgezeichneten Stellen eine Animation zu. Sowohl Objekte als auch Animationen werden mithilfe der Virtual Reality Markup Language (VRML) [VRM97] beschrieben. Animationen dauern an, solange eine Stelle markiert ist, und können theoretisch auch ein Ergebnis liefern, das in die Schaltregel einer Transition im Nachbereich der Stelle eingeht. So könnte der Benutzer – etwa durch Mausklicks – den Fortgang der Simulation beeinflussen. Die Methode benötigt aufwendige Annotationen eines Petrinetzes. Der Nutzen der Dreidimensionalität ist nicht erkennbar.

Ermel et al. stellen in [CE01] einen Ansatz zur Spezifikation und Implementierung sogenannter Animationssichten von Petrinetzen vor. Die einzelnen Markierungen eines Netzes werden auf graphische Ausgaben abgebildet. Der Benutzer lässt während der Simulation Transitionen über eine spezielle Benutzungsschnittstelle schalten. Der Ansatz baut auf dem Graphtransformationswerkzeug GenGEd [Bar98] auf. Es werden insgesamt drei Graphgrammatiken zur Umsetzung eines Beispiels benötigt: Eine Sprachgrammatik beschreibt die allgemeine Syntax von S/T-Netzen. Eine Verhaltensgrammatik beschreibt die Laufzeitsemantik eines Netzes. Eine Animationsgrammatik beschreibt mögliche Übergänge zwischen den Zuständen der Animation. Verhaltens- und Animationsgrammatik

müssen vom Entwickler für jedes zu simulierende Netz einzeln erstellt und konsistent gehalten werden, damit das abstrakte Modell und die anschauliche Animation parallel laufen. Der Ansatz ist formal interessant, da er durchweg mit Graphtransformationen arbeitet, aber aufgrund des hohen Aufwands nicht praxistauglich.

Innerhalb des CPN-Werkzeugs ExSpect [vdAdCG+00, ExS06] existiert ein sogenanntes *Dashboard*, in welchem verschiedene einfache Anzeigelemente (z.B. Lampen oder Skalen) oder Eingabeelemente (z.B. Schalter) an die Stellen und Transitionen eines Netzes gebunden werden können. Für das Werkzeug Design/CPN existiert mit Mimic/CPN [RS95] eine Erweiterung, die beliebige graphische Objekte mit den Stellen und Transitionen des Netzes verknüpft und diese für Eingaben bzw. Ausgaben verwendet. Mimic/CPN wurde erfolgreich zur Modellierung des Verhaltens von Mobiltelefonen eingesetzt [LTX02], wobei ein originalgetreues Bild des in Entwicklung befindlichen Gerätes als „Benutzungsschnittstelle“ des Netzes diente.

Alle vorgestellten Ansätze haben in der Terminologie dieser Arbeit White-Box-Charakter, d.h. sie dienen im Wesentlichen zur Veranschaulichung der Semantik eines gegebenen Netzes. Zur Realisierung von Black-Box-Beispielen sind sie nicht geeignet.

# 14 Das Werkzeug SAM

Die beiden zuvor beschriebenen Werkzeug DAVE und PETRA dienten im Wesentlichen zur Modellierung, Simulation und Visualisierung des Verhaltens eines Softwaresystems. Strukturelle Aspekte wurden nicht einbezogen, spielen aber bei der Entwicklung größerer oder verteilter Softwaresysteme – und damit auch in der Lehre – eine wichtige Rolle. Gerade bei verteilten Systemen ist häufig das Zusammenspiel aller Komponenten so komplex, dass eine Planung der Architektur des Gesamtsystems notwendig ist, ehe mit einer Modellierung und Implementierung einzelner Subsysteme begonnen werden kann.

Der Begriff einer Softwarearchitektur ist innerhalb der bestehenden Literatur nicht völlig eindeutig definiert. Es besteht jedoch in Standardwerken wie [PW92, SG95, BMR+96] weitgehend Konsens darüber, dass die Architektur eines Softwaresystems dessen wesentliche strukturelle und dynamische Eigenschaften in einer frühen Phase der Entwicklung festlegt. Das Abstraktionsniveau einer Architektur liegt im Allgemeinen relativ hoch. Man redet hier eher über die Verschaltung einzelner Komponenten zu einem System als über die konkreten Klassen, mit deren Hilfe diese Komponenten (bei einer objektorientierten Lösung) realisiert werden. Dieser Sichtweise hat sich der Autor mit dem Softwarearchitektur-Modellierer (SAM), der in den folgenden Abschnitten vorgestellt wird, angeschlossen.

SAM betrachtet neben den von DAVE und PETRA unterstützten dynamischen auch die strukturellen Aspekte eines Softwaresystems. Das Werkzeug basiert auf der klassischen Sichtweise von Architekturen als einer Menge von Komponenten und Konnektoren, wie sie in [SG95] vertreten wird. Nicht zuletzt zur Unterstützung einer Simulation, wie sie bereits in DAVE und PETRA erfolgreich implementiert wurde, war aber eine formalere Sprache als die dort beschriebene nötig. Die Entscheidung fiel letztlich zugunsten einer Untermenge der Notation, die in der UML 2.0 [OMG05a] zur Beschreibung architektureller Aspekte von Systemen neu eingeführt wurde und die in Teilen auf Real-Time Object-Oriented Modeling (ROOM) [SGW94] bzw. UML for Real-Time [SR03, Lyo03] zurückgeht.

## 14.1 Abstrakte und konkrete Syntax

Die von SAM unterstützte Notation zur Beschreibung von Architekturen verwendet insgesamt drei verschieden Diagrammarten:

- Auf der obersten Ebene dient eine spezielle Art von Klassendiagramm zur Spezifikation der strukturellen Einheiten des Systems, die hier *Kapseln* genannt werden[1]. Jeder Kapsel werden in diesem Diagramm *Ports* hinzugefügt, über die sie mit der Außenwelt kommunizieren kann. Jeder Port „spricht" ein bestimmtes Protokoll, das ebenfalls hier definiert wird, und nimmt dabei eine bestimmte Rolle ein (normal und konjugiert, etwa analog zu Sender und Empfänger). Zudem wird hier festgelegt, welche Kapsel das Gesamtsystem (analog zu einer Java-Klasse, deren `main`-Methode ausgeführt wird) repräsentiert.
- Das strukturelle Innenleben jeder Kapsel wird über ein Diagramm spezifiziert, das sich am ehesten mit einem Objekt- oder Kollaborationsdiagramm vergleichen läßt. Hier werden *Instanzen* von Kapseln kombiniert und über *Konnektoren* untereinander sowie mit den Ports der sie umgebenden Kapsel verbunden. Damit zwei Ports miteinander verbunden werden können, müssen sie das gleiche Protokoll verwenden, aber entgegengesetzte Rollen einnehmen.
- Das reaktive Verhalten jeder Kapsel – und damit implizit auch des gesamten Systems – wird über ein Zustandsdiagramm beschrieben. Die Ereignisse, die das Zustandsdiagramm einer Kapsel als Ausgabe produziert, können über Konnektoren an andere Kapseln weitergegeben und dort konsumiert werden. Dazu existiert innerhalb jeder Kapsel ein sogenannter *End-Port*, der das Zustandsdiagramm selbst repräsentiert. Dieser kann mit allen anderen Ports unabhängig von deren Protokoll und Rolle verbunden werden.

Abbildung 14.1 zeigt das Metamodell der Notation, die von SAM verwendet wird.

Die Klasse `Architecture` bildet die Wurzel des Modells. Sie enthält eine Menge von Protokollen sowie eine Menge von Kapseln (Klassen `Protocol` und `Capsule`). Für jedes Protokoll wird die Menge der möglichen eingehenden und ausgehenden Signale spezifiziert (Attribute `incoming` und `outgoing`). Eine Instanz eines Protokolls ist ein sogenannter Port, der durch die gleichnamige abstrakte Klasse repräsentiert wird und in normaler oder konjugierter Form vorliegen kann (Attribut `conjugate`). Jeder Kapsel ist eine Menge von spezialisierten Ports zugeordnet, welche die Schnittstelle dieser Kapsel zur Außenwelt bilden (Klasse `CapsulePort`).

Die Strukturbeschreibung jeder Kapsel wird durch Kollaboration anderer Kapseln gebildet (Klasse `Collaboration` bzw. Hilfskonstrukt `Border`, das den Rahmen der Kapsel repräsentiert). Innerhalb einer solchen Kollaboration finden sich primär Instanzen anderer Kapseln wieder (Klasse `Instance`), aber auch drei spezialisierte Arten von Ports:

- Relais-Ports dienen zur Kommunikation zwischen Innen- und Außenwelt einer Kapsel. Für jeden äußeren Port der Kapsel im Klassendiagramm existiert ein Relais-Port in der Kollaboration.

[1] Die UML-Spezifikation verwendet hier den Begriff des *Parts*, aber aufgrund der leichten Verwechselbarkeit mit den ebenfalls vorkommenden *Ports* bevorzugt diese Arbeit den aus der älteren Literatur [SR03, Lyo03] stammenden Begriff. Eine Alternative wäre der naheliegende Begriff der *Komponente* gewesen. Dieser ist aber in der UML bereits semantisch mit einem anderen Konzept belegt.

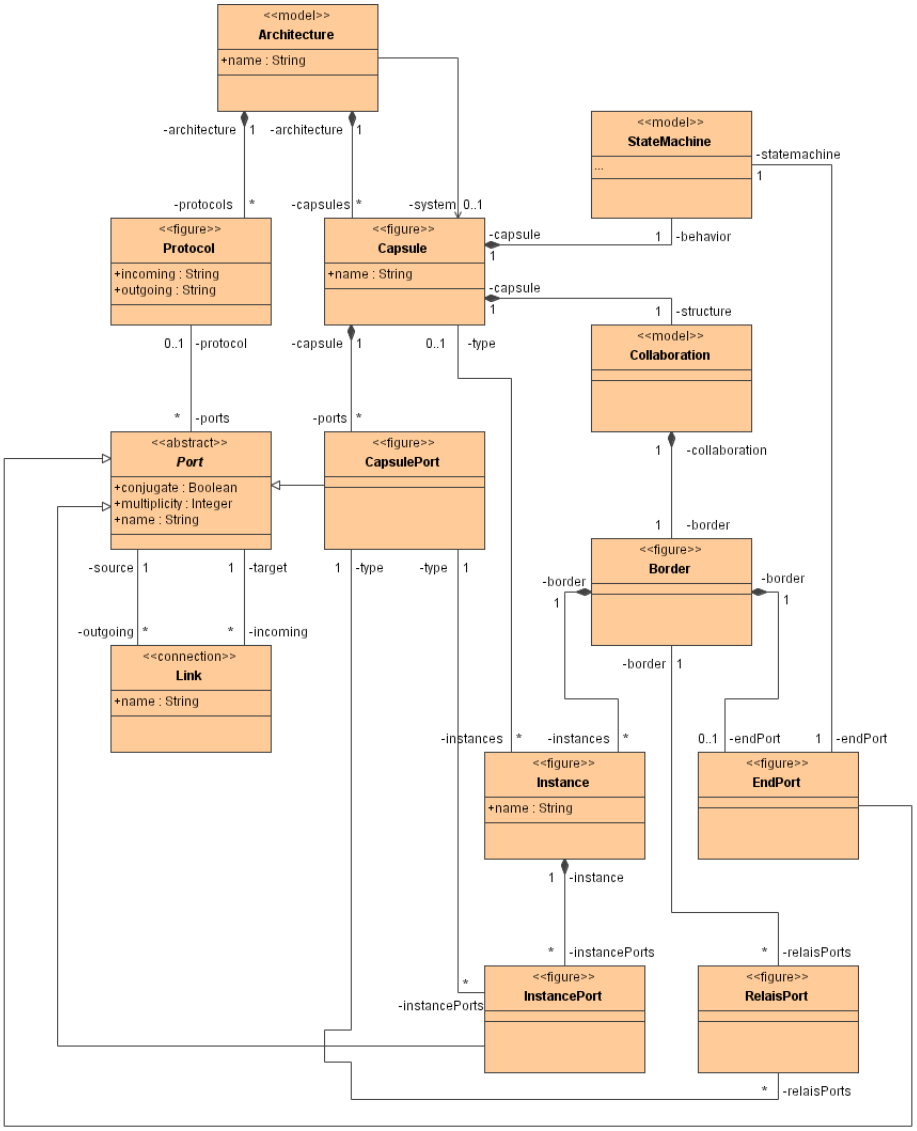



Abbildung 14.1: Metamodell der von SAM unterstützten Architekturen

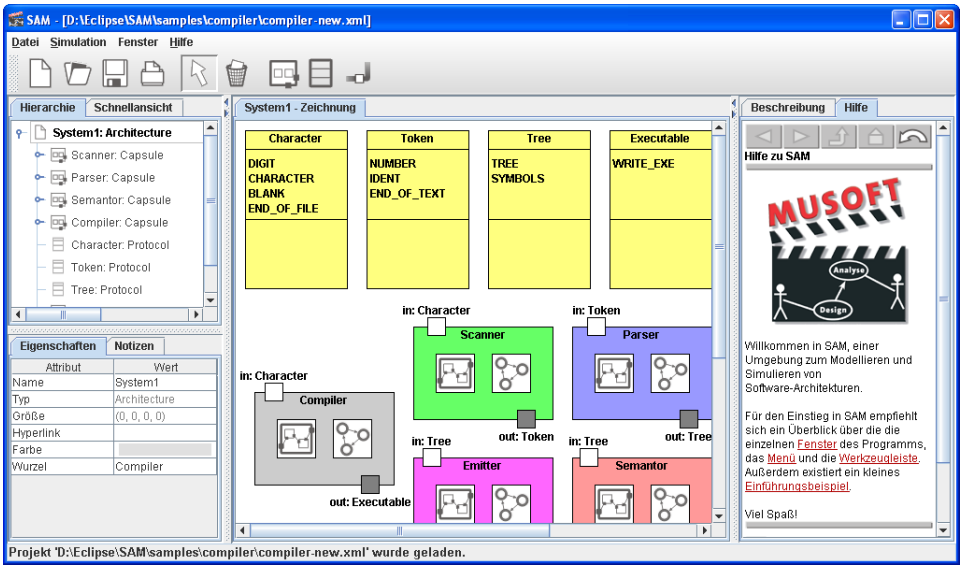



Abbildung 14.2: Bearbeiten einer Architektur in SAM

- Instanz-Ports entstehen bei der Instanziierung einer Kapsel. Für jeden äußeren Port einer Kapsel im Klassendiagramm besitzt die Instanz einen entsprechenden Instanz-Port.
- Ein End-Port dient als Repräsentation des Zustandsdiagramms einer Kapsel. Eine Kapsel, die ohne eigene Verhaltensbeschreibung auskommt, benötigt keinen End-Port.

Innerhalb einer Kollaboration können Ports (über Instanzen der Klasse `Link`) verbunden werden, wobei die zuvor erwähnten Restriktionen bezüglich der Ports eingehalten werden müssen. Die Verhaltensbeschreibung einer Kapsel geschieht über ein Zustandsdiagramm (Klasse `UMLStateMachine`). Dieser Teil des Metamodells entspricht dem von DAVE, so dass an dieser Stelle auf die entsprechende Abbildung 12.1 verwiesen sei.

Die konkrete Syntax von SAM orientiert sich wieder an den Vorgaben der UML. Protokolle werden ähnlich normalen Klassen dargestellt – die beiden Partitionen dienen hier zur Aufteilung in eingehende und ausgehende Signale. Kapseln und deren Instanzen werden als Rechtecke dargestellt, wobei die Ports durch kleinere, auf dem Rand verschiebbare Quadrate repräsentiert werden. Die konkrete Syntax der Zustandsdiagramme entspricht der von DAVE.

Abbildung 14.2 zeigt ein Bildschirmfoto der Bearbeitung einer Architekturbeschreibung in SAM. Das Klassendiagramm definiert die einzelnen Grundkomponenten eines einfachen Compilers – lexikalische, syntaktische und semantische Analyse sowie die Code-Generierung – und die Protokolle, die deren Zusammenspiel reglementieren.

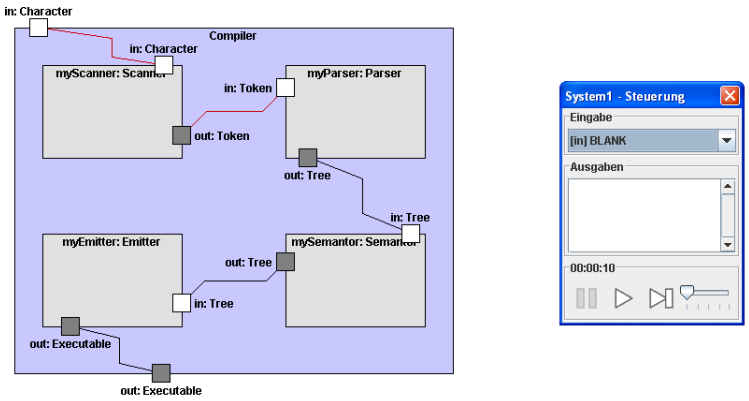



Abbildung 14.3: Simulation einer Architektur in SAM

## 14.2 Simulation einer Architektur

Aus den gleichen Beweggründen, die schon bei DAVE und PETRA angeführt wurden, unterstützt auch SAM die Simulation des modellierten Systems: Den Studierenden soll über das Verständnis der reinen Syntax der Architekturbeschreibung hinaus ein Einblick in deren Laufzeitsemantik ermöglicht werden. Tatsächlich gilt dieses Argument sogar für eine Architekturbeschreibung in weitaus stärkerem Maße, denn das Gesamtverhalten einer aus mehreren, nebenläufig agierenden Kapseln bestehenden Architektur erschließt sich ungleich schwerer als das eines einzelnen Zustandsdiagramms. Bei SAM spielt die Simulation also eine noch wichtigere Rolle als bei den anderen Werkzeugen. Gleichzeitig ist sie technisch schwieriger umzusetzen, da von jeder Kapsel potenziell beliebig viele Instanzen existieren können und das Zusammenspiel all dieser Instanzen und der daraus resultierende Gesamtzustand des System korrekt sein müssen.

Basis für die Architektursimulation ist wieder die Semantik von Zustandsdiagrammen. Jede Instanz einer Kapsel besitzt eine eigene Instanz des Zustandsdiagramms der zugehörigen Kapsel[2]. Jede Kapsel verwaltet zudem eine Warteschlange von Ereignissen, die vom Zustandsdiagramm schrittweise konsumiert und verarbeitet werden. Aus den Konnektoren zwischen einzelnen Kapseln ergibt sich ein Fluß von Ereignissen innerhalb des Systems.

Beim Start der Simulation erscheint das von DAVE bekannte Steuerungsfenster (siehe Abbildung 14.3 – hier erkennt man auch deutlich, dass es sich bei dem Compiler um eine „Pipes and Filters“-Architektur handelt). Mit ihm können Eingabeereignisse für die nach außen geführten Ports des Systems produziert werden. Gelangen im Verlauf der Simulation Ausgabeereignisse an diese Ports, dann werden sie angezeigt. Ebenfalls beim Start

[2] Korrekt betrachtet teilen sich alle Instanzen das gleiche Zustandsdiagramm, aber jede verwaltet ihre eigene Zustandskonfiguration.

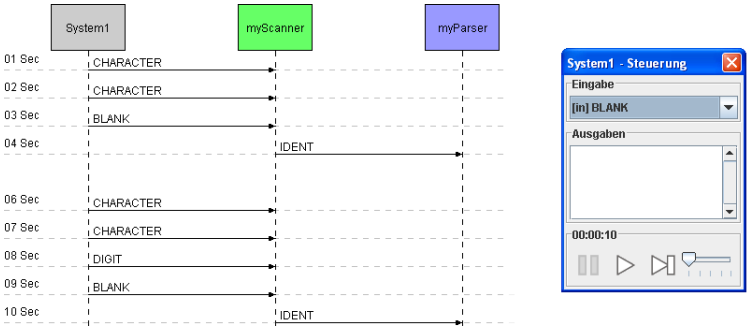



Abbildung 14.4: Sequenzdiagramm einer Simulation in SAM

der Simulation erscheint ein neues Diagramm, das eine Instanzsicht des Systems bietet. In diesem Diagramm sind die Kapseln der obersten Ebene des Systems zu sehen. Ein Doppelklick auf eine der Kapseln zeigt deren strukturelle Innereien bzw. das Zustandsdiagramm inklusive der aktuellen Konfiguration an. Um den Ereignisfluss im Gesamtsystem leichter nachvollziehen zu können, wird die Kommunikation zwischen Kapseln während der gesamten Simulation in einem Sequenzdiagramm aufgezeichnet (siehe Abbildung 14.4). Die Menge der Kapseln, die an diesem Diagramm beteiligt sind, kann frei gewählt werden.

## 14.3 Visualisierung der Simulation

Es bietet sich an, die Konzepte zur Visualisierung einer Simulation, die bei DAVE und PETRA sehr erfolgreich umgesetzt wurden, auch für SAM zu nutzen. Speziell mit DAVE ist SAM inhaltlich wie technisch so eng verwandt, dass eine Übertragung beider Arten von Visualisierungen mit geringem Aufwand möglich ist. Sinnvoll ist eine Abstützung des Zusammenspiels von Modell und Simulation auf Ebene der Kapseln:

- White-Box-Visualisierung – In diesem Fall wird die Visualisierung mit einer einzelnen Kapsel des Systems oder mit dem Gesamtsystem identifiziert. Sie reagiert auf Änderungen der Zustandskonfiguration.
- Black-Box-Visualisierung – In diesem Fall wird die Visualisierung als zusätzliche Kapsel in das System eingefügt oder kommuniziert mit dem Gesamtsystem. Sie reagiert auf Ereignisse anderer Kapseln.

Es sei angemerkt, dass sich im Fall von SAM damit insbesondere die Möglichkeit ergibt, verschiedene Visualisierungen für einzelne Subsysteme des Gesamtsystems zu verwenden (etwa einen Geldautomaten und einen Server einer Bank) oder eine einzelne Visualisierung mehrfach zu verwenden (etwa eine Kreuzung mit vier identischen Ampeln). Auf eine detaillierte Ausarbeitung der Ideen wird an dieser Stelle aus Platzgründen verzichtet.

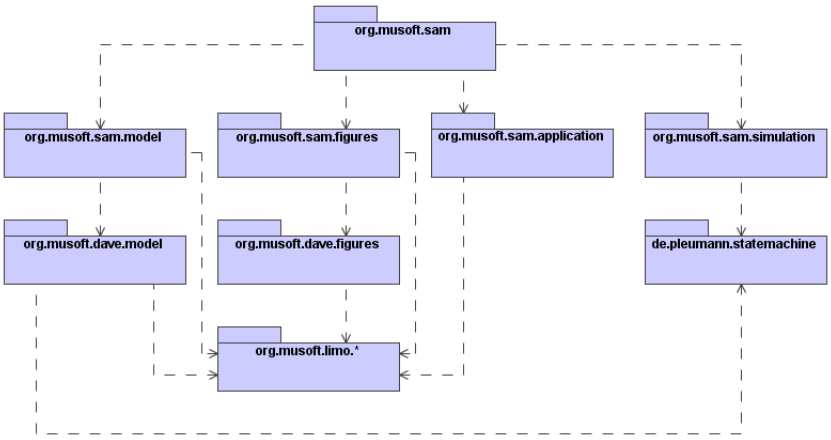



Abbildung 14.5: Grobe Architektur von SAM

## 14.4 Nutzung von Java-Code in Architekturen

Auch für SAM ist die Möglichkeit der Integration von Java-Code in das Modell eine sinnvolle Erweiterung. Wie bei DAVE und PETRA müssen dazu nur Bedingungen und Aktionen geeignet interpretiert werden. Mit den Mitteln, die SAM zur Strukturierung von Systemen bietet (Modularisierung, Wiederverwendung), ergibt sich damit die Möglichkeit, relativ komplexe Softwaresysteme durch eine Mischung von Modellierung und Programmierung aufzubauen und anschließend interpretativ auszuführen. Während Struktur und Kontrollfluss auf einer groben Ebene durch das Modell repräsentiert werden können, drückt der Java-Code eher algorithmische Details aus.

## 14.5 Architektur

Die Architektur von SAM folgt weitgehend dem bereits bekannten Schema und soll deshalb nur kurz diskutiert werden. Abbildung 14.5 zeigt die grobe Struktur der wichtigsten Pakete und Abhängigkeiten. Die Funktionalität wurde – wie üblich – auf die Pakete `model`, `figures` und `application` aufgeteilt, wobei die Implementierung der abstrakten und konkreten Syntax von Zustandsdiagrammen unmittelbar aus DAVE stammt. Der Simulationsteil baut wieder auf der Ausführungsmaschine für Zustandsdiagramme auf, die dem Projekt „Modellieren in einer virtuellen Welt“ entstammt [WPD01].

## 14.6 Vergleich mit anderen Arbeiten

Viele existierende Architekturwerkzeuge verwenden neben einer Strukturbeschreibung durch – wie auch immer geartete – Komponenten spezielle Varianten von Zustandsdiagrammen zur reaktiven Beschreibung des Systemverhaltens. Hierzu zählen insbesondere Statemate [HLN+90], Rhapsody [GHP02], ROOM [SGW94] und Rational Rose Technical Developer [IBM96]. Diese Werkzeuge wurden bereits im Kontext von DAVE diskutiert, so dass sie an dieser Stelle nur der Vollständigkeit halber erwähnt werden sollen. Sie unterstützen üblicherweise die Simulation des modellierten Systems, eventuell mit dem Zwischenschritt einer Code-Generierung, jedoch nicht in der Anschaulichkeit von SAM. Ihre Notationen entsprechen zudem – mit der Ausnahme von Rose – nicht dem aktuellen UML-Standard. Neuere Version von UML-Werkzeugen wie Together [Tog06] und MagicDraw [Mag06] unterstützen teilweise schon die Architektur-Notation der UML 2.0, die auch in SAM verwendet wird, erlauben jedoch keine Simulation.

Unter den älteren Ansätzen zur Modellierung von Architekturen sind einige, die sowohl Struktur als auch Verhalten berücksichtigen und zudem durch ein Werkzeug unterstützt werden:

**Rapide.** Rapide ist eine ereignisbasierte, objektorientierte Sprache zum Prototyping von Systemarchitekturen [LKA+95]. Die Sprache beruht auf ähnlichen Konstrukten wie die aus der UML 2.0 stammende Notation, die von SAM verwendet wird. Komponenten und ihre Schnittstellen zur Außenwelt werden auf der Typebene definiert und zur Bildung von konkreten Architekturen instanziiert. Das reaktive Verhalten einer Komponente wird über Regeln beschrieben, die – ähnlich Zustandsdiagrammen, aber einfacher – eingehende und ausgehende Ereignisse in Beziehung setzen. Rapide wird durch Werkzeuge unterstützt, die eine Simulation des Systems ähnlich zu SAM ermöglichen. Das zentrale Ergebnis einer solchen Simulation ist ein sogenanntes *Poset*, eine halbgeordnete Menge der Ereignisse, die während der Simulation auftreten. Das Poset ist in erster Näherung vergleichbar mit dem Sequenzdiagramm, das SAM zur Aufzeichnung der Simulation verwendet. Eine Anschaulichkeit der Simulation ist jedoch ebenso wenig ein Ziel von Rapide wie ein Einsatz zu Lehrzwecken.

**UniCon.** UniCon ist eine am Software Engineering Institute (SEI) der Carnegie Mellon University entstandene Sprache zur Modellierung von Software-Architekturen [SDK+95]. Der Fokus von UniCon liegt dabei auf der Systemstruktur, die – ähnlich wie bei SAM – durch Komponenten, Konnektoren und deren Konfiguration beschrieben wird. Komponenten können primitiv oder unterstrukturiert sein. Ein graphisches Werkzeug ermöglicht die Neudefinition von Komponenten sowie deren Auswahl aus einer Menge vordefinierter Komponenten. Für das gesamte System kann C-Code generiert werden, so dass mit dem Umweg über den C-Compiler auch eine Ausführbarkeit des Modells gegeben ist. UniCon stellt jedoch keine dedizierten Sprachkonstrukte zur Beschreibung des Systemverhaltens (analog zu den Zustandsdiagrammen) bereit. Dieser Teil des Systems muss durch Annotation der einzelnen Komponenten in C implementiert werden.

**Wright.** Die ebenfalls am SEI entwickelte Sprache Wright [AG97] stellt eine formale Basis zur Beschreibung und Analyse von Softwaresystemen bereit. Wright berücksichtigt sowohl die Struktur als auch das Verhalten von Systemen. Die Strukturbeschreibung geschieht weitgehend kanonisch durch Komponenten, Konnektoren und deren Konfiguration. Zur Beschreibung des Verhaltens dient eine Untermenge von CSP [Hoa78]. Diese erlaubt die Spezifikation erlaubter oder erwarteter Interaktionsfolgen. Die Abstützung von Wright auf CSP macht das Gesamtsystem einer formalen Analyse zugänglich.

Über die genannten Arbeiten hinaus existiert eine Vielzahl von älteren Ansätzen zur Architektur-Beschreibung, die ebenfalls durch Werkzeuge unterstützt werden. Einen guten Überblick liefert etwa [SW99]. Die zugrundeliegenden Sprachen stammen teilweise noch aus den frühen 90er Jahren, sind also vor (oder parallel zu) der UML entstanden und syntaktisch wie semantisch weit von diesem Standard entfernt. Auch wenn einige der Ansätze formal interessant sind, eignen sie sich damit nicht als Alternative zur Notation von SAM. Viele der zugehörigen Werkzeuge werden zudem nicht mehr gewartet oder sind gar nicht mehr zugänglich.

# 15 Das Werkzeug ProMod / ProTut

Bei der Software-Entwicklung spielt neben der Modellierung von funktionalen, strukturellen und dynamischen Aspekten des eigentlichen Systems auch die Planung und Organisation des Entwicklungsprozesses eine wichtige Rolle. Dies gilt umso mehr für große Projekte mit vielen Beteiligten oder die heute vielfach übliche räumlich oder zeitlich verteilte Entwicklung. Dementsprechend beziehen viele Lehrveranstaltungen im Bereich der Softwaretechnik oder angrenzender Ingenieurwissenschaften den Prozess betreffende Aspekte – insbesondere dessen Modellierung – mit ein. Neben klassischen Prozessmodellen wie dem Wasserfallmodell [Roy87], dem Spiralmodell [Boe86] oder dem für deutsche Behörden verbindlichen V-Modell [VMo06, RB06] spielt dabei der Unified Process [JBR98] eine immer größere Rolle. Der Unified Process ist ein generisches und iteratives Prozessmodell, das vor seiner Anwendung an die konkreten Bedürfnisse des durchzuführenden Projekts angepasst wird.

Um Studierende mit diesem „Maßschneidern“ des Unified Process vertraut zu machen, wurden im Rahmen des Projekts MuSofT spezielle Werkzeuge auf Basis des LiMo-Frameworks entwickelt [APS04a, APS04b]. Der Prozessmodellierer für den Unified Process (ProMod) ermöglicht die graphische Modellierung eines Softwareentwicklungsprozesses. Für Studierende ist dies aufgrund ihres begrenzten Erfahrungsschatzes meist eine schwierige Aufgabe. Als Hilfestellung gibt das Werkzeug die „strategischen“ Elemente des Prozesses (Zyklen, Phasen und Iterationen) bereits vor. Im einfachsten Fall sind lediglich die Workflows (Aktivitäten und Artefakte) auszugestalten, um ein komplettes Prozessmodell zu erhalten.

Ähnlich wie bei den zuvor diskutierten Werkzeugen Dave, PetRA und Sam war auch für den Prozessmodellierer eine Art Ausführungskomponente wünschenswert, um das erstellte Prozessmodell mit Leben zu füllen. Eine Simulations- oder Analysemöglichkeit bot sich in diesem Fall jedoch nicht unbedingt an. Zum einen fehlt dem Unified Process eine definierte Semantik, die ihn diesen Möglichkeiten zugänglich machen würde. Zum anderen liegen die Probleme beim Lehren des Unified Process (oder eines anderen Prozessmodells) weniger auf der semantischen als vielmehr auf der pragmatischen Ebene: Die Studierenden müssen Erfahrungen damit sammeln, ein Projekt „entlang“ des modellierten Prozesses durchzuführen. Diese Erfahrungen sollten idealerweise im Anschluss reflektiert und in die Planung folgender Projekte einbezogen werden [KA02], ähnlich wie dies auch der Personal Software Process (PSP) [Hum96] propagiert.

Um dies zu unterstützen, wurde als ergänzendes Werkzeug der Prozesstutor (ProTut) entwickelt, der sich in erster Näherung als einfaches Projektmanagementwerkzeug für den

Unified Process beschreiben lässt. Basierend auf einem mit ProMod maßgeschneiderten Prozessmodell leitet ProTut die Studierenden bei der Durchführung eines Projektes an und bietet Hilfestellung bei der Verfolgung und Visualisierung des Fortschritts, etwa in Form von rollenbezogenen Aufgabenlisten oder der schematischen Darstellung des Verlaufs durch ein Gantt-Diagramm [Wil03]. Im universitären Kontext bietet sich der kombinierte Einsatz beider Werkzeuge innerhalb eines Praktikums an [KSS04, KMGSD04].

## 15.1 Abstrakte und konkrete Syntax

Abbildung 15.1 zeigt das Metamodell des Unified Process, das innerhalb von ProMod und ProTut Verwendung findet. Ein Prozess (Klasse `ProcessModel`) setzt sich aus einer beliebigen Anzahl von Zyklen (Klasse `Cycle`) zusammen. Jeder Zyklus untergliedert sich in fünf Phasen (Klasse `Phase`) sowie fünf Meilensteine, die das Ergebnis einer jeden Phase markieren (Klasse `Milestone`). Eine Phase besteht aus einer beliebigen Anzahl von Iterationen (Klasse `Iteration`), die ihrerseits bis zu fünf Workflows besitzen (Klasse `Workflow`). Zyklen, Iterationen und Workflows besitzen eine Reihenfolge (Assoziationsenden `predecessor` bzw. `successor`).

Ein Workflow ist ein bipartiter Graph, der sich aus Aktivitäten und Artefakten zusammensetzt (Klassen `Aktivity` und `Artifact`, abstrahiert durch `WorkflowElement`). Aktivitäten können Artefakte voraussetzen und produzieren. Dieser Datenfluss innerhalb eines Workflows wird über die Kanten des Graphen (Klasse `WorkflowConnection`) modelliert. Jedes Artefakt ist eindeutig einem Workflow zugeordnet. Um Artefakte auch über die Grenzen eines Workflows hinaus referenzieren zu können, existiert das Konzept eines Artefakt-Stellvertreters (Klasse `ArtifactProxy`), der auf bestehende Artefakte verweist.

Jedem Artefakt ist ein Anfangs- und ein Enddatum zugeordnet (Attribute `startDate` und `endDate`), durch welches das maximale Bearbeitungsintervall festgelegt wird. Jede Aktivität besitzt einen Bearbeitungsstatus (Attribut `status`). Die meisten anderen Elemente des Modells besitzen diese Attribute ebenfalls, jedoch in abgeleiteter (also berechneter) Form: Das Bearbeitungsintervall auf den höheren Stufen des Prozessmodells ergibt sich automatisch durch die Vereinigung aller Intervalle auf tieferliegenden Stufen. Der Bearbeitungsstatus ergibt sich durch eine ähnliche Berechnungsvorschrift.

Einer Aktivität wird eine Rolle zugeordnet, die für deren Bearbeitung zuständig ist. Die verfügbaren Rollen können für jeden Prozess individuell festgelegt werden (Klasse `Role`), wobei als Vorgabe eine Menge üblicher Rollen (Designer, Implementierer, Tester, ...) angenommen wird.

Zur Begrenzung der Komplexität und da einige Elemente in einem Ausbildungskontext keine Entsprechung finden, wurde das Metamodell gegenüber dem eigentlichen Unified Process an mehreren Stellen leicht eingeschränkt. Die wesentlichen Unterschiede liegen in

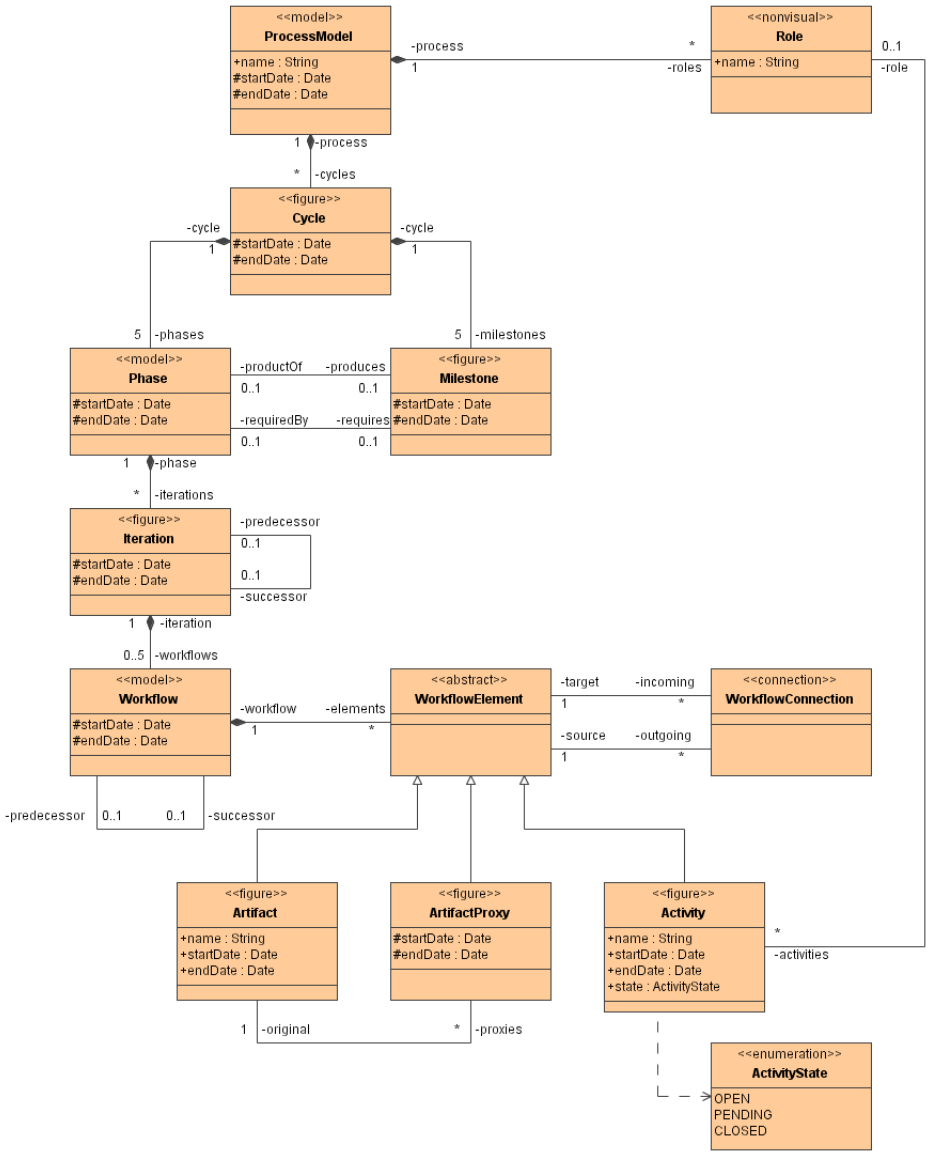



Abbildung 15.1: Metamodell von ProMod und ProTut

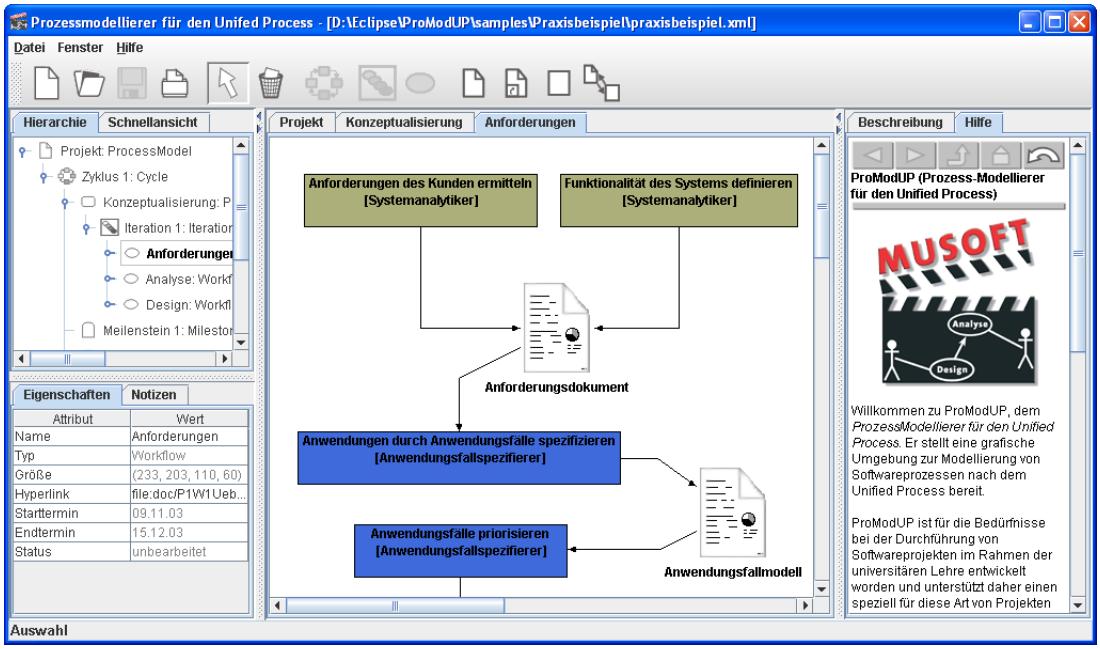



Abbildung 15.2: Maßschneidern eines Prozesses mit ProMod

der Menge der Workflows einer Iteration (es werden nur die fünf zentralen Workflows Anforderung, Analyse, Entwurf, Implementierung und Test unterstützt) und in der Menge der vorgegebenen Rollen (die aber angepasst werden kann).

Die konkrete Syntax setzt sich aus nur wenigen Figuren zusammen: Die strategischen Elemente des Prozessmodells (Zyklen, Phasen, Iterationen) werden durch Rechtecke dargestellt, Workflows durch Ellipsen. Innerhalb der Workflows kommt eine Notation zum Einsatz, die grob an Aktivitätsdiagramme erinnert. Es wurde Sorge dafür getragen, durch eine konsistente Farbgebung der Elemente die Orientierung innerhalb des Prozesses zu unterstützen: Jede Iteration besitzt die gleiche Farbe wie die Phase, der sie untergeordnet ist. Jede Aktivität wird in der Farbe der zugehörigen Rolle dargestellt.

## 15.2 Benutzungsschnittstelle

Die Benutzungsschnittstelle von ProMod ergibt sich völlig kanonisch aus dem Framework. Über die Standardelemente (Menü, Werkzeugleiste, Navigator, Inspektor, Arbeitsbereich und Hypertext) hinaus wurde keine neue Funktionalität aufgenommen. Abbildung 15.2 zeigt ein Bildschirmfoto von ProMod während der Bearbeitung eines Anforderungs-Workflows. Man erkennt verschiedene Aktivitäten und Artefakte sowie deren Abhängigkeiten.

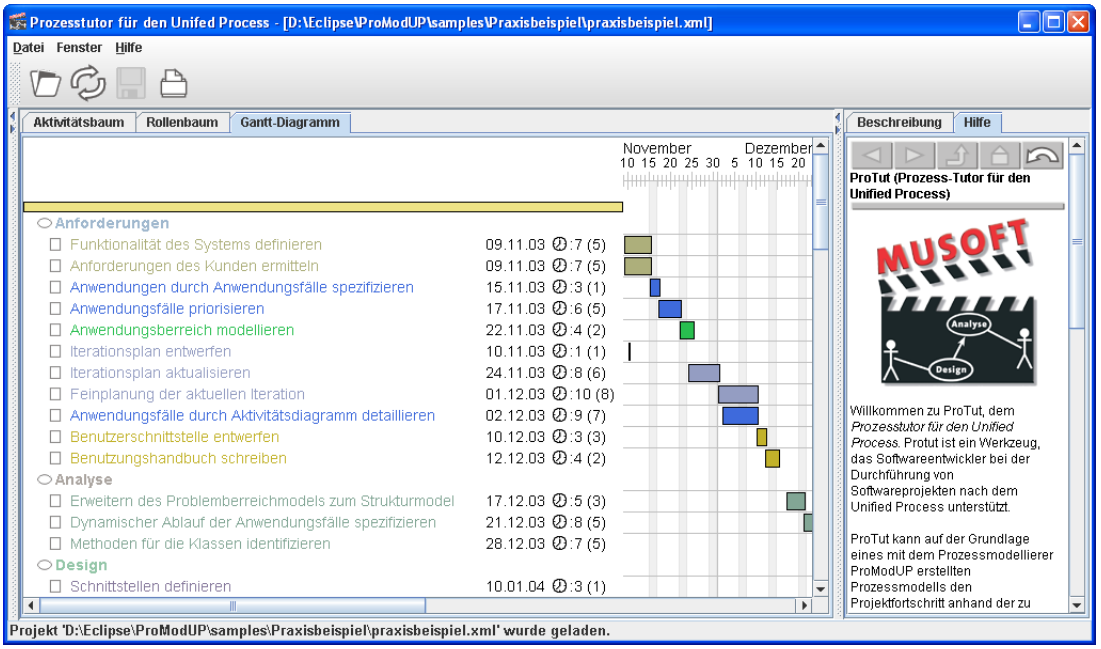



Abbildung 15.3: Begleitung eines Prozesses mit ProTut

Die Benutzungsschnittstelle des zweiten Werkzeugs ProTut unterscheidet sich in einigen Elementen relativ stark von den anderen LiMo-Anwendungen. Auffällig ist das Fehlen sämtlicher Modellierungselemente: Navigator, Inspektor, Arbeitsbereich und die entsprechenden Teile der Werkzeugleiste fehlen vollständig, da sie während der Durchführung des Projektes nicht mehr benötigt werden. Stattdessen wurden verschiedene neue Sichten integriert, die den Projektfortschritt wiedergeben. Abbildung 15.3 zeigt ein entsprechendes Bildschirmfoto von ProTut, bei dem ein Gantt-Diagramm eines Beispielprojekts dargestellt wird. In der linken oberen Ecke des Fensters wird eine Liste unerledigter Aufgaben für einen zuvor ausgewählten Workflow angezeigt. Darunter werden die Vorbedingungen einer ausgewählten Aktivität aufgelistet, also die Dokumente, die in diese Aktivität eingehen. Neben dem Gantt-Diagramm existieren noch zwei andere Übersichten: Ein Aktivitätsbaum zeigt die hierarchische Struktur des gesamten Prozesses, ein Rollenbaum eine Gliederung der Aktivitäten nach zuständigen Rollen (beide werden in der Abbildung durch das Gantt-Diagramm verdeckt).

## 15.3 Implementierung

Abbildung 15.4 zeigt die grobe Paketstruktur von ProMod und ProTut. Die Architektur von ProMod entspricht weitgehend den bisher diskutierten Werkzeugen (Pakete `model`, `figures` und `application`). ProTut nutzt die Metamodell-Implementierung von

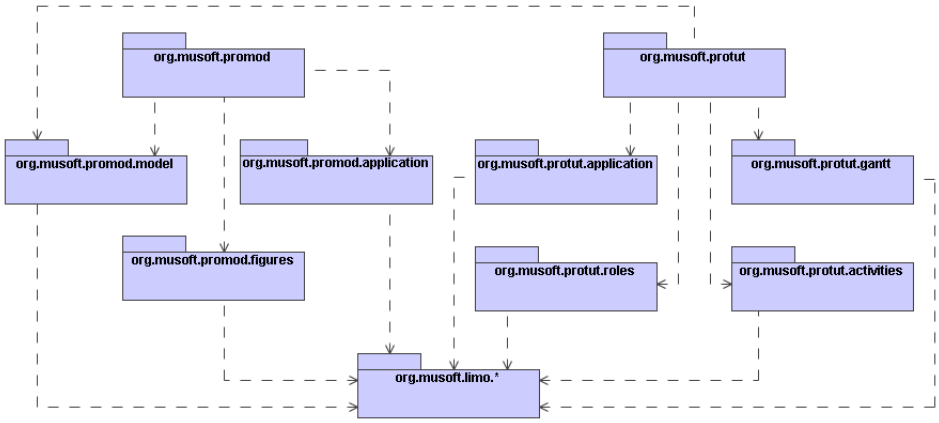



Abbildung 15.4: Paketdiagramm von ProMod und ProTut

ProMod und stellt darauf aufbauend eine Anwendung im Paket `application` bereit. Die zusätzliche Funktionalität der neuen Sichten auf das Prozessmodell (bzw. dessen Instanziierung) findet sich in den Paketen `gantt`, `roles` und `activities` wieder. Da das Metamodell beider Werkzeuge identisch ist, können die mit ProMod erstellten Modelle problemlos in ProTut weiterverwendet werden.

## 15.4 Verwandte Arbeiten

Obwohl der Unified Process in der Literatur weitreichende Berücksichtigung findet, existieren nur wenige unterstützende Werkzeuge. Die beiden wesentlichen stammen von IBM und basieren auf dem Rational Unified Process [Kru03], der „kommerziellen“ Variante des Unified Process.

- Das Produkt Rational Unified Process (RUP) [IBM06b, Gor03] ist ein hypertextuelles Methodenhandbuch, welches den Prozess in allen Einzelheiten beschreibt. Berücksichtigung findet jedoch nur der vollständige Prozess (alle Workflows, Rollen etc.), der für kleinere – speziell studentische – Projekte ungeeignet ist [PF01].
- Zum Maßschneidern dient der Rational Method Composer (RMC) [IBM06a], mit dessen Hilfe der vollständige Prozess an die Bedürfnisse einer Organisation oder eines konkreten Projektes angepasst werden kann. Das Ergebnis ist wieder ein Methodenhandbuch wie das zuvor beschriebene.

Die beiden Produkte lassen sich in der Aufgabenverteilung grob mit ProMod (RMC) und ProTut (RUP) vergleichen, sind aber deutlich komplexer. Sowohl RUP als auch mit RMC erstellte Methodenhandbücher dienen ausschließlich als Referenzmaterial und

sind nicht zur Buchführung und Visualisierung des Projektfortschritts geeignet. Sie arbeiten zudem im wesentlichen mit textuellen und tabellarischen Mitteln, was es weniger erfahrenen Nutzern erschwert, einen Überblick über den Prozess zu gewinnen. Auch die Vermengung verschiedener Aspekte des Prozesses in einer Sicht kann nach [Som04, Kap. 4.4] zu einem Problem werden. ProMod und ProTut hingegen verwenden eine durchweg grafische, intuitiv zugängliche Notation und teilen zudem die verschiedenen Belange des Prozessmodells in einzelne Diagramme auf.

# 16 Das Werkzeug MiningMart

Die bisher beschriebenen Werkzeuge entstanden alle mit der Zielsetzung, die Lehre einer ausgewählten graphischen Modellierungssprache geeignet zu unterstützen. Sie wurden zudem unter unmittelbarer Mitwirkung des Autors (im Falle von Dave, PetRA und Sam) bzw. am gleichen Lehrstuhl (im Falle von ProMod und ProTut) entwickelt. Das im Folgenden beschriebene Werkzeug MiningMart [Eul06, ES06, Min06], das ebenfalls das LiMo-Framework verwendet, entstammt inhaltlich wie personell einem anderen Umfeld:

- Es wurde am Lehrstuhl für Künstliche Intelligenz der Universität Dortmund und ohne Mitwirkung des Autors entwickelt, ist mithin ein Beleg für die Zugänglichkeit des Frameworks aus Entwicklersicht.
- Es ist kein Lehrwerkzeug (wenngleich Wissensvermittlung eine Rolle spielt) und erweitert zudem die Funktionalität von LiMo erheblich. Dies zeigt, dass das Framework auch über die avisierte Anwendungsklasse hinaus sinnvoll nutzbar ist.

Das primäre Ziel von MiningMart war die Schaffung einer integrierten Umgebung zur Unterstützung von Data Mining in relationalen Datenbanken [MS04]. Eines der zeitaufwendigsten Probleme beim Data Mining ist die Vorverarbeitung der Daten, die dem eigentlichen Analyseschritt vorangeht. Zur Vorverarbeitung zählt zum Beispiel die Wahl einer geeigneten Stichprobe, das Auffüllen von fehlenden Werten oder das Aggregieren von Merkmalen. Zur Umsetzung der Vorverarbeitung existieren vordefinierte, parametrisierte Operatoren, die auf eine Datenbank angewendet werden können. Der gesamte Prozess der Vorverarbeitung besteht dann aus einer Kette von Anwendungen dieser Operatoren. MiningMart unterstützt sowohl die graphische Modellierung eines solchen Prozesses als auch dessen tatsächliche Durchführung auf einer Datenbank.

Da die Vorverarbeitung zudem eine Tätigkeit ist, die umfangreiches Erfahrungswissen über die jeweilige Anwendungsdomäne erfordert, war ein weiteres Ziel von MiningMart die Publikation erfolgreich angewendeter Modelle („best practice“) im Internet [Eul05]. Letzteres konnte über die Fähigkeit von LiMo realisiert werden, alle Elemente eines Modells individuell mit Hypertext zu annotieren.

## 16.1 Abstrakte und konkrete Syntax

Abbildung 16.1 zeigt das Metamodell der graphischen Sprache von MiningMart. Es setzt sich aus zwei Teilmodellen zusammen, zwischen denen innerhalb der Anwendung

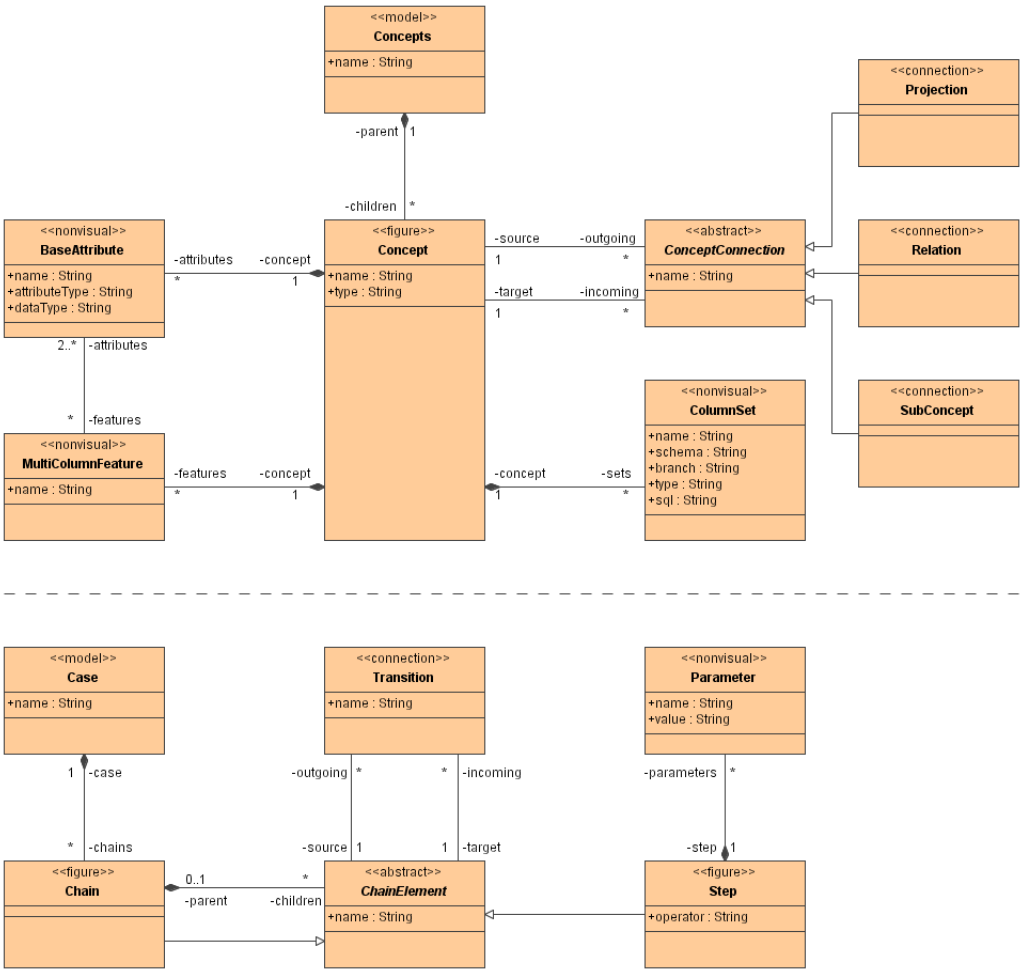



Abbildung 16.1: Metamodell von MiningMart

gewechselt werden kann:

- Ein konzeptuelles Modell (Klasse `Concepts`) beschreibt die Daten der Anwendung. Es lässt sich grob mit einem als Klassendiagramm formulierten Problembereichsmodell vergleichen, wenngleich die Syntax eine andere ist.
- Ein Fallmodell (Klasse `Case`) beschreibt die Vorverarbeitung der Daten in einer Kette von Schritten. Schritte können iteriert angewendet werden. Ketten können strukturiert sein, also untergeordnete Ketten besitzen.

Die Elemente des konzeptuellen Modells (Klasse `Concept`) repräsentieren entweder unmittelbar Tabellen der Datenbank oder sind künstliche Hilfskonstrukte, die nur dem Data Mining dienen. Jedes Konzept setzt sich aus einer Menge von einfachen oder aggregierten Attributen (Klassen `BaseAttribute` und `MultiColumnFeature`) zusammen oder verweist

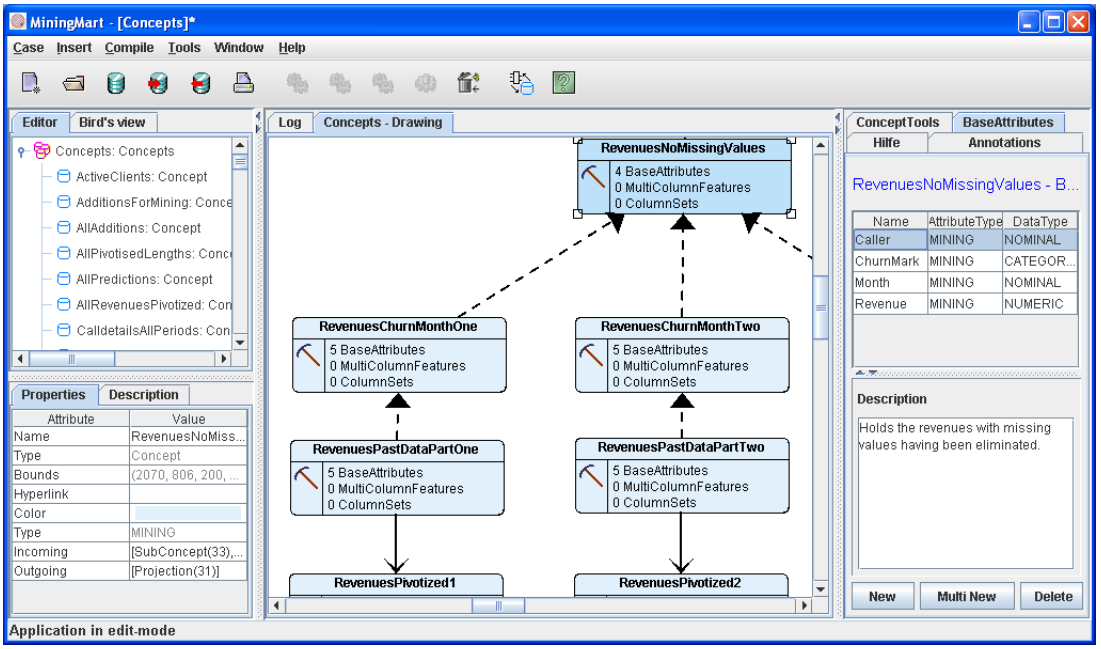



Abbildung 16.2: Bearbeiten eines konzeptuellen Modells mit MiningMart

auf eine Menge von Spalten (Klasse `ColumnSet`). Konzepte können auf verschiedene Weise in Beziehung gesetzt werden (Klassen `Projection`, `Relation` und `SubConcept`).

Das Fallmodell setzt sich aus Ketten zusammen (Klasse `Chain`), deren Elemente (abstrahiert durch die Klasse `ChainElement`) mittels Transitionen (Klasse `Transition`) verbunden werden. Elemente von Ketten sind entweder Schritte, in denen ein Operator angewendet wird (Klasse `Step`), oder untergeordnete Ketten. Jeder Schritt besitzt zudem eine Menge von Parametern, die durch die Art des Operators bestimmt wird.

Die konkrete Syntax von MiningMart verwendet drei Grundfiguren und eine Reihe von Verbindungen. Ein Konzept wird über ein Rechteck dargestellt, das neben dem Namens des Konzepts dessen Typ (in Form einer Grafik) und eine Übersicht der Attribute (jeweilige Anzahlen) enthält. Eine Kette sowie jeder der Operatoren wird über ein Rechteck mit einer entsprechenden Grafik repräsentiert. Instanzen der mit dem Stereotyp `«nonvisual»` gekennzeichneten Klassen sind zwar Teil des Modells, werden aber nicht graphisch im (LiMo-) Editor repräsentiert. Die MiningMart-Anwendung stellt stattdessen proprietäre Funktionalität bereit, um diese Teile des Modells bearbeiten zu können.

## 16.2 Benutzungsschnittstelle

Abbildung 16.2 zeigt ein Bildschirmfoto der Bearbeitung eines konzeptuellen Modells mit MiningMart. Das Werkzeug besitzt die gleiche, durch das LiMo-Framework vorgegebe-

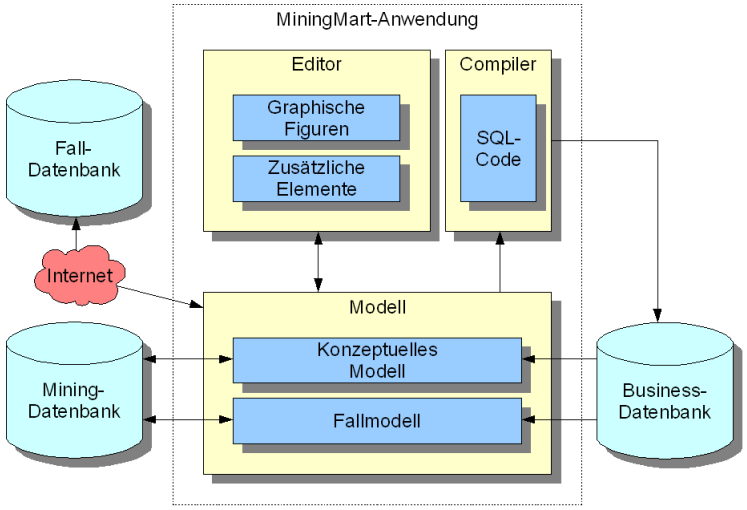



Abbildung 16.3: Architektur von MiningMart

ne Benutzungsschnittstelle wie die bisher beschriebenen Anwendungen, fügt jedoch eine Reihe von eigenen Elementen hinzu:

- Es kann zwischen konzeptuellem Modell und Fallmodell gewechselt werden. Da sich die Benutzungsschnittstelle dynamisch an das (Meta-) Modell anpasst (siehe Abschnitt 7), ist dieser Austausch ohne Probleme möglich.
- Im rechten Abschnitt des Fensters werden bei Bedarf zusätzliche (meist tabellenorientierte) Editoren eingeblendet. Diese dienen zur Bearbeitung der Modellelemente, die im Metamodell als `«nonvisual»` gekennzeichnet waren.
- Das Modell kann vollständig oder in Teilen in SQL-Anweisungen übersetzt werden, die anschließend auf der Datenbank ausführbar sind. Die modellierte Vorverarbeitung kann also auch tatsächlich durchgeführt werden.

## 16.3 Architektur

Abbildung 16.3 skizziert die Architektur von MiningMart. Diese weicht an einigen Stellen signifikant von der Architektur bisher vorgestellter Anwendungen ab. Neben den bereits erwähnten Unterschieden (zwei Modelle, zusätzliche Elemente des Editors) ist dies im Wesentlichen die Anbindung von MiningMart an verschiedene Datenbanken:

- Modelle werden in einer speziellen Datenbank abgelegt („Mining-Datenbank“), statt den XML-Persistenzmechanismus von LiMo (siehe Abschnitt 6.6) zu verwenden.
- Das konzeptuelle Modell kann teilweise aus einer zu analysierenden Datenbank („Business-Datenbank“) generiert werden.

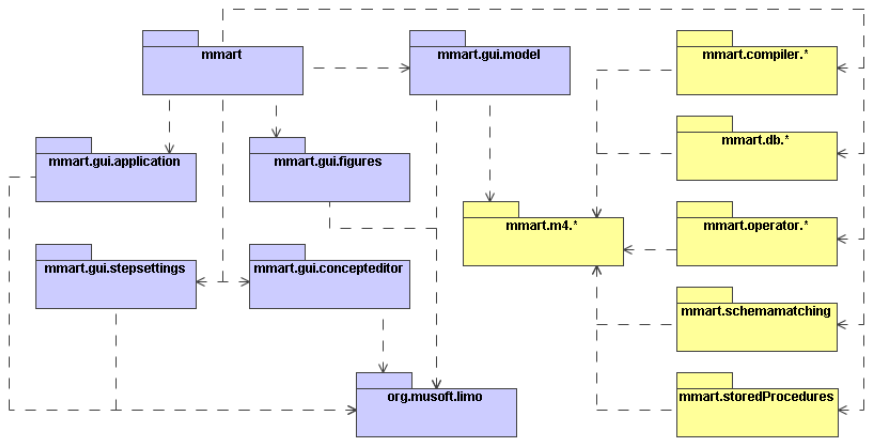



Abbildung 16.4: Paketdiagramm von MiningMart

- Ein Compiler übersetzt Modelle in SQL-Code, auf dessen Basis die Vorverarbeitung der Business-Datenbank anschließend tatsächlich durchgeführt werden kann.
- Eine Import-/Export-Schnittstelle ermöglicht den Austausch annotierter Modelle mit einer zentralen Falldatenbank in einem definierten Format.

Entsprechend der umfangreichen, über die reine Modellierung hinausreichenden Funktionalität macht der auf LiMo basierende Teil nur einen geringen Teil der Implementierung von MiningMart aus. Der weitaus größte Teil entfällt auf den Compiler, die zahlreichen Operatoren sowie die Datenbankschnittstelle. Abbildung 16.4 gibt die wichtigsten Pakete und Abhängigkeiten in einem entsprechenden Diagramm wieder.

## 16.4 Verwandte Arbeiten

Neben MiningMart existieren bereits andere, ebenfalls graphische Werkzeuge zur Unterstützung der Vorverarbeitung im Rahmen von Data Mining. Als kommerzielle Beispiele seien hier der IBM DB2 Intelligent Miner [IBM06c] und SAS Enterprise Miner [SAS06] genannt. Auch im akademischen Umfeld existieren entsprechende Lösungsansätze, etwa [RRT06]. Da MiningMart jedoch kein unmittelbarer Bestandteil der vorliegenden Arbeit ist, soll im Folgenden weniger der Vergleich mit diesen Werkzeugen eine Rolle spielen (dies leistet zum Beispiel [Eul06]) als vielmehr die Aspekte, die sich aus der Nutzung des LiMo-Frameworks ergeben haben:

- Existierende Lösungen berücksichtigen üblicherweise nur entweder die Datensicht oder die Prozesssicht, so dass mit mehreren Werkzeugen gearbeitet werden muss. MiningMart integriert beide Sichten (und deren Teilmodelle) in einem Werkzeug. Das LiMo-Framework unterstützt diese Integration durch seine dynamisch an ein (Meta-) Modell angepasste Benutzungsschnittstelle.

- Existierende Lösungen bieten keine Infrastruktur zum Austausch von Fallbeispielen über das Internet. Dieser Ansatz ist neu und spezifisch für MiningMart. Dementsprechend spielt die Dokumentation von Fällen, die in MiningMart durch die entsprechende LiMo-Infrastruktur übernommen wird, dort keine Rolle.

MiningMart stellt in der hier dokumentierten Fassung bereits die zweite Implementierung des Werkzeugs dar. Die erste entstand ebenfalls auf der Basis von Java, aber ohne Nutzung des LiMo-Frameworks. Das Ergebnis war ein Modellierungswerkzeug, dessen Bedienbarkeit und Wartbarkeit für das Projekt nicht zufriedenstellend war, so dass nach einer Alternative gesucht wurde. Die Neuentwicklung auf der Basis von LiMo profitierte dann stark von den Vorteilen des Frameworks: Die Entwicklung des Werkzeugs war effizient und ohne Unterstützung des Autors möglich. Das neue Werkzeug besitzt zudem die gewünschte Bedienbarkeit. MiningMart kann also als Beleg für die Gebrauchstauglichkeit und Zugänglichkeit des LiMo-Frameworks betrachtet werden, wie auch für dessen Anwendbarkeit außerhalb des reinen Lehrkontextes.

# 17 Zwischenresümee

In Teil III der Arbeit wurden konkrete Modellierungswerkzeuge vorgestellt, die auf der Basis des LiMo-Frameworks entstanden sind. Im Unterschied zu dem zwar funktionalen, aber nur zu Demonstrationszwecken dienenden Werkzeug für Anwendungsfalldiagramme, das in Teil II beschrieben wurde, sind diese Werkzeuge konkret in dem Sinne, dass sie für echte Lehr-/Lernszenarien konzipiert wurden und innerhalb der Softwaretechnik-Ausbildung an der Universität Dortmund und an anderen deutschen Hochschulen eingesetzt werden.

Die Werkzeuge decken dabei ein breites Spektrum an graphischen Modellierungssprachen ab. Jedes Werkzeug fügt der Grundfunktionalität von LiMo – abhängig von der jeweiligen Sprache – didaktisch motivierte Funktionalität hinzu, die das Lehren und Lernen unterstützt:

- Das Werkzeug Dave verwendet eine interaktive Simulation zur Vermittlung der Semantik von Zustandsdiagrammen. Die anschauliche Visualisierung der Simulation überbrückt dabei die Kluft zwischen abstraktem Modell und konkretem Problem und trägt so unter anderen zur Motivation der Studierenden bei.
- Das Werkzeug PetRA überträgt die Ideen der interaktiven Simulation und der anschaulichen Visualisierung erfolgreich auf Petrinetze. Gleichzeitig fügt es Funktionalität hinzu, die dem Umstand Rechnung trägt, dass bei Petrinetzen aufgrund der formalen Semantik oft die Analyse von Systemen im Vordergrund steht.
- Das Werkzeug Sam kann als Erweiterung von Dave betrachtet werden, da es ebenfalls Zustandsdiagramme zur Spezifikation des Systemverhaltens nutzt. Zusätzlich berücksichtigt Sam die Modellierung einer komponentenbasierten Systemstruktur. Das modellierte System ist auch hier wieder simulierbar.
- Das Werkzeug ProMod dient zur Modellierung und zum Maßschneidern eines Software-Entwicklungsprozesss mit den Mitteln des Unified Process. Das zusätzliche Werkzeug ProTut unterstützt anschließend das Projektmanagement auf der Basis des Modells und eignet sich zum Beispiel für Praktika.

Eine Sonderrolle kommt dem Werkzeug MiningMart zu, das kein dediziertes Lehrwerkzeug ist, sondern zum Data Mining eingesetzt wird. Auch MiningMart profitiert von den Vorteilen, die sich aus dem leichtgewichtigen Ansatz des LiMo-Frameworks ergeben.

Alle beschriebenen Werkzeuge besitzen – resultierend aus der Nutzung des LiMo-Frameworks – eine überschaubare und weitgehend identische Benutzungsschnittstelle, was eines

| Produkt | Download | Festplatte | Hauptspeicher |
|---|---|---|---|
| DAVE | 6 MB | 6 MB | 32 MB |
| PETRA | 2 MB | 4 MB | 32 MB |
| SAM | 2 MB | 2 MB | 32 MB |
| PROMOD | 1 MB | 1 MB | 32 MB |
| PROTUT | 1 MB | 1 MB | 32 MB |

Tabelle 17.1: Systemanforderungen der verschiedenen LIMO-Werkzeuge

der erklärten Ziele des Ansatzes war. Alle Werkzeuge liegen, wie Tabelle 17.1 zeigt, in den Systemanforderungen deutlich unter den Werten, die in Abschnitt 3.4 für professionelle Werkzeuge diskutiert wurden. Damit – und mit der Integration der jeweils werkzeugspezifischen, didaktisch motivierten Funktionalität – sind die in Teil I der Arbeit definierten Ziele erfüllt. Offen bleibt jedoch die Frage nach der Akzeptanz und dem Nutzen der Werkzeuge in der Lehre. Der abschließende Teil IV der Arbeit untersucht dies exemplarisch anhand von zwei evaluierten Einsätzen des Werkzeugs DAVE an der Universität Dortmund.

# Teil IV

# Evaluation des Werkzeugs DAVE

# 18 Überblick

Das LiMo-Framework und die darauf aufbauenden Werkzeuge stellen in mehrerlei Hinsicht Neuentwicklungen dar, die als solche einer Evaluation bedürfen. So ist aus softwaretechnischer Perspektive während der Entwicklung jeglicher Software eine Überprüfung notwendig, die sicherstellt, dass diese Software ihre Spezifikation erfüllt und die von den Nutzern erwartete Funktionalität bereitstellt [Som04, S.516]. Eine solche Überprüfung gliedert sich nach der klassischen Terminologie von Boehm [Boe76] in zwei wesentliche Teilaspekte, Verifikation und Validation:

- Verifikation stellt sicher, dass die Software die spezifizierten funktionalen und nichtfunktionalen Anforderungen erfüllt.
- Validation stellt sicher, dass überhaupt die vom Kunden erwartete und benötigte Software realisiert wird.

Die Grenze zwischen beiden Teilaspekten ist oft fließend. Zudem wird bei einer klassischen industriellen Softwareentwicklung die Validation schon weitgehend durch die Mitwirkung des Kunden an der Anforderungsdefinition gewährleistet. Es verbleibt dann während oder nach der Implementierung der Software eine Phase, in der die Verifikation mit Hilfe von Tests – zunächst durch die Entwickler, später gegebenenfalls auch durch den Kunden selbst – operationalisiert wird. Dieses Sicherstellen von Fehlerfreiheit und Stabilität ist natürlich auch für das Framework und die einzelnen Werkzeuge nötig, jedoch nicht das zentrale Anliegen der im Folgenden beschriebenen Evaluation.

Im Rahmen dieser Arbeit spielt vielmehr der Gedanke der Validation eine große Rolle. Das Framework und die darauf basierenden Modellierungswerkzeuge sind zunächst ohne unmittelbare Mitwirkung der Kunden – in diesem Fall der Studierenden – entstanden. Die Anforderungen wurden, wie in Teil I der Arbeit beschrieben, aus existierenden Werkzeugen, insbesondere aus deren beobachteten Defiziten beim Einsatz in einem universitären Ausbildungskontext, abgeleitet. Ob die Funktionalität der leichtgewichtigen Werkzeuge der Zielgruppe angemessen ist und von dieser akzeptiert wird, ob die angestrebte Gebrauchstauglichkeit erreicht wurde und ob die didaktisch motivierte Funktionalität die gewünschten Effekte erzielt, ist durch entsprechende Untersuchungen zu ermitteln.

## 18.1 Methodische Anlage

Im Sinne der empirischen Sozialforschung besteht das Ziel einer Evaluation darin, den Erfolg oder Misserfolg einer Maßnahme bzw. eines Handlungsprogramms mit Hilfe em-

pirischer Informationen zu beurteilen [Kro02, S.70]. Das zentrale methodische Problem ist immer der Nachweis eines Wirkungszusammenhangs zwischen der Maßnahme (z.B. dem Einsatz der Werkzeuge) und einer oder mehreren abhängigen Variablen (z.B. der Zufriedenheit oder einem wie auch immer zu quantifizierenden Wissen der Nutzer). Es sind also entsprechende Zusammenhangshypothesen zu formulieren, die anschließend mit Hilfe einer experimentellen oder wenigstens quasi-experimentellen Versuchsanordnung[1] geprüft werden müssen (siehe [Kro02, S.100] bzw. [Die04, S.35]). Diese Vorgehensweise erweist sich jedoch im Rahmen der vorliegenden Arbeit als problematisch:

- Die Werkzeuge sind keine eigenständigen e-Learning-Materialien, sondern als Unterstützung traditioneller Präsenzlehrformen konzipiert. Damit lassen sie sich nur schlecht von ihrem Einbettungskontext – etwa dem Übungsbetrieb einer Softwaretechnik-Vorlesung – trennen, was die Herstellung von Laborbedingungen und damit die Prüfung des Wirkungszusammenhangs erschwert. Zudem stellt sich die Frage nach geeigneten Probanden, die mit der eigentlichen Zielgruppe (Informatik-Studierende im Grundstudium) vergleichbar sind und über ein ähnliches Vorwissen verfügen.
- Die Reliabilität und die Validität von Experimenten zur Gebrauchstauglichkeit von Software sind gering, wenn diese Experimente unter Laborbedingungen stattfinden [Nie93, S.165ff]. Tests mit echten Nutzern sind also hier vorzuziehen. Zur Sicherstellung der Reliabilität einer solchen Untersuchung werden viele Nutzer benötigt, speziell wenn es um die Aufdeckung von Fehlern geht. Die Validität ist nur gegeben, wenn die Probanden mit möglichst realistischen Aufgabenstellungen betraut werden. Sehr ähnliche Kritik wird an Lernangeboten geäußert, deren didaktische Wirkung zwar im Labor, nicht aber unter realen Bedingungen überprüft wurde [Ker01, S.107].
- Die klassische Zusammenhangshypothese zwischen der Nutzung eines Lernangebotes und einem Wissenzuwachs der Nutzer gilt im Kontext neuer Medien als unzureichend. Bei Untersuchungen zur Wirksamkeit neuer Medien gerät zunehmend der Lernprozess als solcher in den Fokus [Wie04, S.22]. Auch Kerres [Ker01, S. 112] kritisiert die Diskrepanz zwischen Evaluationsstudien, die vielfach nur eine reine Behaltensleistung prüfen, und aktuellen mediendidaktischen Ansätzen. Er schlägt vor, im Kontext neuer Medien andere Kriterien für den Erfolg eines Lernangebotes heranzuziehen, insbesondere die subjektive Zufriedenheit der Nutzer und den motivationalen Effekt des Angebotes.

All dies legt den realen Einsatz der Werkzeuge innerhalb einer realen Veranstaltung nahe. Allerdings kann dann schwerlich mit einer Kontrollgruppe gearbeitet werden, da dies eine Hälfte der Studierenden – je nach Ausgang des Experiments – bewusst bevorzugen oder benachteiligen würde. Dafür lässt sich in diesem Rahmen leicht eine hohe Nutzerzahl

[1] Quasi-experimentelle Versuchsanordnungen orientieren sich an der Vorgehens- und Argumentationslogik des Experiments, basieren aber auf einer weniger strikten Versuchsanordnung. So kann zum Beispiel auf die Verwendung einer Kontrollgruppe verzichtet werden [Kro02, S.98].

erreichen, so dass den von Nielsen geäußerten Problemen von Validität und Reliabilität begegnet werden kann. Die von Kerres angeregten Evaluationskriterien lassen sich gut im Rahmen einer deskriptiven Studie erheben, so dass diese bei der angestrebten Nutzerzahl angemessener scheint als andere Formen der Befragung, etwa Interviews.

Aus pragmatischen Gründen wurde zudem entschieden, die Evaluation formativ anzulegen, sie also bereits zu einem möglichst frühen Zeitpunkt durchzuführen und die Ergebnisse zur weiteren Entwicklung der Werkzeuge zu nutzen (siehe [Kro02, S.102] und [Nie93, S.170]). Dies war auch im Sinne des BMBF-Projekts MuSofT, in dessen Rahmen die Entwicklung ursprünglich angestoßen wurde.

## 18.2 Untersuchungsgegenstand

Neben der Festlegung der Methodik der Evaluation ist eine weitere wichtige Vorbedingung die Wahl eines geeigneten Untersuchungsgegenstandes. Das eigentliche Framework entzieht sich als nicht selbständig lauffähige Software einer solchen Untersuchung. Eine Einbeziehung sämtlicher Werkzeuge schien aus Aufwandsgründen nicht praktikabel. Die Wahl des Untersuchungsgegenstandes fiel aus einer Reihe von Gründen auf das Werkzeug Dave (siehe Kapitel 12):

- Dave war der erste Vertreter der LiMo-Familie, der eine Reife erlangt hatte, die einen Einsatz mit echten Nutzern erlaubte. Damit bot sich Dave insbesondere für die gewünschte frühzeitige, formative Evaluation an.
- Die von Dave unterstützten Zustandsdiagramme sind etablierter Inhalt verschiedener Veranstaltungen, die regelmäßig am Fachbereich Informatik der Universität Dortmund angeboten werden. Die Notationen der anderen Werkzeuge haben teilweise etwas spezielleren Charakter oder sind – wie im Fall von SAM – sogar relativ neu und noch nicht fest in der Lehre verankert.
- Dave kann als repräsentativer Vertreter der LiMo-Familie betrachtet werden. Sämtliche Werkzeuge besitzen in Form des LiMo-Framworks die gleiche technische Basis, so dass insbesondere Aussagen über die Benutzungsschnittstelle, die Gebrauchstauglichkeit und die Fehlerfreiheit unmittelbar für alle Werkzeuge der Familie gelten. Die bei der Evaluation von Dave erzielten Ergebnisse sind damit unmittelbar auf die anderen Werkzeuge übertragbar, sofern sie sich nicht auf den speziellen Lehrinhalt – hier also UML-Zustandsdiagramme – erstrecken.
- Dave integriert mit der Simulationsmaschinerie und der Möglichkeit der anschaulichen Visualisierung durch Alltagsgeräte (siehe Abschnitt 12.2 und 12.3) Funktionalität, die rein didaktisch motiviert ist und über den Umfang traditioneller Modellierungswerkzeuge hinausgeht. Gerade für diese Funktionalität ist es von Interesse, die Akzeptanz und den Effekt auf den Lernprozess der Studierenden zu untersuchen.

Es fanden insgesamt zwei umfangreiche Evaluationen von DAVE statt, die beide im Kern über einen Fragebogen umgesetzt wurden. Bei der ersten Evaluation wurde im Vorfeld eine explorative Pilotstudie bestehend aus teilnehmender Beobachtung und Interviews durchgeführt. Die Ergebnisse dieser Pilotstudie wurden zur Konstruktion des eigentlichen Fragebogens verwendet. Beide Evaluationen werden in den folgenden Kapiteln detailliert beschrieben. Die Beschreibung folgt einem weitgehend einheitlichen Schema, um einen Vergleich der Ergebnisse zu erleichtern.

Da eine Auflage des Programms „Neue Medien in der Bildung“, in dessen Rahmen MuSofT gefördert wurde, die Einbeziehung der Perspektive des Gender Mainstreaming [Sti00, KMGT03] war, wurden die Daten in beiden Umfragen getrennt nach Geschlecht erhoben und ausgewertet. Wo sich signifikante Unterschiede zwischen den Geschlechtern ergeben haben, werden diese entsprechend diskutiert.

# 19 Erste Evaluation (SS 2003)

Im Sommersemester 2003 war die Entwicklung des Frameworks und des Werkzeugs DAVE so weit fortgeschritten, dass ein erster evaluierter Einsatz im Vorlesungsbetrieb möglich war. Die Evaluation wurde vom Hochschuldidaktischen Zentrum (HDZ) der Universität Dortmund unterstützt. Sie war in eine umfangreichere Erhebung eingebettet, innerhalb derer auch andere Fragen untersucht wurden. Dazu zählten zum Beispiel das allgemeine Lernklima im Studiengang Informatik und der Einfluss des Geschlechts auf die Einstellung zu und die Nutzung von E-Learning-Angeboten. Details zu dieser Untersuchung sind in [KMGT+04] nachzulesen und werden außerdem im Rahmen einer weiteren Dissertation [Tig06] aus hochschuldidaktischer Sicht behandelt.

## 19.1 Vorgehen

Aufgrund der thematischen Nähe bot sich für die Evaluation von DAVE die Vorlesung „Software-Technologie“ an, die in Dortmund zu diesem Zeitpunkt letztmalig im Hauptstudium (sechstes Semester) angeboten wurde. Im Rahmen dieser Veranstaltung wurden neben anderen formalen Spezifikations- und Beschreibungssprachen auch UML-Zustandsdiagramme behandelt. Die Vorlesung wurde von etwa 100 Studierenden besucht, 80 davon haben an den Übungen teilgenommen. Das Vorwissen der Studierenden erstreckte sich (laut Studienordnung) auf Datenstrukturen, Algorithmen und objektorientierte Programmierung in Java. Die Studierenden hatten außerdem typischerweise zuvor bereits ein Software-Praktikum absolviert, in dem sie mit einigen Diagrammtypen der UML (jedoch nicht mit Zustandsdiagrammen) sowie einschlägigen Werkzeugen (insbesondere dem in Abschnitt 3.4.1 beschriebenen Together) gearbeitet hatten.

Einsatz und Evaluation von DAVE im Rahmen der Veranstaltung gestalteten sich wie folgt:

- Im Anschluss an die üblichen Vorlesungen zu Zustandsdiagrammen fand eine etwa einstündige Einführung in die Verwendung des Werkzeugs statt. Diese wurde durch den Autor gehalten und beinhaltete eine kurze Folienpräsentation und eine längere Live-Demonstration von DAVE. Die Veranstaltung wurde von mehreren Mitarbeitern des HDZ beobachtet.

- Zur Bearbeitung der vorlesungsbegleitenden Übungszettel zu Zustandsdiagrammen haben die Studierenden über einen Zeitraum von zwei Wochen das Werkzeug genutzt. In jeder Woche war ein Übungszettel zu bearbeiten. Insgesamt kamen dabei alle vier Aufgaben aus Anhang B zum Einsatz, insbesondere die Aufgaben zu Waschmaschine und Kaffeemaschine.
- Eine Gruppe von Studierenden wurde bei ihrem Erstkontakt mit dem Werkzeug durch das HDZ beobachtet. Kommentare der Studierenden wurden protokolliert, und die Studierenden wurden abschließend befragt. Das Vorgehen entsprach in etwa dem in [Nie93] beschriebenen „thinking aloud".
- Bei der Besprechung des zweiten Aufgabenzettels in den Übungsgruppen wurde ein umfangreicher, mit Hilfe des HDZ erarbeiteter Fragebogen an die Studierenden ausgehändigt. Mit der Ausnahme einer einzigen Studentin, die aufgrund einer starken Sehbehinderung den Fragebogen nicht bearbeiten konnte, wurde dieser von allen Studierenden ausgefüllt. Damit lag die Rücklaufquote der Befragung mit 79 von 80 Fragebögen bei nahezu 100%.
- Schließlich fanden noch Interviews mit den beiden wissenschaftlichen Mitarbeitern des Fachbereichs Informatik statt, die für die Betreuung der Übungsgruppen verantwortlich waren.

Die Interviews sowohl mit den Studierenden als auch mit den beiden wissenschaftlichen Mitarbeitern hatten eher qualitativen Charakter. Das Interview mit der Studierendengruppe bildete dabei im wesentlichen die bereits erwähnte Pilotstudie, deren Ergebnisse zur Konstruktion des eigentlichen Fragebogens verwendet wurden. Innerhalb des Fragebogens wurden dann weitgehend quantitative Daten erhoben.

## 19.2 Aufbau des Fragebogens und getroffene Annahmen

Wie zuvor erwähnt, ging die Zielsetzung der Umfrage über die reine Evaluation von DAVE hinaus. Insgesamt enthielt der Fragebogen 64 Fragen, von denen viele jeweils wieder in Teilfragen untergliedert waren[1]. Für DAVE von Interesse waren die produktbezogenen Fragen 6 bis 8, 10 bis 15 und 17 bis 32, weiterhin die Fragen 39, 41 und 42 zur technischen Ausstattung sowie die personenbezogenen Fragen 57, 58, 59 und 61. Die für die weitere Diskussion relevanten Teile des Fragebogens finden sich in Anhang C der Arbeit. Zur Bearbeitung des Fragebogens in den Übungsgruppen wurden etwa 20-25 Minuten veranschlagt.

Der überwiegende Teil des Fragebogens war quantitativer Natur, ein geringer Teil hatte qualitativen Charakter oder erlaubte die Angabe von frei formuliertem Text. Den Studierenden wurde fast immer eine zweiwertige Nominalskala zur Bewertung vorgegeben, d.h. Antworten konnten als wahr oder falsch markiert werden. Bei Fragen, mit denen

[1] In der Sprechweise der empirischen Sozialforschung wären dies 64 einzelne Items bzw. Batterien von Items.

eine Einstellung der Studierenden detaillierter erfragt werden sollte, kam eine fünfstufige Likert-Skala zum Einsatz. Eine einzelne größere Fragengruppe zur Nutzungshäufigkeit der verschiedenen Elemente von DAVE verwendete eine vierelementige Ordinalskala. Die Auswahl der Skalen geschah auf der Basis von [Kro02]. Insgesamt folgte der Fragebogen der Maßgabe, möglichst wenige verschiedene Skalierungen zu verwenden (vergleiche [Nie93, S.213]).

Im einzelnen sollten mit dem Fragebogen die folgenden vier Annahmen überprüft werden:

- **Annahme 1 (Funktionalität und Stabilität):** Die technische Basis des Werkzeugs bestehend aus Java und dem LiMo-Framework ist tragfähig. Das entstandene Werkzeug DAVE besitzt einen ausreichenden Funktionsumfang und arbeitet hinreichend stabil, um im Übungsbetrieb von einer größeren Anzahl von Studierenden verwendet zu werden.
- **Annahme 2 (Gebrauchstauglichkeit):** Das Konzept eines leichtgewichtigen, auf einen eng begrenzten Einsatzkontext zugeschnittenen Werkzeugs ist tragfähig. Die bewusst schlicht gehaltene Benutzungsschnittstelle wird als intuitiv zugänglich und konsistent empfunden. Die Bedienbarkeit des Werkzeugs ist so gut, dass dessen Einsatz im Übungsbetrieb ohne längere Einarbeitungszeit möglich ist.
- **Annahme 3 (Wirkung der Simulation):** Die Möglichkeit, ein modelliertes Zustandsdiagramm zu simulieren und es damit zu testen, wird von den Studierenden aktiv genutzt. Die Simulation hilft den Studierenden bei der Bearbeitung der Aufgaben. Sie trägt außerdem zu einem vertieften Verständnis der Laufzeitsemantik von Zustandsdiagrammen bei.
- **Annahme 4 (Wirkung der Visualisierung):** Die Möglichkeit, die Simulation eines Zustandsdiagramms mit einer Visualisierung des modellierten Systems zu verbinden, wird von den Studierenden aktiv genutzt. Die Visualisierung trägt zu einer höheren Anschaulichkeit bei. Sie erhöht außerdem die Motivation, sich mit der Bearbeitung der Aufgaben zu beschäftigen.

Wie bereits erwähnt, wurde die Umfrage getrennt nach Geschlechtern ausgewertet, um eventuelle Unterschiede – etwa in Erwartungshaltung oder Akzeptanz – aufzudecken. Zu den Geschlechterunterschieden wurde keine explizite Annahme formuliert. Implizit liegt jedoch die Annahme vor, dass keine Unterschiede zwischen den Studentinnen und Studenten der Untersuchungsgruppe bezüglich der Fragen zum Werkzeug – insbesondere der vier oben aufgestellten Annahmen – existieren.

## 19.3 Ergebnisse

Die informalen und formalen Befragungen der Studierenden ergaben ein weitgehend positives Bild des Werkzeugs. Die folgenden Abschnitte geben das Feedback der Studierenden

in verschiedenen Bereichen, die durch den Fragebogen berührt wurden, wieder. Zur Straffung der Diskussion wurden mehrere Vereinfachungen vorgenommen:

- Bei Fragen, welche die o.a. Likert-Skala nutzen, werden im Text die Antworten „stimme eher zu“ und „stimme voll zu“ als Zustimmung interpretiert. Analog werden die Antworten „stimme weniger zu „ und „stimme nicht zu“ als Ablehnung interpretiert.
- Alle Prozentangaben werden ganzzahlig gerundet. Ungültige und fehlende Antworten werden nicht diskutiert. Beides kann zu Summen führen, die nicht 100% entsprechen.

Die kompletten Ergebnisse finden sich in Anhang D.

### 19.3.1 Zusammensetzung der Untersuchungsgruppe

Die Untersuchungsgruppe setzte sich aus 11 weiblichen und 68 männlichen Teilnehmern zusammen. Davon studierten zum Zeitpunkt der Erhebung 75% Kerninformatik und 23% Angewandte Informatik (Ingenieurinformatik) mit verschiedenen Anwendungsfächern. Die restlichen 3% besuchten die Veranstaltung im Rahmen ihres Nebenfachs oder eines Zweitstudiums.

Die Studierenden befanden sich – soweit man die Semesterzahl zugrundelegt – fast ausnahmslos im Hauptstudium (46% im sechsten Fachsemester, 30% im achten Fachsemester, 22% hatten 10 oder mehr Fachsemester, nur 3% Studierende aus dem Grundstudium hatten sich in die Veranstaltung „verirrt“). Dies entsprach den Erwartungen, da die Veranstaltung „Software-Technologie“ eine für das Hauptstudium vorgesehene Stammvorlesung war.

### 19.3.2 Technische Ausstattung

Alle Befragten gaben an, privat einen eigenen Rechner zu besitzen oder Zugriff auf einen Rechner zu haben. Im Durchschnitt lag die Geschwindigkeit dieses Rechners bei etwa 1600 MHz, der Hauptspeicher bei 384 MB und die Festplattenkapazität bei etwa 80 GB. Damit waren die privat verfügbaren Rechner zum Zeitpunkt der Untersuchung technisch auf einem aktuellen Stand und insbesondere in der Lage, Java-Programme wie Dave auszuführen.

Bei der Frage nach dem verwendeten Betriebssystem wurden Microsoft Windows mit 90% und Linux mit 47% am häufigsten genannt. Es folgen Sun Solaris mit 9%, Apple Mac OS mit 4% und verschiedene weitere Betriebssysteme mit ebenfalls 4% (Mehrfachnennungen waren möglich). Die Verteilung belegt die erwartete Experimentierfreude, die Studierende der Informatik bezüglich der von ihnen verwendeten Rechner und Betriebssysteme aufbringen. Sie zeigt auch, dass die Anforderung, das Framework und die Werkzeuge

plattformunabhängig zu realisieren, gerechtfertigt war. Es fällt auf, dass, während die Nutzungshäufigkeit von Windows bei beiden Geschlechtern etwa gleich hoch ist, der Anteil weiterer Betriebssysteme bei den Studentinnen deutlich geringer ausfällt (nur 18% nutzen Linux, während dies bei 52% der Studenten der Fall ist).

Fast alle Studierenden (96%) verfügen zu Hause über einen Zugang zum Internet. Der überwiegende Teil dieser Zugänge ist breitbandig: 47% nutzen DSL, 11% ISDN und 19% ein Modem. Damit ist die Distribution von vorlesungsbegleitenden Werkzeugen wie DAVE über das Internet – zumindestens innerhalb der Informatik – offenbar unproblematisch. Es ist nicht notwendig, größere Mengen von CD-ROMs in der Vorlesung bereitzustellen.

### 19.3.3 Vorerfahrung

Die mit 71% meisten Studierenden haben die vorangehenden Übungsaufgaben der Veranstaltung mit Stift und Zettel bearbeitet. 43% haben dies mit Software-Unterstützung getan (Mehrfachnennungen waren möglich). Differenziert man hier nach Geschlechtern, stellt man fest, dass sich die Software-Nutzer ausschließlich aus der Gruppe der Studenten rekrutieren: 50% der Studenten haben den Rechner zur Lösung vorangehender Aufgaben genutzt, aber keine einzige Studentin.

Dieses ungleiche Nutzungverhältnis kann jedoch nicht auf Unkenntnis der entsprechenden Werkzeuge zurückzuführen sein. Der mit 79% überwiegende Teil hatte bereits Erfahrung mit graphischen Modellierungswerkzeugen. Hier wurde am häufigsten das CASE-Werkzeug Together genannt, das den Studierenden aus der Veranstaltung „Software-Praktikum“ (drittes oder viertes Semester) bekannt sein sollte. Weiterhin genannt wurden Microsoft Visio und Corel Draw, die aber beide keine dedizierten Modellierungswerkzeuge, sondern eher allgemeine, vektororientierte Zeichenprogramme sind.

### 19.3.4 Erwartungshaltung

Dem Einsatz des neuen Werkzeugs DAVE standen die Studierenden nach der einführenden Vorlesung überwiegend positiv gegenüber (62% Zustimmung, 25% Ablehnung, 8% unentschieden). Die meisten sahen den Mehrwert, den das Werkzeug mit sich bringt (72% Zustimmung, 19% Ablehnung, 4% unentschieden). Bei den konkreten Erwartungen an das Werkzeug (siehe Abbildung 19.1) wurden am häufigsten eine mögliche Arbeitserleichterung (62%) und mehr Spaß beim Bearbeiten der Aufgaben (34%) genannt. Etwa ein Drittel der Studierenden erwartete technische Probleme (29%), ein geringerer Teil (14%) erhöhten Arbeitsaufwand (Mehrfachnennungen waren möglich).

Eine Studentin oder ein Student äußerste auf dem Fragebogen, dass der Aufgabenzettel zu DAVE der erste sei, auf den sie oder er sich gefreut habe. Es ist also insgesamt davon auszugehen, dass die Studierenden dem Einsatz neuer Medien als unterstützendes Element in einer Vorlesung oder im Übungsbetrieb prinzipiell offen gegenüberstehen.

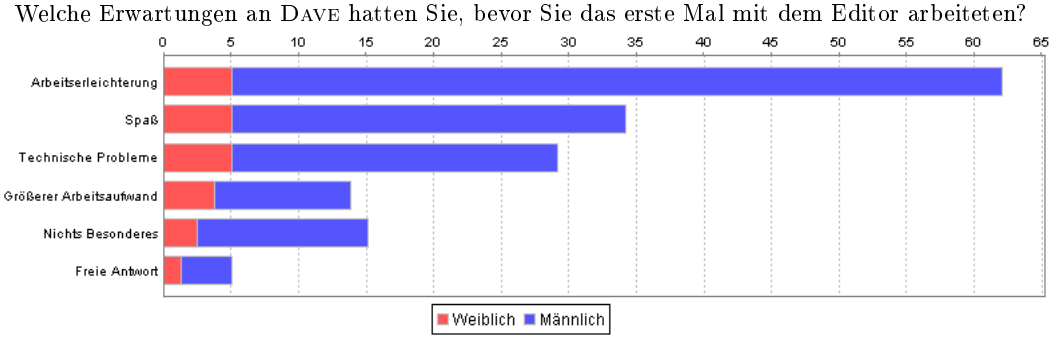



Abbildung 19.1: Histogramm zu Evaluation 1, Frage 10

Bei der Frage nach der Erwartungshaltung gegenüber dem Werkzeug zeigt sich aber an einer Stelle ein deutlicher Unterschied zwischen den Geschlechtern: Der Anteil der Studenten, die sich von DAVE eine Arbeitserleichterung versprechen, liegt mit 66% knapp doppelt so hoch wie der Anteil der Studentinnen, die dies tun. Entsprechend befürchten mit 27% mehr als doppelt so viele Studentinnen wie Studenten einen erhöhten Arbeitsaufwand. Es hat also den Anschein, dass trotz der bei beiden Geschlechtern vorhandenen Offenheit gegenüber dem Werkzeugeinsatz die Studentinnen etwas skeptischer sind und insbesondere Kosten und Nutzen des Einsatzes kritischer abwägen.

### 19.3.5 Installation

Die Installation von DAVE bestand darin, eine ZIP-Datei von der Web-Seite des Lehrstuhls zu laden, die Datei zu entpacken und anschließend das Programm zu starten. Dies stellte für die meisten Studierenden keine größere Hürde dar (86% hatten keine Probleme, 13% hatten lösbare Probleme, 1% schaffte die Installation gar nicht). Die Probleme bestanden fast ausschließlich darin, dass zunächst eine aktuelle Java-Laufzeitumgebung eingerichtet werden mußte (34% mußten Java erstmalig installieren oder aktualisieren, 63% besaßen bereits ein aktuelles Java, 3% machten keine Angabe). Diejenigen Studierenden, die zunächst Java installieren mußten, hatten damit im wesentlichen keine Schwierigkeiten (81% hatten keine Probleme, 19% hatten lösbare Probleme).

### 19.3.6 Verwendung

Die Studierenden haben DAVE zur Bearbeitung von insgesamt vier Aufgaben über einen Zeitraum von zwei Wochen genutzt. Der größte Teil (43%) hat für die Bearbeitung aller Aufgaben weniger als fünf Stunden benötigt, eine etwas kleinere Gruppe (38%) hat fünf bis 10 Stunden als Bearbeitungszeit angegeben. Die Gruppen der Studierenden, die zwischen 11 und 20 Stunden (15%) oder sogar über 20 Stunden (4%) benötigt haben, waren relativ klein.

Dabei fällt auf, dass die Studentinnen deutlich weniger Zeit mit dem Programm verbracht haben als ihre Kommilitonen: 64% haben weniger als fünf Stunden mit DAVE gearbeitet, 27% fünf bis 10 Stunden und 9% 11 bis 20 Stunden. Keine Studentin hat mehr als 20 Stunden mit dem Programm verbracht. Ebenfalls fällt bei der Frage nach den Arbeitsgewohnheiten auf, dass die Studentinnen eher zur Arbeit in Gruppen neigen. Nur eine einzige Studentin (9%) gab an, die Aufgaben immer allein bearbeitet zu haben, während dies für immerhin etwa die Hälfte der Studenten zutrifft (53%).

Bei der Frage nach der Nutzungshäufigkeit der einzelnen Elemente von DAVE (Navigator, Inspektor etc.) ergab sich, dass jede wesentliche Funktionalität zumindestens von einem Teil der Studierenden mehrfach oder regelmäßig genutzt wurde. Dies ist ein positiver Beleg für die Validation des Ansatzes, da die angebotene Funktionalität offenbar Anklang findet. Gleichzeitig ist es nach [Nie93] wichtig für die Verifikation, da die Aufdeckungswahrscheinlichkeit von Fehlern abhängig von den Nutzerzahl ist.

### 19.3.7 Simulation

Ein zentrales Merkmal, das DAVE von anderen Modellierungswerkzeugen unterscheidet, ist die Möglichkeit der Simulation. Diese Funktion wurde von allen Studierenden genutzt (72% ständig, 24% ein- oder mehrmals). Abbildung 19.2 gibt die entsprechenden Ergebnisse wieder. Die Studierenden schätzten das frühzeitige Feedback, das sie durch diese Simulationsfunktion erhielten (90% Zustimmung, 8% Ablehnung, 2% unentschlossen) und sahen die Simulation als Hilfe beim Lösen der Aufgaben (94% Zustimmung, 6% Ablehnung).

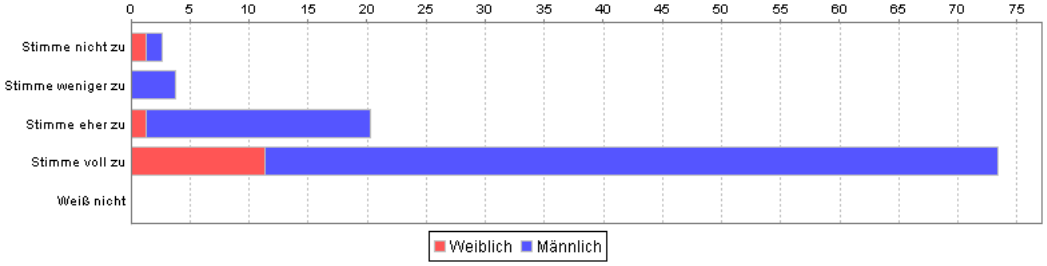
Die Mehrheit der Studierenden hat nach eigener Einschätzung durch die Simulation ein besseres Verständnis für den zugrunde liegenden Formalismus der UML-Zustandsdiagramme gewonnen (81% Zustimmung, 18% Ablehnung, 1% keine Angabe), was ja eines der erklärten didaktischen Ziele war. Kein einziger Studierender möchte auf die Simulationsfunktion zugunsten eines ausschließlichen späteren Feedbacks durch den Übungsgruppenleiter verzichten (99% Ablehnung, 1% unentschlossen).

Ein großer Teil der Studierenden ist der Meinung, dass die Simulationsfunktion dazu verleitet, die Aufgaben nach dem Prinzip „Trial and Error“ zu lösen (52% Zustimmung, 47% Ablehnung, 1% unentschlossen). Dies muß aber nicht unbedingt negativ gesehen werden, da aus jedem Modellierungsfehler, der im Rahmen einer Simulation aufgedeckt wird, gelernt werden kann.

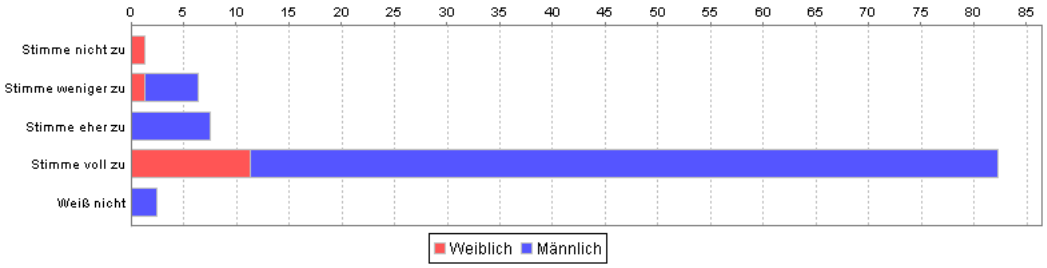
### 19.3.8 Visualisierung

Von den insgesamt vier Aufgaben waren die zwei, die durch multimediale Darstellungen echter Geräte visualisiert und unterstützt wurden, am beliebtesten. Die Möglichkeit

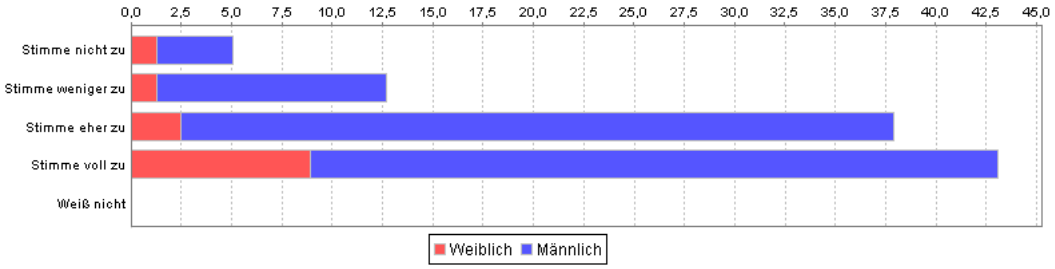
Die Fähigkeit, meine Modelle zu simulieren, hat mich beim Lösen der Aufgaben unterstützt.




Ich fand es gut, ein unmittelbares Feedback zu meinem Modell zu bekommen.




Durch die Simulation habe ich ein besseres Verständnis der Statecharts gewonnen.




Ich halte die Simulation für überflüssig, weil ich von meinem Übungsgruppenleiter ein Feedback bekomme.

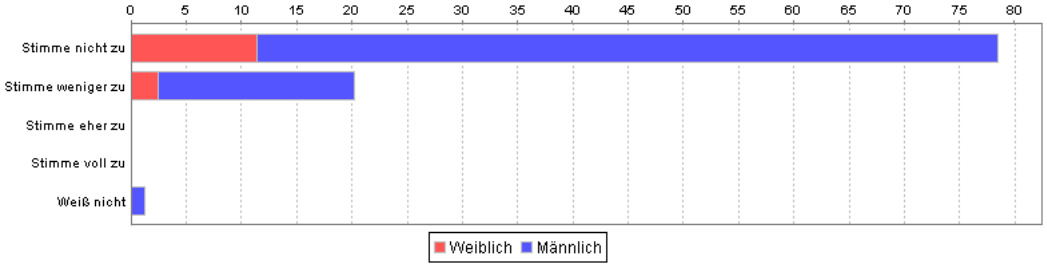



Abbildung 19.2: Histogramme zu Evaluation 1, Frage 19

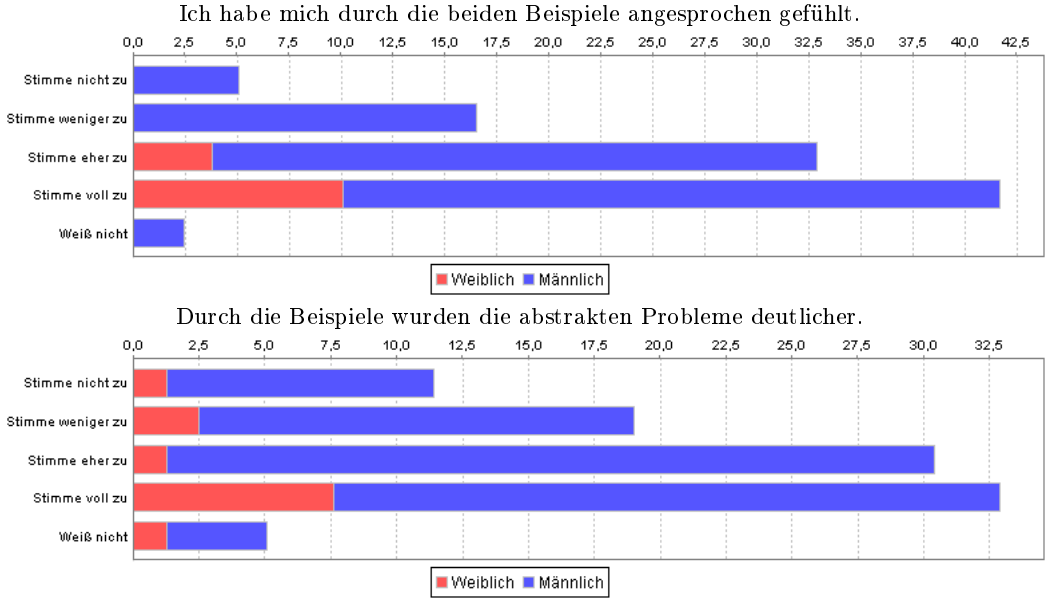



Abbildung 19.3: Histogramme zu Evaluation 1, Frage 20

zur Visualisierung wurde von den meisten Studierenden angenommen (38% bei jeder Aufgabe, 49% ein- oder mehrmals, 10% niemals)[2].

Mit dem für sie neuen Konzept, ein Modell nicht „auf der grünen Wiese“, sondern zur Steuerung eines gedachten Gerätes zu erstellen, hatten die Studierenden keine Schwierigkeiten. Nur wenigen Studierenden war unklar, wie ihr Modell technisch mit der Visualisierung der Waschmaschine bzw. Kaffeemaschine zusammenspielt (13% Zustimmung, 85% Ablehnung, 2% unentschlossen). Es liegt nahe zu vermuten, dass sich die Studierenden, die hier Probleme hatten, mit denen decken, die die Visualisierung niemals genutzt haben. Diese Korrelation wurde aber nicht geprüft.

Abbildung 19.3 gibt zwei Ergebnisse zur Visualisierung detailliert wieder. Die meisten Studierenden fühlten sich durch die anschaulichen Beispiele angesprochen (75% Zustimmung, 22% Ablehnung, 3% unentschlossen). Interessant ist an dieser Stelle der Unterschied zwischen den Geschlechtern: Während die Studenten hier mit 70% Zustimmung etwas unter dem Schnitt liegen, fühlten sich 100% der Frauen von den beiden Beispielen angesprochen.

Die meisten Studierenden sind der Meinung, dass die anschaulichen Beispiele dazu beitrugen, die abstrakten Probleme deutlicher zu machen (63% Zustimmung, 30% Ablehnung, 7% unentschlossen). Dies kann als Bestätigung für die didaktische Idee der Brücke zwischen abstraktem Formalismus und Realität gewertet werden.

[2]Die Frage war ungünstig formuliert. Da nur die Hälfte der Aufgaben überhaupt die multimediale Visualisierung unterstützte, war es eigentlich nicht möglich, sie bei jeder Aufgabe zu nutzen.

Nur wenige Studierende sehen die Beispiele als nicht hilfreich an (14% Zustimmung, 79% Ablehnung, 7% unentschlossen). Trotzdem war nur etwas mehr als die Hälfte der Studierenden der Meinung, dass durch die Beispiele der praktische Nutzen des zugrundeliegenden Formalismus deutlich wurde (51% Zustimmung, 46% Ablehnung, 3% unentschieden). Hier besteht also noch Verbesserungsbedarf, zum Beispiel derart, dass die Einbettung von Zustandsdiagrammen in den gesamten Entwicklungsprozeß deutlicher gemacht wird.

### 19.3.9 Probleme

Da dies der erste größere Einsatz von DAVE war und das Werkzeug zu diesem Zeitpunkt in Teilen sicherlich noch den Charakter eines Prototypen hatte, war es nicht verwunderlich, dass während der Nutzung bei etwas mehr als der Hälfte der Studierenden (53%) technische Probleme auftraten. Tatsächlich war es ja gerade ein Ziel der formativen Evaluation, solche Probleme frühzeitig zu entdecken und zu beheben. In den meisten Fällen haben die Studierenden den Fragebogen genutzt, um diese Probleme detailliert zu schildern.

Ein Fünftel der Studierenden (20%) hat sich im Zusammenhang mit den Problemen Hilfe bei Kommilitonen, Übergruppenleitern oder dem Entwickler des Werkzeugs geholt (eine Kontaktadresse wurde während der Einführungsveranstaltung genannt und war auch im Werkzeug abrufbar). Diejenigen, die dies nicht taten, konnten das Problem entweder allein lösen (10%) oder hielten es nicht für relevant genug (22%).

Obwohl keines der aufgetretenen Probleme schwerwiegend genug war, den Einsatz von DAVE im Übungsbetrieb ernsthaft zu gefährden, wurde nach der ersten Nutzungswoche eine leicht überarbeitete Version von DAVE bereitgestellt und entsprechend angekündigt. In dieser Version waren einige der aufgetretenen Fehler korrigiert, so dass die Probleme eines Teils der Studierenden (11%) bereits behoben waren.

### 19.3.10 Gesamteindruck

Insgesamt wurde das Werkzeug von den Studierenden positiv beurteilt (siehe Abbildung 19.4). Die Oberflächengestaltung wurde als ansprechend empfunden (89% Zustimmung, 10% Ablehnung, 1% unentschlossen). Die Ausführungsgeschwindigkeit war offenbar angenehm (79% Zustimmung, 21% Ablehnung), was bei der Verwendung von Java nicht selbstverständlich ist. Der Ressourcenbedarf scheint also im Rahmen dessen zu liegen, was die Rechner der Studierenden (fast alle nutzten das Werkzeug zu Hause) bereitstellen.

Der Nutzen von DAVE bei der Lösung der Übungsaufgaben wurde von den meisten als gut (47%) bis sehr gut (28%) eingeschätzt (siehe Abbildung 19.5). Ein geringerer Teil sah den Nutzen immer noch als zufriedenstellend (17%) an. Nur wenige Studierende empfanden den Einsatz von DAVE als unbefriedend (9%). Bei den Studentinnen wurde die Note „sehr gut“ etwas häufiger vergeben (36%) als bei den Studenten (27%).

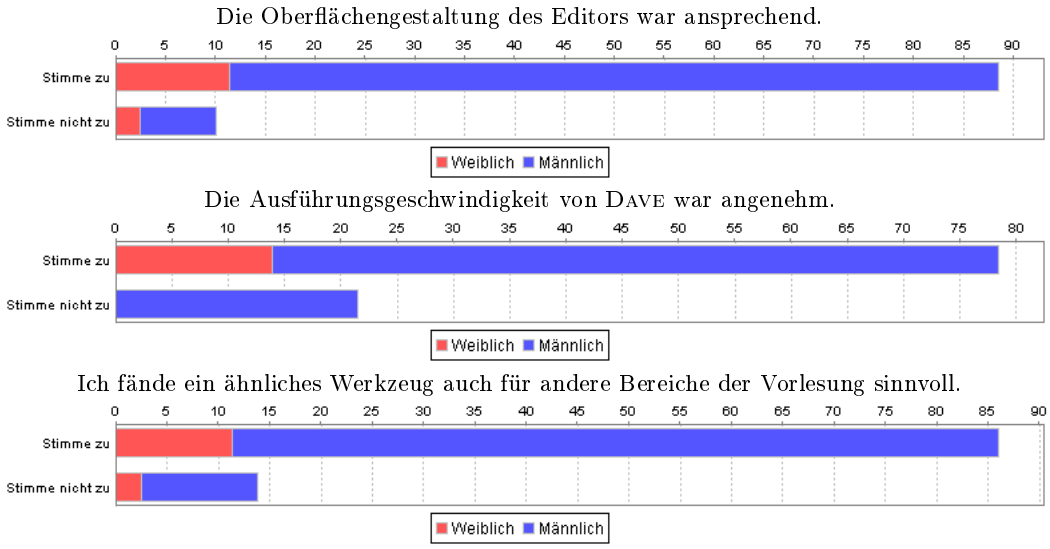



Abbildung 19.4: Histogramme zu Evaluation 1, Frage 26

Die meisten würden das Werkzeug anderen Studierenden weiterempfehlen (75% Zustimmung, 25% Ablehnung). Ein sehr großer Teil der Studierenden wünscht sich ähnliche Werkzeuge für andere Themenbereiche der Vorlesung (86% Zustimmung, 14% Ablehnung).

Kritisch angemerkt wurde von vielen Studierenden, dass sie sich während der Arbeit mit dem Werkzeug beeinträchtigt gefühlt haben (40% fühlten sich beeinträchtigt, 47% nicht, 13% waren unentschlossen oder haben keine Angabe gemacht). Als Grund wurden hier fast ausschließlich die Unausgereiftheit des Programms angegeben, was angesichts der Tatsache, dass dies der erste Einsatz war, nicht verwunderlich ist. Interessanterweise ist ungeachtet der Tatsache, dass technische Probleme (siehe 19.3.9) bei beiden Geschlechtern fast gleich verteilt auftragen, das subjektive Empfinden der Beeinträchtigung stark unterschiedlich: Nur 18% der Studentinnen fühlten sich beeinträchtigt, aber immerhin 44% der Studenten.

### 19.3.11 Vor- und Nachteile

Bei der Frage nach den Vorteilen von Dave (siehe Abbildung 19.6) gegenüber der traditionellen Arbeitsweise mit Stift und Papier wurde mit 90% am häufigsten das unmittelbare Feedback genannt, das durch die Simulation ermöglicht wird. Es folgten mit 68% die Möglichkeit der besseren optischen Gestaltung der Lösungen und mit 56% eine Arbeitserleichterung. Jeweils etwa ein Drittel der Studierenden sah als Vorteil den erhöhten Spaß bei der Arbeit mit Dave (37%), eine höhere Motivation (34%), das Einüben des im professionellen Umfeld üblichen Arbeitens mit Werkzeugen (30%) sowie ein besseres

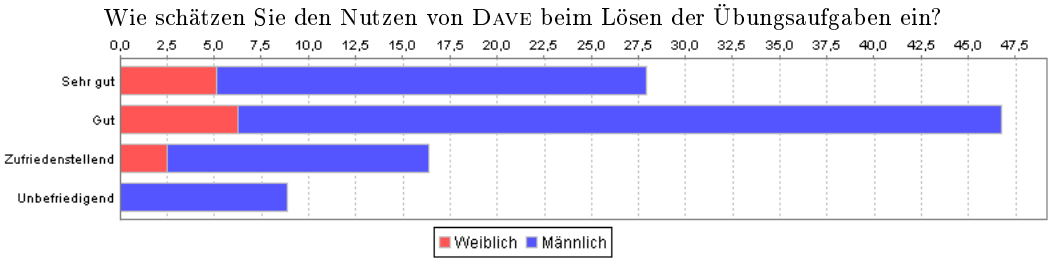



Abbildung 19.5: Histogramm zu Evaluation 1, Frage 27

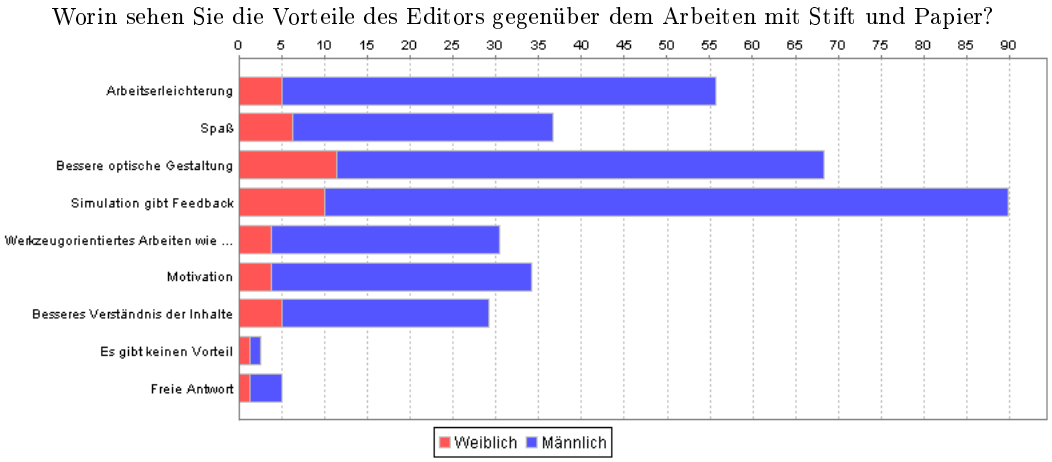



Abbildung 19.6: Histogramm zu Evaluation 1, Frage 28

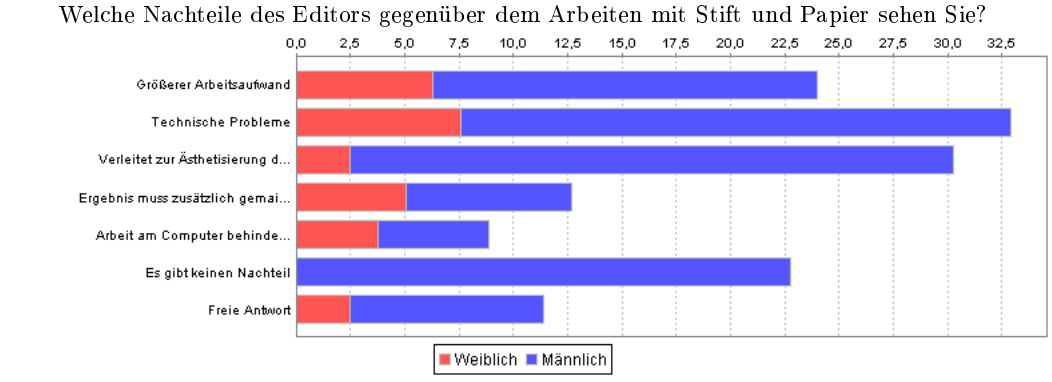



Abbildung 19.7: Histogramm zu Evaluation 1, Frage 29

Verständnis der Inhalte (29%). Ein mit 3% nur sehr geringer Anteil sah keine Vorteile im Einsatz von DAVE.

Bei der entgegengesetzten Frage nach den Nachteilen von DAVE (siehe Abbildung 19.7) kamen die aufgetretenen technischen Probleme mit 33% auf den ersten Rang. Immer noch 30% sahen in der Nutzung von DAVE die Gefahr der Ästhetisierung der Ergebnisse, d.h. den intrinsischen oder extrinsischen Druck, nicht nur korrekte, sondern auch möglichst perfekt aussehende Diagramme abzuliefern. Für weitere 24% bedeutete der Einsatz von DAVE nicht einen verringerten, sondern einen erhöhten Arbeitsaufwand. Immerhin noch 13% waren unzufrieden mit der dualen Abgabe der Lösungen (gedruckte Fassung und elektronische Fassung per E-Mail). Knapp 9% fanden, dass die Arbeit am Computer generell das Arbeiten in Gruppen erschwert. Für 23% der Studierenden hatte die Nutzung von DAVE keine Nachteile.

Betrachtet man die Einschätzung der Vor- und Nachteile getrennt nach Geschlechtern, ergibt sich für die Studentinnen ein etwas anderes Bild. Hier werden als Hauptvorteile die bessere optische Gestaltung (82%), das Feedback der Simulation (73%) und der erhöhte Spaß (46%) gesehen. Speziell die Arbeitserleichterung wird mit nur 36% unterdurchschnittlich häufig genannt, was aber zu der in Abschnitt 19.3.4 diskutierten Erwartungshaltung im Vorfeld passt.

Entsprechend tauchen bei der Rangliste der Nachteile aus Sicht der Studentinnen zunächst wieder die technischen Probleme (55%) auf, gefolgt vom erhöhten Arbeitsaufwand (46%) und der Tatsache, dass das Ergebnis in zwei Varianten abzuliefern war (36%). Auf immerhin 27% kam die Behinderung der Gruppenarbeit, was nicht verwundert angesichts der Tatsache, dass die Studentinnen deutlicher zur Gruppenarbeit tendieren als die Studenten (siehe 19.3.6). Das von den Studenten mit 30% am häufigsten genannte Problem der Überästhetisierung kommt bei den Studentinnen mit nur 18% auf den letzten Rang. Auch dies ist ein interessanter Unterschied in der Priorisierung, der aber zu der geschlechtsspezifischen Einschätzung der Vorteile passt.

### 19.3.12 Verbesserungsvorschläge

Die Studierenden zeigten eine Reihe von Schwachstellen und Fehlern auf, die im Anschluss an den Einsatz korrigiert werden konnten. Außerdem wurden einige Funktionen gewünscht, die in der ersten Fassung von DAVE nicht vorhanden waren. Dazu zählten zum Beispiel eine Online-Hilfe mit einem Einführungsbeispiel, ein Fangraster zum leichteren Ausrichten von Zeichnungselementen, eine bessere Behandlung von Stützpunkten in Transitionen sowie eine „Undo“-Funktion und eine Unterstützung der Zwischenablage. Vieles davon wurde dem Werkzeug ebenfalls in den ersten Wochen nach dem Einsatz hinzugefügt und stand somit bereits für die zweite Evaluation zur Verfügung.

### 19.3.13 Feedback der Betreuer

Das qualitative Feedback der beiden Übungsgruppenbetreuer war ebenfalls positiv. Hervorgehoben wurde die Erleichterung beim Korrigieren der Aufgaben, da hier ebenfalls die Simulationsfunktion des Werkzeugs zum Einsatz kommen konnte.

Außerdem ergab sich in der Diskussion, dass mit dem Werkzeug neue Aufgabentypen möglich sind, die sich bei einer Bearbeitung mit Papier und Bleistift nicht anbieten: Die Aufgaben können, da sie durch die Simulation unterstützt werden, insgesamt komplexer und damit realistischer werden. Wo traditionelle Übungszettel zu Zustandsdiagrammen oft im wesentlichen die Syntax abfragen, erlaubt die Simulation die gewünschten Einblicke in deren (Laufzeit-) Semantik. Die multimedial unterstützten Beispiele tragen zudem zur Motivation der Studierenden bei, indem sie die erwähnte Brücke zwischen abstraktem Vorlesungsstoff und einer praktischen Anwendung schlagen. Dies deckt sich mit den Aussagen der Studierenden im Fragebogen.

# 20 Zweite Evaluation (WS 2003/2004)

Der zweite evaluierte Einsatz von DAVE fand im Wintersemester 2003/2004 statt – also ein Semester nach dem Ersteinsatz. In der Zwischenzeit wurde das Werkzeug auf der Basis des zuvor gewonnenen Feedbacks an zahlreichen Stellen verbessert. Die zweite Erhebung wurde mit einem deutlich überschaubareren Fragebogen, aber dafür einer um ein Vielfaches größeren Untersuchungsgruppe durchgeführt. Auf Interviews und teilnehmende Beobachtungen wurde verzichtet. Verfolgt wurde mit der neuerlichen Evaluation im wesentlichen das Ziel, die Daten, die im Rahmen der ersten Erhebung gewonnen worden waren, zu bestätigen. Dies war insbesondere von Interesse für die geschlechtsspezifischen Aussagen, die aufgrund der geringen absoluten Zahlen von Studentinnen in der ersten Untersuchungsgruppe wenig Signifikanz besaßen. Zudem sollten einige zusätzliche, noch offene oder neu hinzugekommene Fragestellungen untersucht werden.

## 20.1 Vorgehen

Die zweite Evaluation fand in der Vorlesung „Softwaretechnik“ statt, die zu diesem Zeitpunkt in Dortmund im Grundstudium (drittes Semester) angeboten wurde. Im Rahmen dieser Veranstaltung, die Voraussetzung für die im vierten Semester nachfolgende Veranstaltung „Software-Praktikum“ ist, werden alle wesentlichen Teilsprachen der UML sowie allgemeine Modellierungs- und Vorgehensprinzipien vermittelt[1]. UML-Zustandsdiagramme sind damit integraler Bestandteil des Lehrstoffs. Zum Vorwissen der Studierenden gehörten – analog zur ersten Erhebung – wieder Datenstrukturen, Algorithmen und Programmierung in Java. Die Studierenden hatten jedoch noch kein Software-Praktikum absolviert und damit typischerweise zuvor keine Erfahrungen mit graphischen Modellierungssprachen und -werkzeugen gesammelt.

Zum Übungsbetrieb der Vorlesung hatten sich anfänglich 446 Studierende angemeldet, wobei hier – wie in anderen Veranstaltungen auch – im Lauf des Semesters ein gewisser Schwund zu beobachten war. Etwa in der Mitte des Semesters bearbeiteten die Studierenden insgesamt drei Übungszettel zu Zustandsdiagrammen mit Hilfe von DAVE. Dabei kamen die Aufgaben 1, 2 und 3 aus Anhang B zum Einsatz. Bei der Besprechung des

[1] Die Veranstaltung ersetzt in Dortmund zusammen mit der ebenfalls neu eingeführten Hauptstudiumsveranstaltung „Softwarekonstruktion“ die inzwischen nicht mehr angebotene Veranstaltung „Software-Technologie“, in deren Rahmen die erste Evaluation durchgeführt wurde.

dritten Aufgabenzettels in den Übungsgruppen wurde – analog zur ersten Erhebung – ein Fragebogen ausgehändigt. Dieser wurde von allen 324 zu diesem Zeitpunkt in den Übungsgruppen anwesenden Studierenden bearbeitet.

## 20.2 Aufbau des Fragebogens und getroffene Annahmen

Da diese Erhebung ausschließlich der Evaluation von DAVE diente, fiel der Fragebogen weniger umfangreich aus als der erste. Er umfasste nunmehr 24 Fragen (-komplexe), von denen viele unmittelbar aus dem ersten Fragebogen übernommen wurden. In einigen Fällen wurden Fragen auf der Basis der Ergebnisse der ersten Evaluation angepasst oder präzisiert. Schließlich kamen noch einige gänzlich neue Fragen hinzu, die den Schwierigkeitsgrad der einzelnen Themen der Vorlesung zum Inhalt hatten. Der gesamte Fragebogen findet sich in Anhang E der Arbeit. Zur Bearbeitung in den Übungsgruppen wurden etwa 15 Minuten benötigt.

Primäres Ziel der Erhebung war die Bestätigung der Ergebnisse der ersten Evaluation sowie eine Erfolgskontrolle der Änderungen, die im Anschluß an die erste Evaluation durchgeführt worden waren. Damit wurden die Annahmen 1 bis 4 (siehe Abschnitt 19.2) unverändert übernommen. Von Interesse war jedoch zusätzlich, wie sich die Zusammensetzung der neuen Untersuchungsgruppe auswirken würden. Es war anzunehmen, dass bei dieser Gruppe aufgrund der geringeren Vorerfahrung mit Modellierungssprachen- und Werkzeugen sowohl die Bedienbarkeit als auch die von DAVE gelieferte didaktische Unterstützung eine höhere Rolle spielen würden.

Weiterhin sollte mit dem Fragebogen ermittelt werden, welchen Schwierigkeitsgrad die Studierenden den einzelnen innerhalb der Vorlesung vermittelten Themen und Fertigkeiten beimessen. Die Motivation hierfür war im Wesentlichen eine Bedarfsanalyse für weitere Werkzeuge, die von den Studierenden im Rahmen der ersten Evaluation explizit gewünscht worden waren.

## 20.3 Ergebnisse

Die zweite Evaluation bestätigt die Ergebnisse der ersten und übertrifft diese an einigen Stellen sogar. Die folgenden Abschnitte diskutieren die Ergebnisse. Die Gliederung entspricht dabei weitgehend der, die schon für die erste Evaluation verwendet wurde, um den Vergleich beider Erhebungen zu erleichtern. Das komplette Zahlenmaterial der zweiten Erhebung findet sich in Anhang F.

### 20.3.1 Zusammensetzung der Untersuchungsgruppe

Die Untersuchungsgruppe bestand in diesem Fall aus 274 Studenten und 40 Studentinnen. 10 Studierende machten keine Angaben zum Geschlecht. Diese Fragebögen wurden aus der weiteren Auswertung ausgeschlossen, so dass die Gesamtpopulation anschließend aus 314 Studierenden bestand.

Der mit 81% größte Teil studierte zum Zeitpunkt der Erhebung Kerninformatik, gefolgt von 10%, die Angewandte (Ingenieur-) Informatik studierten, und 2%, deren Ziel das Lehramt Informatik war. Insgesamt 5% der Teilnehmer studierten Informatik als Nebenfach zu verschiedenen ingenieurwissenschaftlich-technischen Studiengängen. Die restlichen 2% machten keine Angaben zum Studiengang.

Der weitaus größte Teil der Studierenden befand sich – wie erwartet – im Grundstudium. 66% waren im dritten Fachsemester, das auch gleichzeitig das vorgesehene Semester für die Veranstaltung „Softwaretechnik“ ist. 23% waren im fünften Fachsemester. Der Rest verteilte sich auf andere Semester, wobei das Minimum beim einem und das Maximum bei 11 Fachsemestern lag.

### 20.3.2 Technische Ausstattung

94% der Studierenden gaben an, einen eigenen Rechner zu besitzen. Im Durchschnitt lag die Geschwindigkeit dieses Rechners bei 1700 MHz, der Hauptspeicher bei knapp 512 MB und die Festplattenkapazität bei knapp 100 GB. Die im Vergleich zur ersten Evaluation minimal bessere Ausstattung der Geräte lässt sich durch das halbe Jahr erklären, das zwischen beiden Evaluationen lag.

Auch bei den Betriebssystemen ergab sich eine ähnliche Verteilung wie bei der ersten Erhebung: 90% der Studierenden nutzen Microsoft Windows, 38% nutzten Linux, 1% nutzte MacOS und 1% sonstige Betriebssysteme. Auch die ungleiche Nutzung der Betriebssysteme durch die beiden Geschlechter bestätigt sich: Während immerhin 43% der Studenten Linux alternativ oder parallel zu Windows einsetzen, gilt dies nur für 10% der Studentinnen.

Mit 87% lag auch der Anteil der Studierenden, die zu Hause auf das Internet zugreifen konnten, auf einem ähnlichen Niveau wie bei der ersten Erhebung. 73% nutzten eine breitbandigen Zugang (DSL oder Standleitung), 17% wählten sich per Modem oder ISDN in das Internet ein. Die Distribution des Werkzeugs über das Netz war also auch bei diesem Einsatz unproblematisch. Auffällig ist hier der im Vergleich zur ersten Erhebung deutlich gestiegene Anteil breitbandiger Zugänge.

### 20.3.3 Vorerfahrung

Trotz des frühen Zeitpunkts innerhalb des Studiums und der Tatsache, dass die Studierenden bislang weder Kontakt mit graphischen Modellierungssprachen noch mit entsprechenden Werkzeugen hatten, gab ein relativ hoher Anteil an, einen Teil der Übungsaufgaben mit Hilfe einer Software bearbeitet zu haben (57% hatten eine Software verwendet, 60% Stift und Zettel; Mehrfachnennungen waren möglich). Es stellte sich jedoch heraus, dass die verwendete Software in den meisten Fällen ein Büro- oder Zeichenprogramm war (24% hatten Visio verwendet, nur jeweils 8% ArgoUML und Together; der Rest verteilte sich auf zahlreiche andere Programme).

Die Geschlechterdifferenzen der ersten Untersuchung waren auch hier sichtbar, jedoch nicht in der starken Form. Den 59% der Studenten, die zuvor mit Stift und Zettel gearbeitet haben, standen 68% der Studentinnen gegenüber.

### 20.3.4 Erwartungshaltung

Die Erwartungshaltung der Studierenden wich nicht wesentlich von der ab, die bei der ersten Erhebung festgestellt wurde: Der überwiegende Teil stand der Nutzung des Werkzeugs positiv gegenüber (55% Zustimmung, 29% Ablehnung, 12% unentschieden), wobei die Studentinnen minimal skeptischer waren. Für die meisten Studierenden wurde der Nutzen des Werkzeugs innerhalb der einführenden Vorlesung deutlich (74% Zustimmung, 16% Ablehnung, 7% unentschieden).

### 20.3.5 Installation

Die Installation des Werkzeugs wurde auf die gleiche Weise gehandhabt wie bei dessen erstem Einsatz, d.h. die ZIP-Datei wurde auf dem Web-Server des Lehrstuhls bereitgestellt und musste von den Studierenden heruntergeladen und ausgepackt werden. Die Angaben der Studierenden zu Problemen während der Installation sind nahezu identisch zu denen der ersten Erhebung (84% hatten keine Probleme, 14% hatten lösbare Probleme, 1% schaffte die Installation gar nicht). Auch hier war das häufigste Probleme wieder das Fehlen einer aktuellen Java-Laufzeitumgebung (20% mussten Java erstmalig installieren oder aktualisieren, 76% besaßen bereits ein aktuelles Java). Bei den Studentinnen war eine Aktualisierung von Java in knapp doppelt so vielen Fällen nötig wie bei den Studenten (33% gegenüber 19%). Die Installation von Java stellte im wesentlichen keine Hürde dar (75% hatten keine Probleme, 21% hatten lösbare Probleme, 1% konnte Java nicht installieren).

Man würde bei Studierenden der Informatik vermuten, dass sie die Installation einer Software nicht vor unlösbare Probleme stellt, und beide Erhebungen bestätigen dies. Um jedoch den Vorgang für Studierende mit geringerem technischen Vorwissen (etwa außerhalb der Informatik) zu erleichtern, könnte es sinnvoll sein, künftige Versionen entweder

mit einem automatischen Installationsprogramm oder über den WebStart-Mechanismus von Java auszuliefern.

### 20.3.6 Verwendung

Die Studierenden haben insgesamt drei Übungszettel über einen Zeitraum von drei Wochen mit DAVE bearbeitet. Dabei enthielten der erste und der dritte Übungszettel nur jeweils eine Aufgabe zu Zustandsdiagrammen, der mittlere zwei. Vom Umfang der Aufgaben war der Einsatz von DAVE also vergleichbar mit dem der ersten Erhebung. Auch die Angaben der Studierenden zum zeitlichen Umfang der Nutzung von DAVE entsprechen ziemlich exakt denen der ersten Erhebung: der größte Teil liegt im Bereich von bis zu fünf Stunden (41%), eine nur unwesentlich kleinere Gruppe hat fünf bis 10 Stunden benötigt (40%). Nur wenige Studierende haben mehr als 10 Stunden auf die Bearbeitung der Aufgaben verwendet (18%).

Die geschlechtsspezifischen Nutzungszeiten, die sich in der ersten Erhebung andeuteten, haben sich hier nicht bestätigt. Die Zahlen liegen für männliche wie weibliche Teilnehmer sehr nah beieinander.

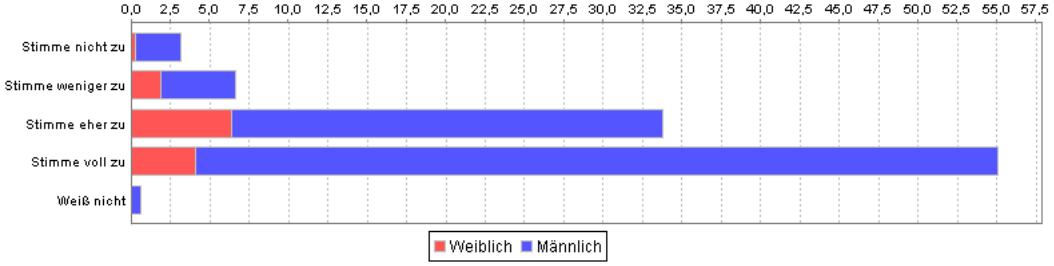
Auch bei diesem Einsatz von DAVE war sichergestellt, dass jede Kernfunktionalität von DAVE tatächlich genutzt wurde. Mit Ausnahme der Homepage des Programms, die keine Funktionalität im eigentlichen Sinne darstellt, gab für jeden Teil von DAVE mindestens ein Drittel der Studierenden an, diesen mehrmals oder ständig genutzt zu haben. Damit sind im Sinne von [Nie93] zuverlässige Aussagen zur Fehlerfreiheit möglich.

### 20.3.7 Simulation

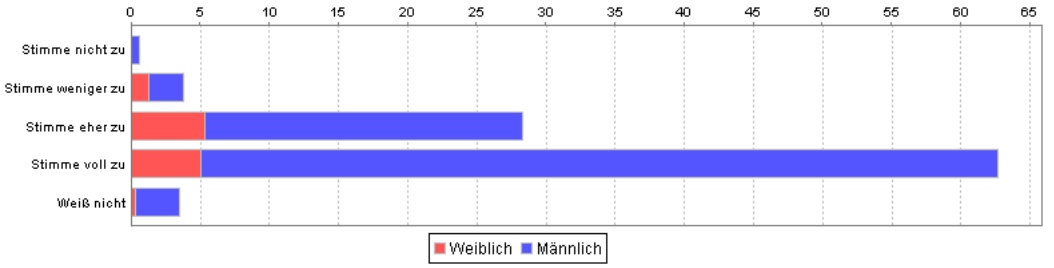
Die Möglichkeit der Simulation von Zustandsdiagrammen wurde von den Studierenden genutzt (46% ständig, 52% ein- oder mehrmals, 1% nie). Auch hier wurde von den Studierenden die Wichtigkeit des frühzeitigen Feedbacks zum eigenen Modell herausgestellt (91% Zustimmung, 4% Ablehnung, 4% unentschieden). Die meisten Studierenden sehen die Simulation als Hilfe beim Lösen der Aufgaben (89% Zustimmung, 10% Ablehnung, 1% unentschieden). Der Simulation wird auch hier ein besseres Verständnis von Zustandsdiagrammen zugesprochen (82% Zustimmung, 16% Ablehnung, 2% unentschieden). Nur wenige Studierende würden auf die Simulation zugunsten eines ausschließlichen Feedbacks durch den Übungsgruppenleiter verzichten wollen (8% Zustimmung, 91% Ablehnung, 1% unentschieden). Abbildung 20.1 stellt diese Ergebnisse graphisch dar.

Die Möglichkeit, die Aufgaben per „Trial and Error“ zu lösen, wird auch hier von relativ vielen Studierenden gesehen (57% Zustimmung, 38% Ablehnung, 5% unentschieden). Jedoch sind nur wenige Studierende der Meinung, dass die Aufgaben durch die Simulationsmöglichkeit zu einfach werden (21% Zustimmung, 76% Ablehnung, 2% unentschieden).

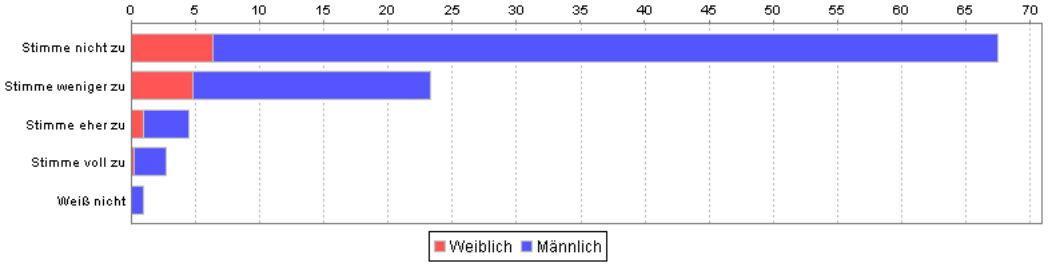
Die Simulation hat mich beim Lösen der Aufgaben unterstützt.




Ich fand es gut, ein unmittelbares Feedback zu meinem Modell zu bekommen.




Ich halte die Simulation für überflüssig, weil ich von meinem Übungsgruppenleiter ein Feedback bekomme.




Durch die Simulation habe ich ein besseres Verständnis der Zustandsdiagramme gewonnen.

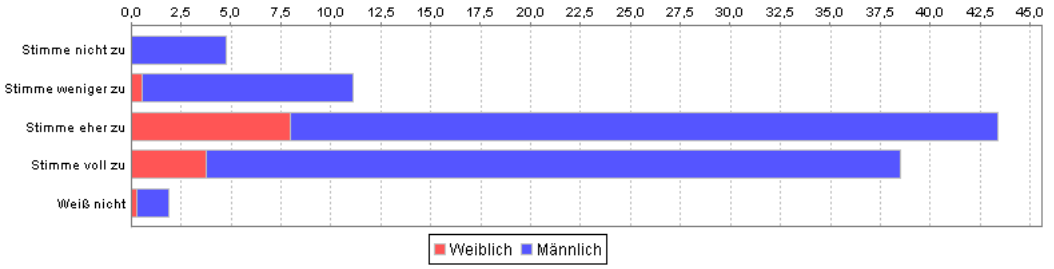



Abbildung 20.1: Histogramme zu Evaluation 2, Frage 14

### 20.3.8 Visualisierung

Im Gegensatz zum Einsatz im vorangehenden Sommersemester konnte beim zweiten Einsatz des Werkzeugs aus organisatorischen Gründen nur eine Aufgabe gestellt werden, die sich die multimediale Veranschaulichung der Simulation zunutze machte. Die Entscheidung fiel auf die Waschmaschine. Die Möglichkeit der Veranschaulichung wurde wieder intensiv genutzt (12% ständig, 76% ein- oder mehrmals, 11% nie). Die meisten Studierenden hatten keine Probleme, das eigene Modell und die Veranschaulichung des gedachten „Gerätes“ zu verbinden (77% Zustimmung, 16% Ablehnung, 5% unentschieden).

Die Einschätzung der Studierenden zur Visualisierung gibt Abbildung 20.2 wieder. Auch bei der zweiten Erhebung fühlten sich die Studierenden überwiegend durch das anschauliche Beispiel angesprochen (62% Zustimmung, 30% Ablehnung, 7% unentschieden) und waren der Meinung, dass es dazu beitrug, die abstrakten Probleme anschaulicher zu machen (71% Zustimmung, 22% Ablehnung, 5% unentschieden). Für einen im Vergleich zur ersten Erhebung größeren Anteil der Studierenden wurde durch das Beispiel der praktische Nutzen von Zustandsdiagrammen deutlich (70% Zustimmung, 25% Ablehnung, 4% unentschieden). Motivation bezog immerhin etwas mehr als die Hälfte der Studierenden aus dem Beispiel (54% Zustimmung, 39% Ablehnung, 6% unentschieden).

Insgesamt war die anschauliche Aufgabe etwas beliebter als die beiden Aufgaben, die ausschließlich mit abstrakten Modellen arbeiteten (52% Zustimmung, 37% Ablehnung, 9% unentschieden).

Bei den Studentinnen fallen die Urteile zur anschaulichen Aufgabe – ähnlich der ersten Erhebung – noch deutlicher positiv aus: Für 85% wurde dadurch das abstrakte Problem anschaulicher (gegenüber 69% der Studenten). 68% wurden durch die Aufgabe motiviert (gegenüber 52% der Studenten). 80% haben durch die Aufgabe ein Gefühl für den Nutzen von Zustandsdiagrammen bekommen (gegenüber 68% der Studenten). 70% hat die Aufgabe zur Waschmaschine besser gefallen als die anderen, eher abstrakten Aufgaben (gegenüber 49% der Studenten).

### 20.3.9 Probleme

Technische Probleme traten in deutlich geringerem Maße auf als beim ersten Einsatz (27% hatten Probleme, 71% hatten keine Probleme), so dass davon auszugehen ist, dass sich die im Anschluss an den ersten Einsatz durchgeführten Verbesserungen positiv ausgewirkt haben. Insbesondere die dem Werkzeug neu hinzugefügte Online-Hilfe wurde insgesamt positiv aufgenommen, wenngleich Vorschläge zur Verbesserung von deren Struktur und Wünsche für zusätzliche Inhalte geäußert wurden.

Gelöst wurden eventuelle Probleme in den meisten Fällen durch Eigeninitiative (30%), Diskussionen unter den Studierenden (16%) oder in der Übungsgruppe (8%) sowie durch einen Blick in die Online-Hilfe (11%) oder auf die Homepage des Programms (7%), auf der sich eine FAQ-Liste befindet. Ein geringer Teil der Probleme (3%) benötigte den


Abbildung 20.2: Histogramme zu Evaluation 2, Frage 15

Support des Entwicklers. Etwa 8% der Probleme konnten nicht (während der Dauer der Erhebung) gelöst werden und hatten Fehlerkorrekturen am Anschluss an den Einsatz zur Folge.

### 20.3.10 Gesamteindruck

Auch die neue Untersuchungsgruppe beurteilte das Werkzeug insgesamt positiv. Die Oberfläche wurde allgemein als ansprechend und gut strukturiert empfunden (83% Zustimmung, 15% Ablehnung, 1% unentschieden). Die Bedienung von Dave fiel dem überwiegenden Teil der Studierenden leicht (65% Zustimmung, 33% Ablehnung, 1% unentschieden), und die Ausführungsgeschwindigkeit wurde als angenehm betrachtet (74% Zustimmung, 24% Ablehnung, 1% unentschieden). Die meisten Studierenden sehen Dave als hilfreich beim Lösen der Aufgaben an (87% Zustimmung, 12% Ablehnung, 1% unentschieden).

Ein im Vergleich zur ersten Untersuchung etwas größerer Anteil würde Dave weiterempfehlen (82% Zustimmung, 13% Ablehnung, 5% unentschieden). Auch hier wurden ähnliche Werkzeuge für andere Teile der Vorlesung explizit gewünscht (83% Zustimmung, 11% Ablehnung, 5% unentschieden).

Die Werte liegen alle im Bereich dessen, was für den ersten Einsatz ermittelt wurde. Auffällige Unterschiede zwischen den Geschlechtern haben sich nicht ergeben.

### 20.3.11 Vor- und Nachteile

Die Fragen zu Vor- und Nachteilen des Werkzeugs fielen aus Platzgründen etwas weniger umfangreich aus als in der ersten Erhebung, verwendeten dafür die etwas feinere Likert-Skala zur Wiedergabe der Einschätzungen.

Den meisten Zuspruch erhielt die bessere optische Gestaltung der Lösungen, die durch das Werkzeug möglich wird (91% Zustimmung, 7% Ablehnung, 1% unentschieden), wobei auch hier ein Teil der Studierenden die Gefahr der Überästhetisierung sah (26% Zustimmung, 63% Ablehnung, 8% unentschieden). Viele Studierende schätzen die professionelle Arbeitsweise mit einem Werkzeug (82% Zustimmung, 9% Ablehnung, 7% unentschieden).

Auch beim zweiten Einsatz von Dave waren Lösungen sowohl elektronisch als auch per E-Mail abzuliefern. Ein großer Teil der Studierenden würde dies gern zugunsten einer ausschließlich elektronischen Abgabe aufgeben (69% Zustimmung, 25% Ablehnung, 5% unentschieden). Hier stehen allerdings die Studentinnen der elektronischen Varianten deutlich skeptischer gegenüber als ihre männlichen Kommilitonen (50% Ablehnung gegenüber 22% bei den Studenten).

Ein ähnliches Bild ergibt sich für die Einschätzung, ob die Nutzung des Werkzeugs die Gruppenarbeit behindert (insgesamt 24% Zustimmung, 66% Ablehnung, 8% unentschieden). Die Studentinnen, bei denen sich die höhere Tendenz zur Gruppenarbeit schon in der ersten Erhebung abzeichnete, stimmen dieser Aussage mit 38% zu, während dies nur 22% der Studenten tun.

### 20.3.12 Schwierigkeitsgrad der Inhalte

Neben den aus der ersten Erhebung bekannten Bereichen enthielt der Fragebogen drei neue Fragenkomplexe. In diesen sollten sich die Studierenden zum empfundenen Schwierigkeitsgrad der einzelnen Teilsprachen der UML sowie zu verschiedenen, im Rahmen der Veranstaltung vermittelten Fertigkeiten äußern. Diese Fragen bildeten die Basis zur Ermittlung eines Bedarfs für eventuelle weitere Werkzeuge. Da sie keine Relevanz für die vier zu prüfenden Annahmen besitzen, sollen die Ergebnisse an dieser Stelle aus Platzgründen nur kurz zusammengefasst werden:

- Klassen- und Objektdiagramme werden von den Studierenden als einfach bis sehr einfach empfunden. Dies ist naheliegend, da sich beide Diagrammtypen sehr gut auf das Vorwissen der Studierenden – objektorientierte Programmierung in Java – abbilden lassen.
- Anwendungsfall- und Zustandsdiagramme besitzen aus Sicht der Studierenden eine mittlere Schwierigkeit. Die Einschätzung der Schwierigkeit von Zustandsdiagrammen ist dabei vermutlich bereits durch die Nutzung von Dave beeinflusst.
- Einzig Sequenzdiagramme werden von nur 31% der Studierenden als einfach oder sehr einfach eingeschätzt – der mit 49% überwiegende Teil hingegen schätzt diesen Diagrammtyp als schwer oder sehr schwer ein; 18% können die Schwierigkeit nicht einschätzen.

Insgesamt werden Diagrammtypen offenbar als schwieriger eingeschätzt, wenn sie weiter von der bisherigen Programmiererfahrung der Studierenden entfernt sind und/oder dynamische Anteile eines Systems beschreiben. Ein großer Teil der Studierenden stimmt speziell der Aussage zu, dass die statische Natur von Diagrammen (z.B. auf Papier) das Verständnis der sich dahinter verbergenden Dynamik erschwert (40% stimmen zu, 49% stimmen nicht zu, 9% sind unsicher). Der Einsatz von (weiteren) Werkzeugen wie Dave bietet sich folglich also generell bei Modellen bzw. Diagrammtypen an, die Dynamik beschreiben und über eine komplexe Laufzeitsemantik verfügen. Konkreter Bedarf für ein solches Werkzeug besteht offenbar bei Sequenzdiagrammen.

# 21 Diskussion der Ergebnisse

Die vorangehenden Kapitel haben zwei evaluierte Einsätze des Werkzeugs DAVE im realen Übungsbetrieb an der Universität Dortmund beschrieben. Diese Evaluationen sind aufgrund der leicht unterschiedlichen Zusammensetzung und Vorerfahrung der beiden Untersuchungsgruppen nicht völlig vergleichbar. Dies war aber auch nicht das Ziel der Erhebungen (insbesondere war keine der beiden Gruppen als Kontrollgruppe vorgesehen). Dennoch sind die Gemeinsamkeiten der Versuchsanordnung so groß, dass die getroffenen Annahmen im Folgenden für beide Erhebungen diskutiert werden sollen.

## 21.1 Zu Annahme 1: Funktionalität und Stabilität

Rechnet man beide Erhebungen zusammen, dann wurde das Werkzeug über einen Zeitraum von mehreren Wochen von insgesamt etwa 400 Studierenden zur Bearbeitung von Übungsaufgaben eingesetzt. Da es in dieser Zeit zu keinerlei grundlegenden Problemen gekommen ist, die eine Bearbeitung der Aufgaben verhindert hätten, ist davon auszugehen, dass die Funktionalität des Werkzeugs – Erstellen, Laden, Speichern und Drucken von Zustandsdiagrammen – für den Einsatz im Übungsbetrieb ausreichend ist. Die Studierenden haben keine wesentliche Funktionalität vermisst, wenn man von einer in der ersten Version fehlenden Online-Hilfe absieht. Verbesserungs- und Erweiterungsvorschläge bewegen sich dementsprechend eher auf der Detailebene.

Die Auswertung der Nutzungshäufigkeiten der verschiedenen Teile von DAVE ergab zudem, dass keine Funktionalität realisiert wurde, die ungenutzt blieb. Die Funktionalität ist also insgesamt angemessen. Traten in der ersten Erhebung noch relativ viele technische Probleme während der Nutzung von DAVE auf, so war diese Zahl in der zweiten Erhebung schon deutlich geringer. Da auch im Anschluss an die zweite Erhebung Fehlerkorrekturen vorgenommen wurden, ist davon auszugehen, dass das Werkzeug inzwischen eine hinreichende Stabilität erreicht hat. Damit kann Annahme 1 aus Sicht des Autors akzeptiert werden.

## 21.2 Zu Annahme 2: Gebrauchstauglichkeit

Die Gebrauchstauglichkeit des Werkzeugs wurde von den Studierenden in beiden Erhebungen bestätigt. Die Benutzungsschnittstelle wurde als ansprechend und wohlstrukturiert empfunden. Die Studierenden der zweiten Erhebung bescheinigten dem Werkzeug

zudem explizit eine leichte Bedienbarkeit, was aufgrund der geringeren Vorerfahrung dieser Gruppe mit Modellierungswerkzeugen von besonderer Relevanz ist. Die Ausführungsgeschwindigkeit der Java-basierten Anwendung wurde als weitgehend angenehm bewertet, wobei beide Erhebungen eine durchaus aktuelle technische Ausstattung der Studierenden mit Rechnern ergeben haben.

Die Studierenden der ersten Erhebung vergaben für den Nutzen von DAVE beim Lösen der Aufgaben gute bis sehr gute Noten. In der zweiten Erhebung erhielt eine ähnliche Aussage eine sehr hohe Zustimmung. Der überwiegende Teil der Studierenden würde DAVE weiterempfehlen und wünscht sich ausdrücklich ähnliche Werkzeuge für andere Bereiche der Vorlesung. Damit kann Annahme 2 aus Sicht des Autors akzeptiert werden.

## 21.3 Zu Annahme 3: Wirkung der Simulation

Die Möglichkeit der interaktiven Simulation des Zustandsdiagramms wurde von den Studierenden intensiv genutzt und insbesondere bei der ersten Erhebung als größter Vorteil von DAVE bewertet. Die Studierenden in beiden Erhebungen schätzten das frühzeitige Feedback, das durch die Simulation geliefert wurde, und sahen die Simulation als Hilfe bei der Lösung der Aufgaben an. Entsprechend mochte fast kein Studierender auf die Simulation zugunsten eines ausschließliche späteren Feedbacks durch die Tutoren in den Übungsgruppen verzichten.

Die Mehrheit der Studierenden hat zudem nach eigener Einschätzung durch die interaktive Simulation ein besseres Verständnis der Laufzeitsemantik von Zustandsdiagrammen gewonnen. Damit kann Annahme 3 aus Sicht des Autors akzeptiert werden.

## 21.4 Zu Annahme 4: Wirkung der Visualisierung

Die Möglichkeit, die interaktive Simulation des Zustandsdiagramms mit einer Visualisierung des modellierten Systems zu verbinden, wurde von den Studierenden beider Erhebungen intensiv genutzt. Das Zusammenspiel zwischen Zustandsdiagramm und visualisiertem System über in der Aufgabenstellung vereinbarte Ein-/Ausgabesignale und Überwachungsbedingungen stellte die Studierenden offenbar nicht vor größere Probleme.

Die Teilnehmer beider Erhebungen fühlten sich insbesondere durch die anschaulichen Aufgaben angesprochen und waren der Meinung, dass diese zur Verdeutlichung ansonsten eher abstrakter Probleme beitrugen. Die dem Bereich der eingebetteten Systeme entnommenen Beispiele waren offenbar auch dazu angetan, den Studierenden einen Eindruck vom praktischen Nutzen der Zustandsdiagramme zu geben. Damit kann Annahme 4 aus Sicht des Autors akzeptiert werden.

# 22 Zusammenfassung und Ausblick

Das Ziel der vorliegenden Arbeit war die Konzeption und Realisierung von graphischen Modellierungswerkzeugen zur Unterstützung der Softwaretechnik-Ausbildung an Fachhochschulen und Universitäten. Die Arbeit beruht auf der Beobachtung, dass das Lehren und Lernen graphischer Modellierungssprachen oft schwer ist. Gründe dafür sind die komplexe Syntax und Semantik vieler Sprachen und der hohe Abstraktionsgrad der aus ihnen resultierenden Modelle. Speziell die Semantik von Sprachen zur Modellierung dynamischer Zusammenhänge lässt sich in Vorlesungen nur schwer vermitteln und wird von den Studierenden vielfach entsprechend schlecht verstanden. Eine – wie auch immer geartete – Unterstützung durch Werkzeuge ist hier wünschenswert.

Existierende CASE-Werkzeuge können diese Unterstützung jedoch nicht leisten. Die Werkzeuge richten sich an Nutzer im industriellen Umfeld, bei denen eine Kenntnis der jeweiligen Sprache vorausgesetzt wird. Dementsprechend findet man in der Online-Hilfe eines solchen Werkzeugs üblicherweise nicht einmal eine Erläuterung der einzelnen Modellelemente. Die Werkzeuge sind außerdem zu schwergewichtig für einen Einsatz in der Lehre. Der große Funktionsumfang, die hohen Systemanforderungen und die in vielen Fällen schlechte Bedienbarkeit führen dazu, dass mehr Zeit in die Vermittlung des Werkzeugs als in die Lehre der Modellierungssprache investiert wird.

Die Arbeit stellt diesen schwergewichtigen professionellen CASE-Werkzeugen einen neuartigen Ansatz gegenüber, der sich in seinen Grundzügen an Erfahrungen aus dem Bereich der Werkzeugentwicklung für die Programmierausbildung orientiert. Sie betritt an mehreren Stellen Neuland und leistet dabei insbesondere folgende Beiträge:

- Die Arbeit schlägt eine Architektur für lehretaugliche Modellierungswerkzeuge vor. Die Werkzeuge sind Hybride aus in der Funktionalität beschränkten CASE-Werkzeugen und Lehr-/Lernumgebungen, so dass sie sich als leichtgewichtige Modellierungswerkzeuge mit zusätzlicher, didaktisch motivierter Funktionalität charakterisieren lassen.
- Die Arbeit entwickelt mit LiMo ein plattformunabhängiges Meta-CASE-Framework, aus dem mit geringem Aufwand leichtgewichtige Modellierungswerkzeuge abgeleitet werden können. Das generische Framework wird über die abstrakte und die konkrete Syntax einer Sprache parametrisiert.
- Die Arbeit stellt mit Dave, PetRA, Sam und ProMod/ProTut eine Reihe von konkreten Werkzeugen für die Softwaretechnik-Ausbildung bereit. Jedes Werkzeug fügt der Basis des Frameworks didaktisch motivierte Funktionalität (z.B. Simulationen) hinzu, die zum Lehren und Lernen der jeweiligen Sprache geeignet ist.

- Die Arbeit nutzt realitätsnahe Visualisierungen zur Veranschaulichung der Semantik eines Modells und um eine Brücke zwischen abstrakten Modellen und realen Systemen zu schlagen. Sie zeigt, wie White-Box- und Black-Box-Visualisierungen zu verschiedenen didaktischen Zwecken eingesetzt werden können.
- Die Arbeit zeigt die Tragfähigkeit des vorgestellten Ansatzes durch zwei umfangreiche evaluierte Einsätze des Werkzeugs Dave. Die Evaluationen belegen, dass sowohl die Leichtgewichtigkeit der Werkzeuge als auch die didaktisch motivierte Funktionalität von den Studierenden akzeptiert und als hilfreich empfunden wird.

Weiterer Forschungsbedarf im Anschluss an die Arbeit ergibt sich an mehreren Stellen. Auf der konzeptuellen Seite wäre es zum Beispiel sinnvoll, die genaue Wirkung der Simulation und der Visualisierung eines Modells in Kleingruppen unter Laborbedigungen zu untersuchen, um den Ansatz zu verfeinern. Die Idee der Black-Box-Visualisierung sollte sich dabei auch außerhalb von Modellierungswerkzeugen nutzen lassen, etwa im Rahmen der Programmierausbildung. Schließlich wäre es noch interessant, die offenbar existierenden Unterschiede zwischen Studentinnen und Studenten in der Einschätzung der Werkzeuge weiter zu untersuchen. Daraus lassen sich eventuell Hinweise für die Berücksichtigung von Gender-Aspekten innerhalb der Werkzeuge oder bei der Wahl von Beispielen und Visualisierungen ableiten.

Auf der eher praktischen Seite wäre neben der Weiterentwicklung und Pflege der existierenden Werkzeuge der Aufbau einer Bibliothek von Visualisierungen sinnvoll. Diese Dienstleistung könnte etwa durch das zentrale Distributionsportal des Projekts MuSofT [MUS07] geleistet werden. Wünschenswert ist selbstverständlich auch die Entwicklung weiterer Werkzeuge. Folgt man der zweiten Evaluation, dann besteht konkreter Bedarf an einem Werkzeug zur Modellierung und Simulation von Sequenzdiagrammen. Ein weiterer möglicher Kandidat wäre ein Werkzeug für Klassendiagramme, das die Simulation oder Validation des Modells mithilfe von Objektdiagrammen unterstützt. Als Simulationskomponente könnte dabei ein System wie der Infolayer [HP05] zum Einsatz kommen. Ein solches Werkzeug wäre dann zusätzlich zur Entwicklung der Metamodelle für weitere Werkzeuge nutzbar und könnte auf diese Weise den META-Case-Ansatz der Arbeit abrunden.

# Literaturverzeichnis


[ACS02] Allen, Eric, Robert Cartwright und Brian Stoler: *DrJava: A Lightweight Pedagogic Environment for Java.* In: *33rd SIGCSE Technical Symposium on Computer Science Education*, Seiten 137–141. ACM Press, 2002.

[AG97] Allen, Robert und David Garlan: *A Formal Basis for Architectural Connection.* ACM Transactions on Software Engineering and Methodology, 6(3):213–249, 1997.

[Ald91] Alderson, Albert: *Meta-CASE Technology.* In: *European Symposium on Software Development Environments and CASE Technology*, Nummer 509 in *LNCS*, Seiten 81–91. Springer, 1991.

[Ami04] Amiri, Zahir: *Entwicklung eines Petrinetz-Editors und -Simulators auf Basis des* LiMo*-Frameworks.* Diplomarbeit, Lehrstuhl für Software-Technologie, Universität Dortmund, 2004.

[Ani06] Aniszczyk, Chris: *Using GEF with EMF.* `http://www.eclipse.org/articles/Article-GEF-EMF/gef-emf.html`, online, Stand 05.11.2006.

[APS03] Alfert, Klaus, Jörg Pleumann und Jens Schröder: *A Framework for Lightweight Object-Oriented Design Tools.* Technischer Bericht Nr. 3, Lehrstuhl für Software-Technologie, Universität Dortmund, 2003.

[APS04a] Alfert, Klaus, Jörg Pleumann und Jens Schröder: *Eine Familie didaktischer Modellierungswerkzeuge für den Softwaretechnik-Unterricht.* In: Fellbaum, Klaus, Klaus Rebensburg und Andreas Schwill (Herausgeber): *Zweiter Workshop Grundfragen multimedialer Lehre (GML)*, Seiten 183–191. Books on Demand, 2004.

[APS04b] Alfert, Klaus, Jörg Pleumann und Jens Schröder: *Software Engineering Education Needs Adequate Modeling Tools.* In: Horton, Tom und Ann Sobel (Herausgeber): *17th Conference on Software Engineering Education & Training (CSEE&T)*, Seiten 72–77. IEEE Computer Society, 2004.

[Arg06] *ArgoUML Project Home.* `http://argouml.tigris.org`, online, Stand 25.04.2006.

[Bar98] Bardohl, Roswitha: *GENGED: A Generic Graphical Editor for Visual Languages Based on Algebraic Graph Grammars.* In: *IEEE Symposium on Visual Languages*, Seiten 48–55. IEEE Computer Society, 1998.

[Bau96] Baumgarten, Bernd: *Petri-Netze – Grundlagen und Anwendungen.* Spektrum Akademischer Verlag, 1996.

[BBMS01] Behnke, Ralf, Rudolf Berghammer, Erich Meyer und Peter Schneider: *RELVIEW - A System for Calculating with Relations and Relational Programming.* In: Astesiano, Egidio (Herausgeber): *Fundamental Approaches to Software Engineering*, Nummer 2075 in *LNCS*, Seiten 318–321. Springer, 2001.

[BCC+03] Bloom, James, Clare Clark, Camilla Clifford, Alex Duncan, Haroun Khan und Manos Papantoniou: *PIPE – Platform Independent Petri Net Editor.* Technischer Bericht, Department of Computing, Imperial College London, 2003.

[BCC+04] Barnwell, Tom, Michael Camacho, Matthew Cook, Maxim Gready, Peter Kyme und Michail Tsouchlaris: *Petri Net Analyser.* Technischer Bericht, Department of Computing, Imperial College London, 2004.

[BE01] Bardohl, Roswitha und Claudia Ermel: *Visual Specification and Parsing of a Statechart Variant using GenGEd.* In: *Symposium on Visual Languages and Formal Methods*, Seiten 5–7, 2001.

[Bea06] *BeanShell – Lightweight Scripting for Java.* `http://www.beanshell.org`, online, Stand 26.04.2006.

[BGMT98] Boudier, Gerard, Ferdinando Gallo, Regis Minot und Ian Thomas: *An Overview of PCTE and PCTE+.* ACM SIGSOFT Software Engineering Notes, 13(5):248–257, 1998.

[BLMA+01] Beaudouin-Lafon, Michel, Wendy E. Mackay, Peter Andersen, Paul Janecek, Mads Jensen, Michael Lassen, Kasper Lund, Kjeld Mortensen, Stephanie Munck, Anne Ratzer, Katrine Ravn, Søren Christensen und Kurt Jensen: *CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets.* In: Colom, José-Manuel und Maciej Koutny (Herausgeber): *Applications and Theory of Petri Nets*, Nummer 2075 in *LNCS*, Seiten 71–80. Springer, 2001.

[BMR+96] Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal: *Pattern-Oriented Sofware Architecture.* John Wiley and Sons, 1996.

[BNT02] Biddle, Robert, James Noble und Ewan Tempero: *Lightweight Web-Based Tools for Usage-Centered and Object-Oriented Design.* In: *1st International Conference on Usage-Centered Design, Performance-Centered Design, and Task-Oriented Design*, 2002.

[Boe76] Boehm, Barry W.: *Software Engineering.* IEEE Transactions on Computers, 25(12):1226–1241, 1976.

[Boe86] Boehm, Barry W.: *A Spiral Model of Software Development and Enhancement.* ACM SIGSOFT Software Engineering Notes, 11(4):14–24, 1986.

[Boo94] Booch, Grady: *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings, 1994.

[BPE06] *Business Process Execution Language for Web Services Version (BPEL4WS) Version 1.1.* `http://www-128.ibm.com/developerworks/library/specification/ws-bpel`, online, Stand 03.10.2006.

[BRJ05] Booch, Grady, James Rumbaugh und Ivar Jacobson: *The Unified Modeling Language User Guide.* Addison Wesley Longman, 2. Auflage, 2005.

[BS03] Börger, Egon und Robert Stärk: *Abstract State Machines. A Method for High-Level System Design and Analysis.* Springer, 2003.

[BvKU96] Berghammer, Rudolf, Burghard von Karger und Christiane Ulke: *Relation-algebraic analysis of Petri nets with RELVIEW.* In: Margaria, Tiziana und Berhard Steffen (Herausgeber): *2nd Workshop on Tools and Applications for the Construction and Analysis of Systems*, Nummer 1055 in *LNCS*, Seiten 49–69. Springer, 1996.

[Béz04] Bézivin, Jean: *On the Unification Power of Models.* In: Heckel, Reiko und Jan Hendrik Hausmann (Herausgeber): *Segravis – School on Foundations of Visual Modeling*, Seiten 5–48, 2004.

[CAB+00] Conway, Matthew, Steve Audia, Tommy Burnette, Dennis Cosgrove und Kevin Christiansen: *Alice: Lessons Learned from Building a 3D System for Novices.* In: *SIGCHI Conference on Human Factors in Computing Systems*, Seiten 486–493. ACM Press, 2000.

[CDFG06] Costagliola, Gennardo, Vincenzo Deufemia, Filomena Ferrucci und Carmine Gravino: *Constructing Meta-CASE Workbenches by Exploiting Visual Language Generators.* IEEE Transactions on Software Engineering, 32(3):156–175, 2006.

[CDP04] Costagliola, Gennardo, Vincenzo Deufemia und Guiseppe Polese: *A Framework for Modeling and Implementing Visual Notations with Applications to Software Engineering.* ACM Transactions on Software Engineering and Methodology, 13(4):431–487, 2004.

[CE01] Claudia Ermel, Roswitha Bardohl, Hartmut Ehrig: *Specification and Implementation of Animation Views for Petri Nets.* Technischer Bericht, Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, 2001.

[CEH+97] Corradini, Andrea, Hartmut Ehrig, Reiko Heckel, Michael Löwe, Ugo Montanari und Francesca Rossi: *Algebraic Approaches to Graph Transformation, Part I: Basic Concepts and Double Pushout Approach.* In: Rozenberg, Grzegorz (Herausgeber): *Handbook of Graph*

*Grammars and Computing by Graph Transformation, Volume 1: Foundations*, Seiten 163–245. World Scientific, 1997.

[Che76] Chen, Peter Pin-Shan: *The Entity-Relationship Model – Toward a Unified View of Data.* ACM Transactions on Database Systems, 1(1):9–36, 1976.

[CP00] Costagliola, Gennardo und Guiseppe Polese: *Extended Positional Grammars.* In: Stasko, John und Joseph J. Pfeiffer (Herausgeber): *IEEE Symposium on Visual Languages*, Seiten 103–110. IEEE Computer Society, 2000.

[DEH+05] Doberkat, Ernst-Erich, Gregor Engels, Jan Hendrik Hausmann, Marc Lohmann, Jörg Pleumann und Jens Schröder: *Software Engineering and eLearning: The MuSofT Project www.musoft.org.* e-Learning and Education, 1(2), 2005.

[DeM78] DeMarco, Tom: *Structured Analysis and System Specification.* Yourdon Press, 1978.

[Dia06] *Dia – A Drawing Program.* `http://www.gnome.org/projects/dia`, online, Stand 25.04.2006.

[Die04] Diekmann, Andreas: *Empirische Sozialforschung. Grundlagen, Methoden, Anwendungen.* Rowohlt, 10. Auflage, 2004.

[DKH97] Drewes, F., Hans-Jörg Kreowski und A. Habel: *Hyperedge Replacement Graph Grammars.* In: Rozenberg, Grzegorz (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, Seiten 95–162. World Scientific, 1997.

[DZ98] Dissmann, Stefan und Volker Zurwehn: *Software-Praktikum.* Teubner, 1998.

[Ecl06] *Eclipse - An Open Development Platform.* `http://www.eclipse.org`, online, Stand 05.11.2006.

[EG95] Emmerich, Wolfgang und Volker Gruhn: *Software Process Modelling with FUNSOFT Nets.* Technischer Bericht 47, Lehrstuhl für Software-Technologie, Universität Dortmund, 1995.

[EHK+97] Ehrig, Hartmut, Reiko Heckel, M. Korff, M. Löwe, L. Ribeiro, Annika Wagner und A. Corradini: *Algebraic Approaches to Graph Transformation, Part II: Single Pushput Approach and Comparison with Double Pushout Approach.* In: Rozenberg, Grzegorz (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, Seiten 163–245. World Scientific, 1997.

[ER97] Engelfriet, Joost und Grzegorz Rozenberg: *Node Replacement Graph Grammars.* In: Rozenberg, Grzegorz (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, Seiten 1–94. World Scientific, 1997.

[ERT99] Ermel, Claudia, Michael Rudolf und Gabi Taentzer: *The AGG approach: Language and Environment.* In: Ehrig, Hartmut, Gregor Engels, Hans-Jörg Kreowski und Grzegorz Rozenberg (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, Seiten 551–603. World Scientific, 1999.

[ES06] Euler, Timm und Martin Scholz: *The MiningMart User Guide.* Handbuch, Lehrstuhl für Künstliche Intelligenz, Universität Dortmund, 2006.

[ESU97] Ebert, Jürgen, Roger Süttenbach und Ingar Uhe: *Meta-CASE in Practice: a Case for KOGGE.* In: *9th International Conference on Advanced Information Systems Engineering*, Nummer 1250 in *LNCS*, Seiten 203–216. Springer, 1997.

[Eul05] Euler, Timm: *Publishing Operational Models of Data Mining Case Studies.* In: *Workshop on Data Mining Case Studies at the 5th IEEE International Conference on Data Mining (ICDM)*, Seiten 99–106. IEEE Computer Society, 2005.

[Eul06] Euler, Timm: *Knowledge Discovery in Databases at a Conceptual Level.* Doktorarbeit, Lehrstuhl für Künstliche Intelligenz, Universität Dortmund, 2006. Noch nicht erschienen.

[ExS06] *ExSpecT – Executable Specification Tool.* `http://www.exspect.com`, online, Stand 16.04.2006.

[FM00] Frey, Georg und Mark Minas: *Editing, Visualizing, and Implementing Signal Interpreted Petri Nets.* In: *7. Workshop Algorithmen und Werkzeuge fur Petrinetze (AWPN)*, Seiten 57–62, 2000.

[FNTZ98] Fischer, Thorsten, Jörg Niere, Lars Torunski und Albert Zündorf: *Story Diagrams: A New Graph Grammar Language based on the Unified Modelling Language and Java.* In: *6th International Workshop on Theory and Application of Graph Transformations*, Nummer 1764 in *LNCS*, Seiten 296–309. Springer, 1998.

[FP04] Fronk, Alexander und Jörg Pleumann: *Relationenalgebraische Analyse von Petri-Netzen: Konzepte und Implementierung.* In: *11. Workshop Algorithmen und Werkzeuge für Petrinetze (AWPN)*, Seiten 61–68, 2004.

[FP05a] Fronk, Alexander und Jörg Pleumann: *Relation-Algebraic Analysis of Petri Nets: Concepts and Implementation.* Petri Net News Letters, 2005.

[FP05b] Fronk, Alexander und Jörg Pleumann: *Relation-algebraic Calculation of Elementary Cycles.* In: Düntsch, Ivo, Wendy MacCaull und Michael Winter (Herausgeber): *8th International Seminar on Relational Methods in Computer Science*, 2005.

[FP06] Fronk, Alexander und Jörg Pleumann: *On Relational Cycles.* In: MacCaull, Wendy, Michael Winter und Ivo Düntsch (Herausgeber): *Relational Methods in Computer Science*, Nummer 3929 in *LNCS*, Seiten 83–95. Springer, 2006. Ausgewählte und erweiterte Fassung von [FP05b].

[Fra97] Franzke, A.: *GRAL: A Reference Manual.* Fachbericht Informatik 3/97, Fachbereich Informatik, Universität Koblenz-Landau, 1997.

[Fre00] Frey, Georg: *SIPN, Hierachical SIPN, and Extensions.* Technischer Bericht, Institut für Automatisierungstechnik, Universität Kaiserslautern, 2000.

[Fro04] Fronk, Alexander: *Using Relation Algebra for the Analysis of Petri Nets in a CASE Tool Based Approach.* In: *2nd International Conference on Software Engineering and Formal Methods*, Seiten 396–405. IEEE Computer Society, 2004.

[Fro05] Fronk, Alexander: *Relationenalgebraische Analyse von Petrinetzen.* Vorlesungsskript, Lehrstuhl für Software-Technologie, Universität Dortmund, 2005.

[Fug93] Fugetta, Alfonso: *A Classification of CASE Technology.* IEEE Computer, 26(12):25–38, 1993.

[GHJV95] Gamma, Erich, Richard Helm, Ralph E. Johnson und John Vlissides: *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley, 1995.

[GHP02] Gery, Eran, David Harel und Eldad Palachi: *Rhapsody: A Complete Life-Cycle Model-Based Development System.* In: Butler, Michael, Luigia Petre und Kaisa Sere (Herausgeber): *Integrated Formal Methods*, Nummer 2335 in *LNCS*, Seiten 1–10. Springer, 2002.

[Gor03] Gornik, Davor: *IBM Rational Unified Process: Best Practices for Software Development Teams.* Technischer Bericht, IBM, 2003.

[Gri06] Griffin, Catherine: *Using EMF.* `http://www.eclipse.org/articles/Article-Using-EMF/using-emf.html`, online, Stand 05.11.2006.

[GS79] Gane, Chris und Trish Sarson: *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, 1979.

[Har87] Harel, David: *Statecharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, 8(3):231–274, 1987.

[Har88] Harel, David: *On Visual Formalisms.* Communications of the ACM, 31(5):514–530, 1988.

[HCB04] Hendrix, T. Dean, James H. Cross und Larry A. Barowski: *An Extensible Framework for Providing Dynamic Data Structure Visualizations in a Lightweight IDE.* In: *International Conference of the Special*

*Interest Group for Computer Science Education (SIGCSE)*, Seiten 387–391. ACM Press, 2004.

[HLN+90] Harel, David, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring und Mark Trakhtenbrot: *STATEMATE – A Working Environment for the Development of Complex Reactive Systems.* IEEE Transactions on Software Engineering, 16(4):403–414, 1990.

[HM03] Harel, David und Rami Marelly: *Specifying and Executing Behavioral Requirements: the Play-in/Play-out Approach.* Software and Systems Modeling, 2(2):82–107, 2003.

[HN96] Harel, David und Amnon Naamad: *The STATEMATE Semantics of Statecharts.* ACM Transactions on Software Engineering and Methodology, 5(4):293–333, 1996.

[Hoa69] Hoare, Charles Antony Richard: *An Axiomatic Basis for Computer Programming.* Communications of the ACM, 12(10):576–580, 1969.

[Hoa78] Hoare, Charles Antony Richard: *Communicating Sequential Processes.* Communications of the ACM, 21(8):666–677, 1978.

[HP05] Haustein, Stefan und Jörg Pleumann: *A Model-Driven Runtime Environment for Web Applications.* Software and Systems Modeling, 4(4):443–458, 2005.

[HR04] Harel, David und Bernhard Rumpe: *Meaningful Modeling: What's the Semantics of „Semantics"?* IEEE Computer, 37(10):64–72, 2004.

[Huf92] Huff, Clifford C.: *Elements of a Realistic CASE Tool Adoption Budget.* Communications of the ACM, 35(4):45–54, 1992.

[Hum96] Humphrey, Watts S.: *Using a Defined and Measured Personal Software Process.* IEEE Software, 13(3):77–88, 1996.

[HWS00] Holt, Richard C., Andreas Winter und Andy Schürr: *GXL: Toward a Standard Exchange Format.* In: *Seventh Working Conference on Reverse Engineering*, 2000.

[IBM06a] *IBM Software: IBM Rational Method Composer.* `http://www-306.ibm.com/software/awdtools/rmc/index.html`, online, Stand 17.05.2006.

[IBM06b] *IBM Software: IBM Rational Unified Process.* `http://www-306.ibm.com/software/awdtools/rup/index.html`, online, Stand 17.05.2006.

[IBM06c] *IBM Software: DB2 Intelligent Miner.* `http://www-306.ibm.com/software/data/iminer`, online, Stand 31.12.2006.

[IBM96] *IBM Software: Rational Rose Technical Developer.* `http://www.ibm.com/software/awdtools/developer/technical`, online, Stand 25.04.2996.

[IET05] *Uniform Resource Identifier (URI): Generic Syntax.* Technischer Bericht RFC-3986, Internet Engineering Task Force, 2005.

[Iiv96] Iivari, Juhani: *Why are CASE Tools Not Used?* Communications of the ACM, 39(10):94–103, 1996.

[JBR98] Jacobson, Ivar, Grady Booch und James Rumbaugh: *The Unified Software Development Process.* Addison Wesley, 1998.

[JCJv92] Jacobson, Ivar, Magnus Christerson, Patrick Jonsson und Gunnar Övergaard: *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley, 1992.

[Jen96] Jensen, Kurt: *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use: Volume 1.* Springer, 2. Auflage, 1996.

[JHD06] *JHotDraw Start Page.* `http://www.jhotdraw.org`, online, Stand 12.10.2006.

[Joh92] Johnson, Ralph E.: *Documenting Frameworks using Patterns.* ACM SIGPLAN Notices, 27(10):63–76, 1992.

[KA02] Kopka, Corina und Klaus Alfert: *Der Softwareentwicklungsprozess als Lerngegenstand oder Von einem der auszieht, das reflektierte Handeln zu lehren.* In: Schubert, Sigrid, Bernd Reusch und Norbert Jesse (Herausgeber): *Informatik bewegt. 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, Nummer P-19 in *Lecture Notes in Informatics*, Seiten 401–407. Gesellschaft für Informatik, 2002.

[Kai04] Kaiser, Wolfram: *Become a programming Picasso with JHotDraw.* `http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-jhotdraw.html`, online, Stand 19.01.2004.

[Kem92] Kemerer, Chris F.: *How the Learning Curve Affects CASE Tools Adoption.* IEEE Software, 9(3):23–28, 1992.

[Ker01] Kerres, Michael: *Multimediale und telemediale Lernumgebungen.* Oldenbourg, 2. Auflage, 2001.

[Küh06] Kühne, Thomas: *Matters of (Meta-) Modeling.* Software and Systems Modeling, 5(4):369–385, 2006.

[Köl99a] Kölling, Michael: *The Problem of Teaching Object-Oriented Programming. Part II: Environments.* Journal of Object-Oriented Programming, 11(9):6–12, 1999.

[Köl99b] Kölling, Michael: *Teaching Object Orientation with the Blue Environment.* Journal of Object-Oriented Programming, 12(2):12–23, 1999.

[KMGSD04] Kamphans, Marion, Sigrid Metz-Göckel, Aira Schöttelndreier und Anna Drag: *Der Unified Process im Test. Evaluationsergebnisse zum Einsatz des Unified Process in der Informatik-Lehre.* MuSofT-Bericht Nr. 7, Lehrstuhl für Software-Technologie, Universität Dortmund, 2004.

[KMGT03] Kamphans, Marion, Sigrid Metz-Göckel und Anja Tigges: *Wie Geschlechteraspekte in die digitalen Medien integriert werden können – das*

*BMBF-Projekt MuSofT*. MuSofT-Bericht Nr. 4, Lehrstuhl für Software-Technologie, Universität Dortmund, 2003.

[KMGT+04] Kamphans, Marion, Sigrid Metz-Göckel, Anja Tigges, Anja Drag und Ellen Schröder: *Evaluation des Editors DAVE in der informatischen Hochschullehre*. MuSofT-Bericht Nr. 6, Lehrstuhl für Software-Technologie, Universität Dortmund, 2004.

[KP04] Kindler, Ekkart und Csaba Páles: *3D-Visualitazion of Petri Net Models: Concept and Realization*. In: Cortadella, Jordi und Wolfgang Reisig (Herausgeber): *Application and Theory of Petri Nets*, Nummer 3099 in *LNCS*, Seiten 464–473. Springer, 2004.

[KQPR03] Kölling, Michael, Bruce Quig, Andrew Patterson und John Rosenberg: *The Blue-J System and its Pedagogy*. Journal of Computer Science Education, 13(4):249–268, 2003.

[Kro02] Kromrey, Helmut: *Empirische Sozialforschung*. Leske und Budrich, 10. Auflage, 2002.

[Kru95] Kruchten, Philippe: *The 4+1 View Model of Architecture*. IEEE Software, 12(6):42–50, 1995.

[Kru03] Kruchten, Philippe: *The Rational Unified Software Process: An Introduction*. Addison Wesley, 2. Auflage, 2003.

[KS97] Klein, Peter und Andy Schürr: *Constructing SDEs with the IPSEN Meta Environment*. In: Ebert, Jürgen und Claus Lewerentz (Herausgeber): *8th Conference on Software Engineering Environments*, Seiten 2–10. IEEE Computer Society, 1997.

[KSS04] Kopka, Corina, Doris Schmedding und Jens Schröder: *Der Unified Process im Grundstudium – Didaktische Konzeption, Einsatz von Lehrmodulen und Erfahrungen*. In: Seehusen, Silke und Gregor Engels (Herausgeber): *Zweite e-Learning Fachtagung der Gesellschaft für Informatik (DeLFI)*, Nummer 52 in *LNI*, Seiten 127–138. Gesellschaft für Informatik, 2004.

[KSW95] Kiesel, Norbert, Andreas Schürr und Bernd Westfechtel: *GRAS, a Graph-Oriented (Software) Engineering Database System*. Information Systems, 20(1):21–52, 1995.

[Kum02] Kummer, Olaf: *Referenznetze*. Doktorarbeit, Arbeitsgruppe Theoretische Informatik, Universität Hamburg, 2002.

[KW01] Kindler, Ekkart und Michael Weber: *The Petri Net Kernel – An Infrastructure for Building Petri Net Tools*. Software Tools for Technology Transfer, 3(4):486–497, 2001.

[KWB03] Kleppe, Anneke, Jos Warmer und Wim Bast: *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

[KWD+04] Kummer, Olaf, Frank Wienberg, Michael Duvigneau, Jörg Schuhmacher, Michael Köhler, Daniel Moldt, Heiko Rölke und Rüdiger Valk: *An Extensible Editor and Simulation Engine for Petri Nets: RENEW.* In: Cortadella, Jordi und Wolfgang Reisig (Herausgeber): *Application and Theory of Petri Nets*, Nummer 3099 in *LNCS*, Seiten 484–493. Springer, 2004.

[LC98] Lending, Diane und Norman L. Chervany: *The Use of CASE Tools.* In: *Conference of the Special Interest Group on Computer Personnel Research*, Seiten 49–58. ACM Press, 1998.

[LEG06] *LEGO Mindstorms Robotics Invention System 2.0.* `http://minstorms.lego.com`, online, Stand 16.11.2006.

[LKA+95] Luckham, David C., John J. Kenney, Larry M. Augustin, James Vera, Dough Bryan und Walter Mann: *Specification and Analysis of System Architecture Using Rapide.* IEEE Transactions on Software Engineering, 21(4):336–355, 1995.

[LTX02] Lorentsen, Louise, Antti-Pekka Tuovinen und Jianli Xu: *Modelling of Features and Feature Interactions in Nokia Mobile Phones.* In: Esparza, J. und C. Lakos (Herausgeber): *Application and Theory of Petri Nets*, Nummer 2360 in *LNCS*, Seiten 294–313. Springer, 2002.

[Lyo03] Lyons, Andrew: *UML for Real-Time Overview.* `http://www.rational.com/products/whitepapers/100463.jsp`, online, Stand 06.01.2003.

[Mag06] *Magic Draw – Architecture Made Simple.* `http://www.magicdraw.com`, online, Stand 25.04.2006.

[MBBF03] Michiels, Isabel, Jürgen Börstler, Kim B. Bruce und Alejandro Fernández: *Tools and Environments for Learning Object-Oriented Concepts.* In: *ECOOP 2003 Workshop Reader*, Nummer 3013 in *LNCS*, Seiten 119–129. Springer, 2003.

[MEH01] Maier, Mark W., David Emery und Rich Hilliard: *Software Architecture: Introducing IEEE Standard 1471.* IEEE Computer, 34(4):107–109, 2001.

[Mey97] Meyer, Betrand: *Object-Oriented Software Construction*, 1997.

[MH01] Minas, Mark und Berthold Hoffmann: *Specifying and Implementing Visual Process Modeling Languages with DiaGen.* Electronic Notes in Theoretical Computer Science, 44(4), 2001.

[MH03] Mueller, Frank und Antony L. Hosking: *Penumbra: An Eclipse Plugin for Introductory Programming.* In: *OOPSLA Workshop on Eclipse Technology Exchange*, Seiten 65–68. ACM Press, 2003.

[Min00] Minas, Mark: *Hypergraphs as a Uniform Diagram Representation Model.* In: Ehrig, Hartmut, Gregor Engels, Hans-Jörg Kreowski und Grzegorz Rozenberg (Herausgeber): *6th International Workshop*

*on Theory and Application of Graph Transformation*, Nummer 1764 in *LNCS*. Springer, 2000.

[Min06] *MiningMart – Enabling End-User Datawarehouse Mining.* `http://mmart.cs.uni-dortmund.de`, online, Stand 07.05.2006.

[MK00] Monecke, Marc und Udo Kelter: *Eine integrierte Werkzeugsammlung für den Einsatz in der Softwaretechnik-Ausbildung.* Technischer Bericht, Fachgruppe Praktische Informatik, Universität Siegen, 2000.

[ML05] McAffer, Jeff und Jean-Michel Lemieux: *Eclipse Rich Client Platform. Designing, Coding, and Packaging Java Applications.* Addison Wesley, 2005.

[MMW98] Marriott, Kim, Bernd Meyer und Kent B. Wittenburg: *A Survey on Visual Language Specification and Recognition.* In: Marriott, Kim und Bernd Meyer (Herausgeber): *Visual Language Theory*, Seiten 5–85. Springer, 1998.

[Moc06] Mocek, Christian: *Erweiterung des CASE-Werkzeugs DAVE um einen Code-Generator für LEGO Mindstorms.* Diplomarbeit, Lehrstuhl für Software-Technologie, Universität Dortmund, 2006.

[Mon03] Monecke, Marc: *Adaptierbare CASE-Werkzeuge in prozessorientierten Software-Entwicklungsumgebungen.* Doktorarbeit, Fachbereich Elektrotechnik und Informatik der Universität-Gesamthochschule Siegen, 2003.

[MS04] Morik, Katharina und Martin Scholz: *The MiningMart Approach to Knowledge Discovery in Databases.* In: Zhong, Ning und Jiming Liu (Herausgeber): *Intelligent Technologies for Information Analysis.* Springer, 2004. Noch nicht erschienen.

[MUS07] *MuSofT – Multimedia in der SoftwareTechnik.* `http://www.musoft.org`, online, Stand 04.01.2007.

[MV95] Minas, Mark und Gerhard Viehstaedt: *DiaGen: A Generator for Diagram editor Providing Direct Manipulation and Execution of Diagrams.* In: *IEEE Symposium on Visual Languages*, Seiten 203–210. IEEE Computer Society, 1995.

[Nag93] Nagl, Manfred: *Software-Entwicklungsumgebungen: Einordnung und zukünftige Entwicklungslinien.* Informatik-Spektrum, 16(5):273–280, 1993.

[NB04] Noble, James und Robert Biddle: *Notes on Notes on Postmodern Programming.* ACM SIGPLAN Notices, 39(12):40–56, 2004.

[NE93] *Reference Model for Frameworks of Software Engineering Environments 3rd Edition.* Technischer Bericht NIST Special Publication 500-211, ECMA Technical Report TR/55, National Institute of Standards and Technology (NIST) / European Computer Manufacturers Association (ECMA), 1993.

[Nie93] Nielsen, Jakob: *Usability Engineering.* Morgan Kaufmann, 1993.

[NNZ00] Nickel, Ulrich, Jörg Niere und Albert Zündorf: *Tool Demonstration: The Fujaba Environment.* In: *International Conference on Software Engineering*, Seiten 742–745. ACM Press, 2000.

[NW04] Northover, Steve und Mike Wilson: *SWT: The Standard Widget Toolkit, Volume 1.* Addison Wesley, 2004.

[OMG03a] *MDA Guide Version 1.0.1.* Technischer Bericht, Object Management Group, 2003.

[OMG03b] *Unified Modeling Language (UML) Specification Version 1.5.* Technischer Bericht, Object Management Group, 2003.

[OMG05a] *Unified Modeling Language (UML) Superstructure Specification Version 2.0.* Technischer Bericht, Object Management Group, 2005.

[OMG05b] *XML Metadata Interchange (XMI) Version 2.1*, 2005.

[OMG06] *Meta Object Facility (MOF) Core, Version 2.0.* Technischer Bericht, Object Management Group, online, Stand 10.10.2006.

[Ort05] Orth, Philip: *Rapid Control Prototyping diskreter Steuerungen mit Petrinetzen.* Nummer 1085 in *Fortschritt-Berichte VDI, Reihe 8.* VDI-Verlag, 2005. Dissertation.

[Par92] Parker, Burt: *Introducing EIA-CDIF: the CASE Data Interchange Format Standard.* In: *Second Symposium on Assessment of Quality Software Development Tools*, Seiten 74–82. IEEE Computer Society, 1992.

[Pat81] Pattis, Richard E.: *Karel the Robot: A Gentle Introduction to the Art of Programming.* Wiley, 1981.

[Paw05] Pawlowski, Jan M.: *Das Qualitätssiegel E-Learning (QSEL): Qualitätsentwicklung für Organisationen und Produkte.* In: Kerres, Michael und Reinhard Keil-Slawik (Herausgeber): *Education Quality Forum, Band 2.* Waxmann, 2005.

[Pet62] Petri, Carl Adam: *Kommunikation mit Automaten.* Technischer Bericht 2, Institut für Instrumentelle Mathematik, Bonn, 1962.

[Pet05] Petermeier, Robert: *Integration of a Graph-Grammar-Based Syntax Analysis into the CASE Tool DAVE.* Diplomarbeit, Lehrstuhl für Software-Technologie, Universität Dortmund, 2005.

[Pet06] *Petri Nets World.* `http://www.informatik.uni-hamburg.de/TGI/PetriNets`, online, Stand 16.04.2006.

[PF01] Paula Filho, Wilson P.: *Requirements for an Educational Software Development Process.* In: *6th International Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Seiten 65 – 68. ACM Press, 2001.

[Pip06] *PIPE 2 – Platform Independent Petri Net Editor 2.* `http://pipe2.sourceforge.net`, online, Stand 16.04.2006.

[Pla06] Plante, Frederic: *Introducing the GMF Runtime.* `http://www.eclipse.org/articles/Article-Introducing-GMF/article.html`, online, Stand 05.11.2006.

[Ple03] Pleumann, Jörg: *A Framework for Lightweight Graphical Modeling Applications.* In: Callaos, Nagib, Maurice Margenstern, Jun Zhang, Oscar Castillo und Ernst-Erich Doberkat (Herausgeber): *7th World Multiconference on Systemics, Cybernetics, and Informatics*, Band V, Seiten 440–445. International Institute of Informatics and Systemics (IIS), 2003.

[Ple04] Pleumann, Jörg: *Erfahrungen mit dem multimedialen, didaktischen Modellierungswerkzeug DAVE.* In: Seehusen, Silke und Gregor Engels (Herausgeber): *Zweite e-Learning Fachtagung der Gesellschaft für Informatik (DeLFI)*, Nummer 52 in *LNI*, Seiten 55–66. Gesellschaft für Informatik, 2004. Auszeichnung für den besten Beitrag der Konferenz.

[Plo81] Plotkin, Gordon D.: *A Structural Approach to Operational Semantics.* Technischer Bericht FN-19, Department of Computer Science, University of Aarhus, 1981.

[Pos06] *Gentleware: Poseidon for UML.* `http://gentleware.com/poseidon_for_uml.0.html`, online, Stand 25.04.2006.

[Pre97] Pree, Wolfgang: *Komponentenbasierte Softwareentwicklung mit Frameworks.* dpunkt, 1997.

[PS04] Pleumann, Jörg und Jens Schröder: *Didaktische Modellierungswerkzeuge für die Präsenzlehre der Softwaretechnik.* In: Dadam, Peter und Manfred Reichert (Herausgeber): *34. Jahrestagung der Gesellschaft für Informatik (GI)*, Nummer 50 in *LNI*, Seiten 409–413. Gesellschaft für Informatik, 2004.

[PW92] Perry, David E. und Alexander L. Wolf: *Foundations for the Study of Software Architecture.* ACM SIGSOFT Software Engineering Notes, 17(4):40–52, 1992.

[RB06] Rausch, Andreas und Manfred Broy: *Das V-Modell XT. Grundlagen, Erfahrungen und Werkzeuge.* dpunkt, 2006.

[RBP+91] Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy und William Lorensen: *Object-Oriented Modeling and Design.* Pretice Hall, 1991.

[Ren06] *RENEW – The Reference Net Workshop.* `http://www.renew.de`, online, Stand 16.04.2006.

[RHR98] Robbins, Jason E., David M. Hilbert und David F. Redmiles: *Software Architecture Critics in Argo.* In: *3rd International Conference on Intelligent User Interfaces*, Seiten 141–144. ACM Press, 1998.

[Roy87] Royce, Winston W.: *Managing the Development of Large Software Systems.* In: *International Conference of Software Engineering (ICSE)*, Seiten 328–338. IEEE Computer Society, 1987. Nachdruck des Original-Aufsatzes von 1970.

[Roz97] Rozenberg, Grzegorz (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations.* World Scientific, 1997.

[RR99] Robbins, Jason E. und David F. Redmiles: *Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML.* Technischer Bericht, Department of Information and Computer Science, University of California, Irvine, 1999.

[RRT06] Romei, Andrea, Salvatore Ruggieri und Franco Turini: *KDDML: A Middleware Language and System for Knowledge Discovery in Databases.* Data and Knowledge Engineering, 57(2):179–220, 2006.

[RS95] Rasmussen, Jens Linneberg und Mejar Singh: *Mimic/CPN – A Graphic Animation Utility for Design/CPN.* User's Manual, Department of Computer Science, University of Aarhus, 1995.

[RS00] Roch, Stephan und Peter H. Starke: *INA Integrated Net Analyzer Version 2.2 Manual.* Technischer Bericht, Institut für Informatik, Humboldt-Universität zu Berlin, 2000.

[RWL+03] Ratzer, Anne Vinter, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen und Kurt Jensen: *CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets.* In: Aalst, Eike Best Wil van der (Herausgeber): *Applications and Theory of Petri Nets*, Nummer 2679 in *LNCS*, Seiten 450–462. Springer, 2003.

[SAS06] *SAS Institute Inc.: SAS Intelligent Miner.* `http://www.sas.com/technologies/analytics/datamining/miner`, online, Stand 31.12.2006.

[Sch97] Schürr, Andreas: *Programmed Graph Replacement Systems.* In: Rozenberg, Grzegorz (Herausgeber): *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, Seiten 479–546. World Scientific, 1997.

[SCK97] Soren Christensen, Jens Baek Jorgensen und Lars Michael Kristensen: *Design/CPN – A Computer Tool for Coloured Petri Nets.* Technischer Bericht, Department of Computer Science, University of Aarhus, 1997.

[SCT00] Smith, David C., Allen Cypher und Larry Tesler: *Novice Programming Comes of Age.* Communications of the ACM, 43(3):75–81, 2000.

[SDK+95] Shaw, Mary, Robert DeLine, Daniel Klein, Theodore Ross, David Young und Gregory Zelesnik: *Abstractions for Software Architecture and Tools to Support Them.* IEEE Transactions on Software Engineering, 21(3):314–335, 1995.

[SG95] Shaw, Mary und David Garlan: *Software Architecture – Perspectives on an Emerging Discipline.* Prentice Hall, 1995.

[SGW94] Selic, Bran, Garth Gullekson und Paul T. Ward: *Real-Time Object-Oriented Modeling.* John Wiley and Sons, 1994.

[SMM+03] Storey, Margaret-Anne, Jeff Michaud, Marcellus Mindel, Daniela Damian, Del Myers, Daniel German und Elizabeth Hargreaves: *Improving the Usability of Eclipse for Novice Programmers.* In: *OOPSLA Workshop on Eclipse Technology Exchange*, Seiten 35–39. ACM Press, 2003.

[Som04] Sommerville, Ian: *Software Engineering.* Addison Wesley, 7. Auflage, 2004.

[Spi92] Spivey, Michael: *The Z Notation: A Reference Manual.* Prentice Hall, 1992.

[SR00] Starke, Peter H. und Stephan Roch: *INA et al.* Technischer Bericht, Institut für Informatik, Humboldt-Universität zu Berlin, 2000.

[SR03] Selic, Bran und Jim Rumbaugh: *Using UML for Modeling Complex Real-Time Systems.* `http://www.rational.com/products/whitepapers/442.jsp`, online, Stand 06.01.2003.

[SS93] Schmidt, Gunther und Thomas Ströhlein: *Relations and Graphs – Discrete Mathematics for Computer Scientists.* EACTS Monographs on Theoretical Computer Science. Springer, 1993.

[Sti00] Stiegler, Barbara: *Wie Gender in den Mainstream kommt. Konzepte, Argumente und Praxisbeispiele zur EU-Strategie des Gender Mainstreaming.* Technischer Bericht, Forschungsinstitut der Friedrich-Ebert-Stiftung, Abteilung Arbeit und Sozialpolitik, 2000.

[SW99] Stafford, Judith A. und Alexander L. Wolf: *Architecture-Based Software Engineering.* Technischer Bericht, University of Colorado, 1999. Internal Report CU-CS-891-99.

[SWZ95] Schürr, Andreas, Andreas Winter und Albert Zündorf: *Graph-Grammar Engineering with PROGRES.* In: Schäfer, Wilhelm (Herausgeber): *European Software Engineering Conference*, Nummer 989 in *LNCS*, Seiten 219–234. Springer, 1995.

[Szy03] Szymanski, Oliver: *Relationale Algebra im dreidimensionalen Software-Entwurf – ein werkzeugbasierter Ansatz.* Diplomarbeit, Lehrstuhl für Software-Technologie, Universität Dortmund, 2003.

[Tig06] Tigges, Anja: *Geschlecht und digitale Medien. Eine empirische Studie über die Entwicklung und Nutzung digitaler Medien im hochschulischen Lehr-/Lernkontext.* Doktorarbeit, Fachbereich Erziehungswissenschaften, Universität Dortmund, 2006. Noch nicht erschienen.

[TN92] Thomas, Ian und Brian A. Nejmeh: *Definition of Tool Integration for Environments.* IEEE Software, 9(2):29–35, 1992.

[Tog06] *Borland Together – Model Driven Architecture and Visual Modeling Application.* `http://www.borland.com/us/products/together`, online, Stand 25.04.2006.

[vdAdCG+00] Aalst, Wil M.P. van der, Poul J.N. de Crom, Roy R.H.M.J. Goverde, Kees M. van Hee, Wout J. Hofman, Hajo A. Reijers und Robert A. van der Toorn: *ExSpect 6.4 An Executable Specification Tool for Hierarchical Colored Petri Nets.* In: Nielsen, Morgens und Dan Simpson (Herausgeber): *Applications and Theory of Petri Nets*, Nummer 1825 in *LNCS*, Seite 455. Springer, 2000.

[vdB94] Beek, Michael von der: *A Comparison of Statecharts Variants.* In: Langmaack, Hans, Willem-Paul de Roever und Jan Vytopil (Herausgeber): *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Nummer 863 in *LNCS*. Springer, 1994.

[vGB99] Gurp, Jilles van und Jan Bosch: *On the Implementation of Finite State Machines.* In: *3rd Annual IASTED International Conference Software Engineering and Applications*, Seiten 172–178. IASTED, 1999.

[Vis06] *Microsoft Office Online: Visio 2003 Homepage.* `http://office.microsoft.com/en-us/FX010857981033.aspx`, online, Stand 25.04.2006.

[VMo06] *Das neue V-Modell XT Release 1.2 - Der Entwicklungsstandard für IT-Systeme des Bundes.* `http://www.v-modell-xt.de`, online, Stand 15.05.2006.

[VRM97] *ISO/IEC International Standard, Information Technology, Computer Graphics and Image Processing. The Virtual Reality Modeling Language (VRML) – Part 1: Functional specification and UTF-8 encoding.* Technischer Bericht 14772-1, ISO/IEC, 1997.

[W3C99a] *Hypertext Markup Language (HTML) Specification Version 4.01.* Technischer Bericht, World Wide Web Consortium, 1999.

[W3C99b] *XSL Transformations (XSLT) Version 1.0.* Technischer Bericht, World Wide Web Consortium, 1999.

[W3C04] *Document Object Model (DOM) Level 3 Core Specification.* Technischer Bericht, World Wide Web Consortium, 2004.

[WBJ90] Wirfs-Brock, Rebecca und Ralph Johnson: *Surveying Current Research in Object-Oriented Design.* Communications of the ACM, 33(9):104–124, 1990.

[Wie04] Wienold, Kirsten: *Evaluation onlinebasierter Lehr-/Lernsysteme. Anforderungen an Instrumente zur Evaluation neuer Medien.* Doktorarbeit, 2004.

[Wil03] Wilson, James M.: *Gantt Charts: A Centenary Appreciation.* European Journal of Operational Research, 149(2):430–437, 2003.

[Wir94] Wirsing, Martin: *Algebraic Specification Languages: An Overview.* In: Astesiano, Egidio, Gianna Reggio und Andrzej Tarlecki (Herausgeber): *10th Workshop on Specification of Abstract Data Types*, LNCS, Seiten 81–115. Springer, 1994.

[WK95] Warmer, Jos und Anneke Kleppe: *The Object Constraint Language.* Addison Wesley, 1995.

[WPD01] Wagner, Annika, Jörg Pleumann und Stefan Dissmann: *Abschlussbericht des Projekts „Modellieren in einer virtuellen Welt“.* Technischer Bericht 164, Lehrstuhl für Software-Technologie, Universität Dortmund, 2001.

Der Anteil des Autors an für die Arbeit relevanten, gemeinschaftlich verfassten Publikationen ist wie folgt:

- Für [APS03, APS04a, APS04b, PS04] ist der Beitrag 75%. Der Autor hat das LiMo-Framework und die beiden Werkzeuge Dave und Sam entwickelt sowie die Evaluation von Dave durchgeführt. Jens Schröder hat die Werkzeuge ProMod und ProTut entwickelt.
- Für [FP04, FP05a] ist der Beitrag 50% und für [FP05b, FP06] beträgt er 25%. Der Autor hat das Werkzeug PetRA entwickelt. Die theoretischen Grundlagen zur relationalen Analyse von Petrinetzen stammen von Alexander Fronk.
- Für [DEH+05] ist der Beitrag 25%. Das Papier beschreibt neben den Arbeiten des Autors auch die restlichen Ergebnisse des BMBF-Projekts MuSofT.

# A Technische Details

| Präfix | Beschreibung |
|---|---|
| `model:` | Verweist auf ein Modellelement (und damit implizit auf eine Seite des variablen Teils). Der restliche URI enthält in diesem Fall den ausführlichen Namen des Elements relativ zur Wurzel des Modells, also z.B. `model:MySystem.UseCase1`. Die entsprechende Figur der Zeichnung wird ausgewählt, und die zugehörige Seite wird angezeigt. |
| `help:` | Verweist auf eine Seite aus der Hilfe (also aus dem fixen Teil des Hypertextes). Der restliche URI enthält in diesem Fall den Namen der HTML-Datei, also z.B. `http:toolbar.html`. Verweise innerhalb der Hilfe (auch auf Grafiken oder andere Multimedia-Objekte) sollten relative URIs verwenden, damit die Hilfe und damit die Anwendung portabel bleibt. |
| `extern:` | Verweist auf ein Programm oder Dokument, das außerhalb von LiMo aufgerufen bzw. angezeigt werden soll. Der restliche URI gibt in diesem Fall den Pfad an, also z.B. `extern:c:\requirements.doc`, und ist insbesondere wieder ein vollständiger URI. Das Framework erzeugt einen neuen Prozess und übergibt diesem als Kommandozeile den restlichen URI. |
| `file:` | Verweist auf eine Datei innerhalb des lokalen Dateisystems. Dateien mit den Endungen `.html`, `.rtf` und `.txt` werden im Hypertext-Bereich angezeigt. Dateien mit der Endung `.xml` werden als Modell in das Werkzeug geladen. Gegebenenfalls wird zuvor gefragt, ob das aktuelle Modell gespeichert werden soll. |
| `http:` | Verweist auf ein Dokument, das über das Netz zu beziehen ist. Dateien werden – wie beim Präfix `file:` – anhand ihrer Endung behandelt. Auf diese Weise geladene Modelle können offensichtlich nicht unter dem gleichen URI abgespeichert werden. Beim Versuch zu Speichern wird deshalb ein neuer Name erfragt. |

Tabelle A.1: Nutzung von Protokollpräfixen im Hypertext-System

| Variable | Typ | Beschreibung |
|---|---|---|
| `editor` | `Application` | Verweist auf die laufende Anwendung. Immer definiert. |
| `root` | `Model` | Verweist auf die Wurzel des aktuellen Modells. Undefiniert, wenn kein Modell geöffnet ist. |
| `model` | `Model` | Verweist auf das in der aktuellen Zeichnung dargestellte (Teil-) Modell. Undefiniert, wenn keine Zeichnung sichtbar ist. |
| `element` | `ModelElement[]` | Verweist auf die derzeit ausgewählten Modellelement(e). Leer, wenn keine Element ausgewählt sind. |
| `figure` | `ModelFigure[]` | Verweist auf die derzeit ausgewählte(n) Figur(en). Leer, wenn keine Figur ausgewählt ist. |
| `connection` | `ModelConnection[]` | Verweist auf die derzeit ausgewählte(n) Verbindung(en). Leer, wenn keine Verbindung ausgewählt ist. |

Tabelle A.2: Schnittstellen des LiMo-Frameworks zum Hypertext-System

| Attribut | Element(e) | Beschreibung |
|---|---|---|
| `onload` | `<html>`, `<body>` | Legt den Code fest, der beim Anzeigen der Seite ausgeführt wird. |
| `onunload` | `<html>`, `<body>` | Legt den Code fest, der beim Ausblenden der Seite ausgeführt wird. |
| `onmouseenter` | `<a>` | Legt den Code fest, der beim Betreten des Ankers durch den Mauszeiger ausgeführt wird. |
| `onmouseout` | `<a>` | Legt den Code fest, der ausgeführt wird, wenn der Mauszeiger den Anker verlässt. |
| `onclick` | `<a>` | Legt den Code fest, der beim Verfolgen eines Hyperlinks ausgeführt wird. |

Tabelle A.3: Einbettung von Skriptaufrufen in Hypertext-Seiten

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- Das Wurzelelement definiert ein Menü. -->

  <xsd:element name="Menu" type="MenuType"/>

  <!-- Ein Menü besitzt beliebig viele Untermenüs, Elemente und Titel sowie einen Namen. -->

  <xsd:complexType name="MenuType" >
    <xsd:sequence>
      <xsd:element name="Title" type="TitleType"/>
      <xsd:element name="Menu" type="MenuType"/>
      <xsd:element name="Item" type="ItemType"/>
    </xsd:sequence>

    <xsd:attribute name="name" type="xsd:string"/>
  </xsd:complexType>

  <!--
    Ein Menüelement besitzt beliebig viele Titel und ein Kommando.
    Es kann außerdem mit einem Namen versehen werden und einen Typ
    besitzen. Ferner kann festgelegt werden, ob das Modell vor der
    Ausführung des Kommandos gespeichert wird und ob das Werkzeug
    auf die Ausführung des Kommandos wartet.

    Im Kommando werden die folgenden Zeichenfolgen automatisch ersetzt:
    - %file% durch den Dateinamen des Modells (ohne Verzeichnis)
    - %dir% durch das Verzeichnis des Modells (ohne Dateinamen)
    - %path% durch den kompletten Pfad des Modells
    - %name% durch den Namen des Modells
  -->

  <xsd:complexType name="ItemType" >
    <xsd:sequence>
      <xsd:element name="Title" type="TitleType"/>
      <xsd:element name="Command" type="xsd:string" maxOccurs="1"/>
    </xsd:sequence>

    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="save" type="xsd:boolean"/>
    <xsd:attribute name="wait" type="xsd:boolean"/>
  </xsd:complexType>

  <!--
    Ein Titel ist zunächst ein einfacher String. Er kann mit einer Sprache versehen
    werden. Das System wählt dann zur Laufzeit den passenden Titel aus.
  -->

  <xsd:complexType name="TitleType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="lang" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

</xsd:schema>
```

Abbildung A.1: XML-Schema zur Definition von Menüs

```
package org.musoft.statemachine.samples;

import java.awt.BorderLayout;

import javax.swing.*;

import org.musoft.statemachine.model.UmlState;
import org.musoft.statemachine.simulation.StateMachineDevice;
public class WhiteBoxTV extends StateMachineDevice {

    private JLabel myLabel = new JLabel();

    public JPanel createContentPanel() {
        JPanel buttons = new JPanel();
        buttons.add(createButton("Ein-/ausschalten", "netz.png", "POWER");
        buttons.add(createButton("Sender wechseln", "sender.png", "STATION");

        JPanel panel = new JPanel(new BorderLayout());
        panel.add(buttons, BorderLayout.NORTH);
        panel.add(myLabel, BorderLayout.CENTER);

        myLabel.setIcon(createIcon("schwarz.png"));

        return panel;
    }

    public void onEnterState(UmlState state) {
        String name = state.getName();

        if ("ARD".equals(name))
            myLabel.setIcon(createIcon("ard.png"));
        else if ("ZDF".equals(name))
            myLabel.setIcon(createIcon("zdf.png"));
        else if ("WDR".equals(name))
            myLabel.setIcon(createIcon("wdr.png"));
        else if ("AUS".equals(name))
            myLabel.setIcon(createIcon("schwarz.png"));
    }
}
```

Abbildung A.2: Implementierung einer White-Box-Visualisierung für DAVE

```
1  package org.musoft.statemachine.samples;

   import java.awt.BorderLayout;
   import javax.swing.*;
   import org.musoft.statemachine.simulation.StateMachineDevice;
6
   public class BlackBoxTV extends StateMachineDevice {

       private JLabel label = new JLabel();

11     private boolean crt = false;

       private int channel = 0;

       public JPanel createContentPanel() {
16         JPanel buttons = new JPanel();
           buttons.add(createButton("Ein-/ausschalten", "netz.png", "POWER");
           buttons.add(createButton("Sender wechseln", "sender.png", "STATION");

           JPanel panel = new JPanel(new BorderLayout());
21         panel.add(buttons, BorderLayout.NORTH);
           panel.add(label, BorderLayout.CENTER);

           label.setIcon(createIcon("schwarz.png"));

26         return panel;
       }

       public void onExecuteAction(String event) {
           if (event.startsWith("CRT("))
31             crt = event.substring(4, event.length() - 1).equals("true");
           else if (event.startsWith("CHANNEL("))
               channel = Integer.parseInt(event.substring(8, event.length() - 1));
       }

36     public void onEndStep() {
           if (crt) {
               if (channel == 3)
                   label.setIcon(createIcon("ard.png"));
               else if (channel == 25)
41                 label.setIcon(createIcon("zdf.png"));
               else if (channel == 38)
                   label.setIcon(createIcon("wdr.png"));
               else
                   label.setIcon(createIcon("rauschen.png"));
46         }
           else {
               label.setIcon(createIcon("schwarz.png"));
           }
       }
51 }
```

Abbildung A.3: Implementierung einer Black-Box-Visualisierung für DAVE

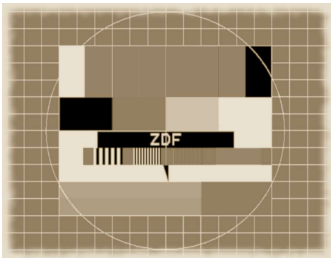
# B Übungsaufgaben

UNIVERSITÄT DORTMUND

Fachbereich Informatik
Lehrstuhl für Software-Technologie

**Aufgabe 1 (Fernseher):**

Lange bevor die Welt mit Kabelfernsehen und Satellitenschüsseln gesegnet wurde, hatten Fernseher nur zwei Farben, drei Programme (ARD, ZDF und WEST3 oder ein anderes drittes Programm) und vier Tasten, über die sie kontrolliert wurden: Es gab zwei Tasten um das Gerät ein- und auszuschalten und zwei weitere, um ein Programm vor- bzw. zurückzuschalten.




Das Wechseln der Programme geschah zyklisch, d.h. nach dem dritten Programm kam wieder das erste und umgekehrt. Die meisten Menschen merkten deshalb gar nicht, dass sie tatsächlich nur drei Programme hatten und schauten so lange fern, bis sie vor dem Gerät einschliefen.

Da es fast niemand sehr lange vor dem Fernseher aushielt, ohne dabei einzuschlafen, schalteten sich die Geräte nach sechs Stunden automatisch ab. Besonders teure Geräte waren in der Lage, sich das zuletzt angeschaute Programm zu merken und beim nächsten Einschalten wieder zu aktivieren.

Modellieren Sie ein solches Gerät mit einem Zustandsdiagramm.

**Aufgabe 2 (Ampelkreuzung):**

Modellieren Sie die Ampelanlage einer Verkehrskreuzung mit einem Zustandsdiagramm.

Eine Verkehrskreuzung hat an allen vier Ecken Ampeln, von denen Paare von gegenüberliegenden Ampeln stets dieselbe Farbe zeigen. Wir nehmen der Einfachheit halber an, dass wir nur die Farben rot und grün zur Verfügung haben. Zeigt ein Ampelpaar die Farbe rot, so zeigt das andere die Farbe grün. Alle 30 Sekunden wechselt die Farbe.

Nachts wird die komplette Ampelanlage über ein Signal der Betriebszentrale ausgeschaltet. Beim Einschalten am folgenden Morgen, das ebenfalls über ein Signal der Betriebszentrale erfolgt, zeigt jede Ampel dieselbe Farbe wie beim Ausschalten. Muss tagsüber eine Glühbirne ausgewechselt werden, so kann auch eine einzelne Ampel unabhängig von den anderen ausgeschaltet werden. Beim Einschalten übernimmt diese wieder die Farbe der ihr gegenüberliegenden Ampel und läuft synchron zu dieser. Für das Ein- und Ausschalten einzelner Ampeln existieren ebenfalls spezielle Wartungssignale der Betriebszentrale.

Abbildung B.1: Aufgaben zu Fernseher und Ampelkreuzung

UNIVERSITÄT DORTMUND

Fachbereich Informatik
Lehrstuhl für Software-Technologie

**Aufgabe 2 (Waschmaschine)**

Die für ingenieurmäßige Spitzenleistungen bekannte Firma LS10 Haushaltsgeräte bringt in Kürze ihre neue Waschmaschine auf den Markt. Die Hardware des Gerätes funktioniert bereits und ist in der Lage, über eine Reihe von Ein-/Ausgabesignalen mit ihrer Umwelt zu kommunizieren. Ihre Aufgabe besteht darin, eine passende eingebettete Steuerung mit Hilfe eines Zustandsdiagramms zu entwerfen.

Das gewünschte Verhalten der Maschine sieht wie folgt aus: Nach dem Einschalten beginnt die Maschine den Hauptwaschgang, der insgesamt 30 Minuten dauert. Zu Beginn des Hauptwaschgangs wird für zwei Minuten die Wasserzufuhr eingeschaltet, während des Hauptwaschgangs läuft der Motor mit geringer Geschwindigkeit. Auf den Hauptwaschgang folgt das Schleudern, das insgesamt 10 Minuten dauert, von denen die letzten 2 Minuten zum Abpumpen des Wassers genutzt werden. Am Ende des gesamten Programms wird ein akustischer Alarm ausgelöst. Die Maschine kann nur gestartet werden, wenn die Luke geschlossen ist, und während eines laufenden Programms wird die Luke verriegelt, so daß sie nicht geöffnet werden kann. Aus Sicherheitsgründen darf nicht geschleudert werden, wenn die Maschine überladen ist. In diesem Fall wird bereits in den letzten zwei Minuten des Hauptwaschgangs abgepumpt. Die Maschine kann jederzeit während des Programms durch erneutes Drücken des Netzschalters abgeschaltet werden. Wird sie anschließend wieder eingeschaltet, nimmt sie das Programm an der alten Position auf. Die Maschine wird im geschlossenen Zustand geliefert.

Die Hardware der Maschine liefert ihnen potentiell folgende Eingabesignale:

`DOOR_CLOSED` ..... Tür wurde geschlossen
`DOOR_OPENED` ..... Tür wurde geöffnet
`POWER` ..... Netzschalter wurde betätigt

Sie können die Hardware mit folgenden Ausgabesignalen kontrollieren:

`MOTOR_OFF` ..... Motor ausschalten
`MOTOR_SLOW` ..... Motor auf niedrige Umdrehungszahl
`MOTOR_FAST` ..... Motor auf hohe Umdrehungszahl
`WATER_OFF` ..... Wasserzufuhr ausschalten
`WATER_ON` ..... Wasserzufuhr einschalten
`PUMP_OFF` ..... Pumpe ausschalten
`PUMP_ON` ..... Pumpe anschalten
`DOOR_LOCK` ..... Tür verriegeln
`DOOR_UNLOCK` ..... Tür freigeben
`BEEP` ..... Akustisches Signal auslösen

Der interne Gewichtssensor setzt die Überwachungsbedingung `LOAD_HEAVY` auf `true`, wenn die Maschine überladen ist, z.B. wenn sie zweimal beladen wurde.

Zur Simulation der Maschine mit DAVE können Sie entweder die einfache Steuerung oder die anschaulichere Variante aus der Vorlesung verwenden. Letztere erhalten Sie, indem Sie im Inspektor den Wert `org.musoft.statemachine.samples.WashingMachine` für das Attribut `Simulator` des Modells eintragen. Achten Sie bei der Verwendung der anschaulichen Steuerung darauf, dass Sie die vorgegebenen Namen der Signale exakt einhalten. Anderenfalls funktioniert das Zusammenspiel von Steuerung und Modell nicht korrekt.

Abbildung B.2: Aufgabe zur Waschmaschine

UNIVERSITÄT DORTMUND

Fachbereich Informatik
Lehrstuhl für Software-Technologie

**Aufgabe 4 (Kaffeemaschine):**

Im Zuge ihrer globalen Expansionsstrategie entwickelt die Firma LS10 Haushaltsgeräte derzeit eine neuartige Kaffeemaschine für den internationalen Markt. Wie bei der Ihnen bereits bekannten Waschmaschine existiert bereits eine Hardware, aber keine Steuerungslogik.

Das gewünschte Verhalten der Maschine sieht wie folgt aus: Die Maschine wird – wie üblich – mit Wasser und Kaffeepulver betankt, bevor sie über den Netzschalter in Betrieb gesetzt wird. Der Brühvorgang, für den wesentlich die interne Heizvorrichtung der Maschine verantwortlich zeichnet, dauert für eine komplette Kanne maximal sechs Minuten. Das Ende des Brühvorgangs wird durch ein akustisches Signal mitgeteilt. Anschließend wird der fertige Kaffee für eine Weile warm gehalten. Eine zweite Taste erlaubt während des Brühens die Wahl zwischen dem Normalmodus und dem Energiesparmodus. Der Modus kontrolliert die Zeitspanne, für die eine fertige Kanne Kaffee warm gehalten wird: Im Normalmodus sind es 30 Minuten, im Energiesparmodus nur 10 Minuten. Der Normalmodus ist die Werkseinstellung, und die Maschine merkt sich die gewählte Einstellung über einen Zubereitungsvorgang hinaus. Ein manuelles Abschalten über den Netzschalter ist jederzeit möglich. Die Maschine darf keinesfalls länger als 40 Minuten am Stück in Betrieb sein, weil sie dann überhitzt. Wird sie nicht mit den beiden nötigen Zutaten befüllt, versetzt sie das Einschalten per Netzschalter direkt in den Warmhaltemodus. Eine rote Lampe signalisiert den laufenden Betrieb, eine gelbe das „Energiespar"-Programm. Damit ein zweiter Brühvorgang möglich wird, muß die Maschine zuvor gereinigt werden.

Entwerfen Sie eine Steuerungslogik für diese Maschine.


Die Hardware der Maschine liefert ihnen potentiell folgende Eingabesignale:

| | |
|---|---|
| `COFFEE` ..... | Kaffee wurde eingefüllt |
| `WATER` ..... | Wassertank wurde gefüllt |
| `POWER` ..... | Netzschalter wurde betätigt |
| `MODE` ..... | Modustaste wurde betätigt |
| `CLEAN` ..... | Maschine wurde entleert und gereinigt |

Sie können die Hardware mit folgenden Ausgabesignalen kontrollieren:

| | |
|---|---|
| `HEATING_OFF` ... | Heizung abschalten |
| `HEATING_ON` .... | Heizung einschalten |
| `RED_OFF` ..... | Rote Lampe abschalten |
| `RED_ON` ..... | Rote Lampe einschalten |
| `YELLOW_OFF` .... | Gelbe Lampe abschalten |
| `YELLOW_ON` ..... | Gelbe Lample einschalten |
| `BEEP` ..... | Akustischen Alarm auslösen |

Zur Simulation der Maschine mit DAVE können Sie entweder die einfache Steuerung oder die anschaulichere Variante aus der Vorlesung verwenden. Letztere erhalten Sie, indem Sie im Inspektor den Wert `org.musoft.statemachine.samples.CoffeeMaker` für das Attribut `Simulator` des Modells eintragen. Achten Sie bei der Verwendung der anschaulichen Steuerung darauf, dass Sie die vorgegebenen Namen der Signale exakt einhalten. Anderenfalls funktioniert das Zusammenspiel von Steuerung und Modell nicht korrekt.

Abbildung B.3: Aufgabe zur Kaffeemaschine

# C Fragebogen der ersten Evaluation

UNIVERSITÄT DORTMUND

Fachbereich Informatik
Lehrstuhl für Software-Technologie

## Fragebogen für die Teilnehmer der SWT-Vorlesung SS 2003

Liebe Studierende,

Sie arbeiten mit dem Editor DAVE, der im Rahmen des Projekts MuSofT (Multimedia in der SoftwareTechnik) für Sie entwickelt worden ist.

Da es sich um ein neues Produkt handelt, ist es für uns wichtig zu erfahren, wie Sie mit dem Editor zurecht kommen und welche Verbesserungen Sie sich wünschen. Die Ergebnisse dieser Befragung werden für die Weiterentwicklung von DAVE, zur Qualitätssicherung der Lehre und zu Forschungszwecken genutzt.

Des Weiteren möchten wir über DAVE hinaus Einiges darüber erfahren, wie Sie in Ihrem Studium den Computer und E-Learning-Angebote nutzen. Die Ergebnisse werden in eine Dissertation über die Nutzung von Neuen Medien im Hochschulbereich eingehen, die eine der Evaluatorinnen zur Zeit verfasst.

Wir bitten Sie auf den folgenden Seiten um eine ausführliche Rückmeldung.

Bitte geben Sie jeweils diejenige(n) Antwort(en) an, die für Sie am ehesten zutreffen.

Das Ausfüllen des Fragebogens dauert etwa 30 Minuten.

Durchgeführt wird diese Befragung von dem Hochschuldidaktischen Zentrum (HDZ) in Kooperation mit dem Lehrstuhl 10 des Fachbereichs Informatik der Universität Dortmund.

Wir garantieren Ihnen, dass Ihre Angaben anonym und vertraulich ausgewertet werden.

Danke für Ihre Mitarbeit!

Prof. Dr. Sigrid Metz-Göckel
Prof. Dr. Ernst-Erich Doberkat

Ansprechpartner/in: Hochschuldidaktisches Zentrum der Universität Dortmund,
Anja Tigges, atigges@hdz.uni-dortmund.de

1

## Fragen zu DAVE

**6. Wie haben Sie vor DAVE Übungsaufgaben bearbeitet?** (Mehrfachnennungen möglich)

- ❑ mit Stift und Zettel
- ❑ mit dem Computer und zwar mit der Software_____

**7. Haben Sie vor DAVE bereits mit einem grafischen Editor gearbeitet?** (Mehrfachnennungen möglich)

- ❑ ja
  - ❑ einmal
  - ❑ mehrmals
  - ❑ an der Uni
  - ❑ privat
- ❑ nein (weiter mit Frage 8)

**7a. Wenn ja, um welchen(n) Editor(en) handelte es sich?** _____

**8. Bitte bewerten Sie folgende Aussagen, die sich auf die Vorstellung des Editors in der SWT-Vorlesung beziehen:**

| | stimme nicht zu | stimme weniger zu | stimme eher zu | stimme voll zu | weiß nicht |
|---|---|---|---|---|---|
| Nach der Vorstellung freute ich mich darauf, mit dem Editor zu arbeiten. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Mir wurde der Sinn/ Mehrwert des Editors deutlich. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Vermittlung der Lehr-/Lerninhalte erfolgte in einem für mich angemessenen Tempo. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich fühle mich von den Lehrenden wahrgenommen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Der Vortrag hat viele Fragen offen gelassen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Der Vortrag war zu lang, da ich mich bereits mit Editoren auskannte. | ❑ | ❑ | ❑ | ❑ | ❑ |

**10. Welche Erwartungen an DAVE hatten Sie, *bevor* Sie das erste Mal mit dem Editor arbeiteten?** (Mehrfachnennungen möglich)

- ❑ Arbeitserleichterung
- ❑ Spaß
- ❑ technische Probleme
- ❑ größerer Arbeitsaufwand
- ❑ nichts besonderes
- ❑ _____

**11. Etwa eine Woche nach der Vorstellung des Editors in der Vorlesung wurde eine zweite Version des Editors ins Netz gestellt.**
**Mit welcher Editor-Version haben Sie gearbeitet?** (Mehrfachnennungen möglich)

- ❑ erste Version
- ❑ zweite Version
- ❑ Ich wusste nicht, dass es eine zweite Version gab.

**12. Auf welche Editor-Version beziehen Sie im Folgenden Ihre Anmerkungen?**

- ❑ erste Version
- ❑ zweite Version
- ❑ beide Versionen

**13. Wie verlief die Installation von DAVE?**

- ❑ ohne Probleme (weiter mit Frage 14)
- ❑ machte Probleme
- ❑ klappte gar nicht

**13a. Wenn es Probleme gab, beschreiben Sie diese bitte.**

✍_____

_____

**14. Mussten Sie zu Hause eigens für DAVE eine Java-Umgebung installieren?**
(Mehrfachnennungen möglich)

- ❑ ja
  - ❑ das lief problemlos
  - ❑ aber ich hatte dabei Probleme
- ❑ nein

**15. *Wie* und *wo* haben Sie mit dem Editor gearbeitet?**
(Mehrfachnennungen möglich)

- ❑ immer in einer Gruppe
- ❑ teils in einer Gruppe, teils alleine
- ❑ immer alleine (weiter mit Frage 17)
- ❑ immer zu Hause
- ❑ teils zuhause, teils im Computer-Pool
- ❑ immer im Computer-Pool
- ❑ _____

**17. Wie lange haben Sie insgesamt mit dem Editor gearbeitet?**
(Mehrfachnennungen möglich)

- ❑ unter 5 Stunden
- ❑ 5 bis 10 Stunden
- ❑ 11-20 Stunden
- ❑ über 20 Stunden

**18. In welchem Umfang haben Sie....**

| | nie | einmal | mehrmals | bei jeder Aufgabe |
|---|---|---|---|---|
| die Simulationsfunktion genutzt? | ❑ | ❑ | ❑ | ❑ |
| die Veranschaulichung der Simulation durch Geräte (Kaffeemaschine, Waschmaschine) genutzt? | ❑ | ❑ | ❑ | ❑ |
| ausgedruckt? | ❑ | ❑ | ❑ | ❑ |
| exportiert? | ❑ | ❑ | ❑ | ❑ |
| den Navigator genutzt? | ❑ | ❑ | ❑ | ❑ |
| die Schnellansicht genutzt? | ❑ | ❑ | ❑ | ❑ |
| die Notizfunktion genutzt? | ❑ | ❑ | ❑ | ❑ |
| die Hilfe aufgerufen? | ❑ | ❑ | ❑ | ❑ |
| das Look ´n´ Feel verändert? | ❑ | ❑ | ❑ | ❑ |
| die Farbe der Modellelemente geändert? | ❑ | ❑ | ❑ | ❑ |
| die DAVE-Vorlesungsfolien zu Rate gezogen? | ❑ | ❑ | ❑ | ❑ |
| die Homepage konsultiert? | ❑ | ❑ | ❑ | ❑ |

**19. Bitte bewerten Sie die Simulationsfunktion des Editors:**

| | stimme nicht zu | stimme weniger zu | stimme eher zu | stimme voll zu | weiß nicht |
|---|---|---|---|---|---|
| Die Möglichkeit, meine Modelle zu simulieren, hat mich beim Lösen der Aufgaben unterstützt. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich fand es gut, ein unmittelbares Feedback zu meinem Modell zu bekommen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Durch die Simulation habe ich ein besseres Verständnis der Statecharts gewonnen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Simulation verleitet dazu, die Aufgaben durch „Trial and Error“ zu lösen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich halte die Simulation für überflüssig, weil ich von meinem Übungsgruppenleiter ein Feedback bekomme. | ❑ | ❑ | ❑ | ❑ | ❑ |

**20. Bitte bewerten Sie die Veranschaulichung der Simulation durch Geräte (Kaffeemaschine und Waschmaschine):**

| | stimme nicht zu | stimme weniger zu | stimme eher zu | stimme voll zu | weiß nicht |
|---|---|---|---|---|---|
| Ich habe mich durch die beiden Beispiele angesprochen gefühlt. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Durch die Beispiele wurden die abstrakten Probleme anschaulicher. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Durch die Beispiele wurde mir der praktische Nutzen des Formalismus´ deutlich. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Mir war nicht klar, wie ich mit meinem Modell das Gerät ansteuern kann. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Beispiele waren nicht hilfreich. | ❑ | ❑ | ❑ | ❑ | ❑ |

**21. Hatten Sie im Zusammenhang mit DAVE technische Probleme?** (Mehrfachnennungen möglich)

❑ ja
- ❑ einmal
- ❑ mehrmals

❑ nein (weiter bei Frage 25)

**21a. Welches Problem/ welche Probleme hatten Sie?**

✍_____
_____

**22. Haben Sie sich, um das Problem zu beheben, Hilfe geholt?**

❑ ja (weiter mit Frage 22b)
❑ nein

**22a. Wenn Sie sich keine Hilfe geholt haben, wieso nicht?**
(Mehrfachnennungen möglich)

- ❑ Ich konnte das Problem alleine lösen
- ❑ Ich halte die universitären Ansprechpartner nicht für fähig, mir bei technischen Problemen zu helfen.
- ❑ Ich hielt das Problem nicht für relevant genug.
- ❑ Ich befürchtete, mich als nicht kompetent bloßzustellen.
- ❑ _____

(weiter mit Frage 24)

**22b. Wenn Sie sich Hilfe geholt haben: An *wen* haben Sie sich *wie* gewandt?**
(Mehrfachnennungen möglich)

| | Telefon | Email | face to face | ✍_____ |
|---|---|---|---|---|
| Kommilitonin(nen) | ❑ | ❑ | ❑ | ❑ |
| Kommilitone(n) | ❑ | ❑ | ❑ | ❑ |
| Übungsgruppenleiter | ❑ | ❑ | ❑ | ❑ |
| Entwickler | ❑ | ❑ | ❑ | ❑ |
| Dozierende | ❑ | ❑ | ❑ | ❑ |
| ✍_____ | ❑ | ❑ | ❑ | ❑ |

**23. Konnten Sie mit der Unterstützung durch die o.g. Person(en) das Problem/die Probleme lösen, das/ die beim Arbeiten mit dem Editor aufgetreten ist/sind?**

- ❑ ja (weiter mit Frage 24)
- ❑ teilweise
- ❑ nein

**23a. Wenn nein, wieso konnte(n) Ihr(e) Problem(e) nicht gelöst werden?**
✍_____
_____

**24. Wurde(n) Ihr(e) Problem(e) durch die 2. Editor-Version behoben?**

- ❑ ja
- ❑ nein
- ❑ welche 2. Version?

**25. Gab es Gründe, weshalb Sie Sich bei der Nutzung des Editors beeinträchtigt gefühlt haben?**
(Mehrfachnennungen möglich)

- ❑ ja
  - ❑ fehlende Hard- oder Software zuhause, nämlich _____
  - ❑ technische Probleme am Computer zu Hause
  - ❑ technische Probleme in den Computer-Pools
  - ❑ fehlende Zeit für Einarbeitung in die Funktionen des Editors
  - ❑ Unausgereiftheit des Editors
  - ❑ _____
- ❑ nein
- ❑ weiß nicht

5

**26. Bewerten Sie bitte folgende Fragen mit Zustimmung oder Ablehnung**

| | stimme zu | stimme nicht zu |
|---|---|---|
| Die Oberflächengestaltung des Editors war ansprechend. | ❑ | ❑ |
| Ich würde DAVE meinen KommilitonInnen weiterempfehlen. | ❑ | ❑ |
| Ich fände ein ähnliches Werkzeug auch für andere Themenbereiche der Vorlesung (z.B. Petri-Netze) sinnvoll. | ❑ | ❑ |
| Die Ausführungsgeschwindigkeit von DAVE war angenehm. | ❑ | ❑ |
| Ich habe DAVE über die Übungsaufgaben hinaus auch privat genutzt. | ❑ | ❑ |

**27. Wie schätzen Sie den Nutzen von DAVE beim Lösen Ihrer Arbeitsaufgaben ein?**

❑ sehr gut
❑ gut
❑ zufriedenstellend
❑ unbefriedigend

**28. Worin sehen Sie *Vorteile* des Editors gegenüber dem Arbeiten mit Stift und Papier?**
(Mehrfachnennungen möglich)

❑ Arbeitserleichterung
❑ Spaß
❑ bessere optische Gestaltung
❑ Simulation gibt Feedback
❑ werkzeugorientiertes Arbeiten wie im professionellen Umfeld
❑ Motivation
❑ besseres Verständnis der Inhalte
❑ es gibt keinen Vorteil
❑ _____

**28a. Warum sehen Sie diesen Punkt/ diese Punkte als Vorteil?**

✎_____
_____

**29. Welche *Nachteile* des Editors gegenüber dem Arbeiten mit Stift und Papier sehen Sie?**
(Mehrfachnennungen möglich)

❑ größerer Arbeitsaufwand
❑ technische Probleme
❑ Verleitung zur Ästhetisierung der Diagramme
❑ Ergebnis muss zusätzlich gemailt werden
❑ Arbeit am Computer behindert die Gruppenarbeit
❑ es gibt keinen Nachteil
❑ _____

**29a. Warum sehen Sie diesen Punkt/diese Punkte als Nachteil?**

✎_____
_____

**30. Was kann an DAVE Ihrer Meinung nach verbessert werden?**

✎_____
_____

**31. Welche weiteren Funktionen wünschen Sie sich?**

✍_____
_____

**32. In die Hilfe aufgenommen werden sollte:**

✍_____
_____

## Fragen zu Computern

**39. Besitzen Sie zur Zeit einen eigenen Computer?**

❑ ja
- _____ Jahre alt
- _____ MHz Prozessor
- _____ MB Hauptspeichergröße
- _____ GB Festplattengröße

❑ nein

**41. Welches Betriebsystem verwenden Sie?**
(Mehrfachnennungen möglich)

❑ Windows
❑ Linux
❑ MacOS
❑ Solaris
❑ _____

**42. Haben Sie zu Hause einen Internetzugang?**

❑ ja
- ❑ Modem
- ❑ ISDN
- ❑ DSL
- ❑ _____

❑ nein

## Angaben zur Person

**57. Geschlecht:** ❑ weiblich ❑ männlich

**58. Studiengang:** _____

**59. angestrebter Abschluss:** _____

**61. Fachsemester:** _____

**Vielen Dank für Ihre Mitarbeit!**

7

# D Ergebnisse der ersten Evaluation

**Frage 6: Wie haben Sie vor DAVE Übungsaufgaben bearbeitet?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Mit Stift und Zettel | 10 | 90.9% | 46 | 67.6% | 56 | 70.9% |
| Mit dem Computer und der Software... | 0 | 0.0% | 34 | 50.0% | 34 | 43.0% |

**Frage 7: Haben Sie vor DAVE bereits mit einem graphischen Editor gearbeitet?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 6 | 54.5% | 56 | 82.4% | 62 | 78.5% |
| Einmal | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Mehrmals | 2 | 18.2% | 41 | 60.3% | 43 | 54.4% |
| An der Uni | 7 | 63.6% | 36 | 52.9% | 43 | 54.4% |
| Privat | 3 | 27.3% | 31 | 45.6% | 34 | 43.0% |
| Nein | 4 | 36.4% | 11 | 16.2% | 15 | 19.0% |

**Frage 7a: Wenn Sie bereits vor DAVE mit einem graphischen Editor gearbeitet haben, um welche(n) Editor(en) handelt es sich?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 6 | 54.5% | 53 | 77.9% | 59 | 74.7% |
| Nicht genannt | 5 | 45.5% | 15 | 22.1% | 20 | 25.3% |

**Frage 8a: Nach der Vorlesung freute ich mich darauf, mit dem Editor zu arbeiten**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 18.2% | 7 | 10.3% | 9 | 11.4% |
| Stimme weniger zu | 1 | 9.1% | 10 | 14.7% | 11 | 13.9% |
| Stimme eher zu | 3 | 27.3% | 28 | 41.2% | 31 | 39.2% |
| Stimme voll zu | 3 | 27.3% | 15 | 22.1% | 18 | 22.8% |
| Weiß nicht | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |
| Fehlend / ungültig | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |

**Frage 8b: Mir wurde der Sinn/Mehrwert des Editors deutlich**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 9.1% | 6 | 8.8% | 7 | 8.9% |
| Stimme weniger zu | 1 | 9.1% | 7 | 10.3% | 8 | 10.1% |
| Stimme eher zu | 5 | 45.5% | 31 | 45.6% | 36 | 45.6% |
| Stimme voll zu | 3 | 27.3% | 18 | 26.5% | 21 | 26.6% |
| Weiß nicht | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| Fehlend / ungültig | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |

**Frage 8c: Die Vermittlung der Lehr-/Lerninhalte erfolgte in einem für mich angemessenen Tempo**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| Stimme weniger zu | 1 | 9.1% | 11 | 16.2% | 12 | 15.2% |
| Stimme eher zu | 1 | 9.1% | 26 | 38.2% | 27 | 34.2% |
| Stimme voll zu | 7 | 63.6% | 24 | 35.3% | 31 | 39.2% |
| Weiß nicht | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |
| Fehlend / ungültig | 1 | 9.1% | 4 | 5.9% | 5 | 6.3% |

**Frage 8d: Ich fühlte mich von den Lehrenden wahrgenommen**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 9.1% | 4 | 5.9% | 5 | 6.3% |
| Stimme weniger zu | 1 | 9.1% | 16 | 23.5% | 17 | 21.5% |
| Stimme eher zu | 3 | 27.3% | 26 | 38.2% | 29 | 36.7% |
| Stimme voll zu | 5 | 45.5% | 11 | 16.2% | 16 | 20.3% |
| Weiß nicht | 0 | 0.0% | 6 | 8.8% | 6 | 7.6% |
| Fehlend / ungültig | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |

**Frage 8e: Der Vortrag hat viele Fragen offen gelassen**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 5 | 45.5% | 16 | 23.5% | 21 | 26.6% |
| Stimme weniger zu | 3 | 27.3% | 31 | 45.6% | 34 | 43.0% |
| Stimme eher zu | 0 | 0.0% | 12 | 17.6% | 12 | 15.2% |
| Stimme voll zu | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| Weiß nicht | 2 | 18.2% | 3 | 4.4% | 5 | 6.3% |
| Fehlend / ungültig | 1 | 9.1% | 4 | 5.9% | 5 | 6.3% |

**Frage 8f: Der Vortrag war zu lang, da ich mich bereits mit Editoren auskenne**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 27.3% | 21 | 30.9% | 24 | 30.4% |
| Stimme weniger zu | 4 | 36.4% | 22 | 32.4% | 26 | 32.9% |
| Stimme eher zu | 1 | 9.1% | 12 | 17.6% | 13 | 16.5% |
| Stimme voll zu | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |
| Weiß nicht | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Fehlend / ungültig | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |

**Frage 9: Wie empfanden Sie die Atmosphäre während der Vorstellung des Editors?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Angenehm | 7 | 63.6% | 28 | 41.2% | 35 | 44.3% |
| Produktiv | 6 | 54.5% | 11 | 16.2% | 17 | 21.5% |
| Neutral | 2 | 18.2% | 28 | 41.2% | 30 | 38.0% |
| Unruhig | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Angespannt | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| Weiß nicht | 1 | 9.1% | 10 | 14.7% | 11 | 13.9% |
| Freie Antwort | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |

**Frage 9a: Inwiefern könnte der Vortragende während der Vorstellung des Editors zu einer besseren Atmosphäre beitragen?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 5 | 45.5% | 33 | 48.5% | 38 | 48.1% |
| Nicht genannt | 6 | 54.5% | 35 | 51.5% | 41 | 51.9% |

**Frage 9b: Wie könnten die Studierenden die Atmosphäre beeinflussen?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 6 | 54.5% | 34 | 50.0% | 40 | 50.6% |
| Nicht genannt | 5 | 45.5% | 34 | 50.0% | 39 | 49.4% |

**Frage 10: Welche Erwartungen an DAVE hatten Sie, bevor Sie das erste Mal mit dem Editor arbeiteten?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Arbeitserleichterung | 4 | 36.4% | 45 | 66.2% | 49 | 62.0% |
| Spaß | 4 | 36.4% | 23 | 33.8% | 27 | 34.2% |
| Technische Probleme | 4 | 36.4% | 19 | 27.9% | 23 | 29.1% |
| Größerer Arbeitsaufwand | 3 | 27.3% | 8 | 11.8% | 11 | 13.9% |
| Nichts Besonderes | 2 | 18.2% | 10 | 14.7% | 12 | 15.2% |
| Freie Antwort | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |

**Frage 11: Mit welcher Editor-Version haben Sie gearbeitet?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Erste Version | 6 | 54.5% | 38 | 55.9% | 44 | 55.7% |
| Zweite Version | 4 | 36.4% | 34 | 50.0% | 38 | 48.1% |
| Wusste nicht, dass es eine zweite Version gibt | 5 | 45.5% | 18 | 26.5% | 23 | 29.1% |

**Frage 12: Auf welche Editor-Version beziehen Sie im folgenden Ihre Antworten?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Erste Version | 7 | 63.6% | 33 | 48.5% | 40 | 50.6% |
| Zweite Version | 3 | 27.3% | 14 | 20.6% | 17 | 21.5% |
| Beide Versionen | 1 | 9.1% | 19 | 27.9% | 20 | 25.3% |
| Fehlend / ungültig | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |

**Frage 13: Wie verlief die Installation von DAVE?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ohne Probleme | 9 | 81.8% | 59 | 86.8% | 68 | 86.1% |
| Machte Probleme | 2 | 18.2% | 8 | 11.8% | 10 | 12.7% |
| Klappte gar nicht | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 13a: Wenn es Probleme gab, beschreiben Sie diese bitte.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 2 | 18.2% | 8 | 11.8% | 10 | 12.7% |
| Nicht genannt | 9 | 81.8% | 60 | 88.2% | 69 | 87.3% |

**Frage 14: Mussten Sie zuhause eigens für DAVE eine Java-Umgebung installieren?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 5 | 45.5% | 22 | 32.4% | 27 | 34.2% |
| Das lief problemlos | 5 | 45.5% | 17 | 25.0% | 22 | 27.8% |
| Ich hatte dabei Probleme | 1 | 9.1% | 4 | 5.9% | 5 | 6.3% |
| Nein | 5 | 45.5% | 45 | 66.2% | 50 | 63.3% |

**Frage 15: Wie und wo haben Sie mit dem Editor gearbeitet?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Immer in der Gruppe | 2 | 18.2% | 7 | 10.3% | 9 | 11.4% |
| Teils in der Gruppe, teils allein | 8 | 72.7% | 22 | 32.4% | 30 | 38.0% |
| Immer alleine | 1 | 9.1% | 36 | 52.9% | 37 | 46.8% |
| Immer zuhause | 5 | 45.5% | 44 | 64.7% | 49 | 62.0% |
| Teils zuhause, teils im Computer-Pool | 0 | 0.0% | 9 | 13.2% | 9 | 11.4% |
| Immer im Computer-Pool | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| Freie Antwort | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 16: Wenn Sie in der Gruppe gearbeitet haben, wer hatte dann in der Regel die Hand an der Maus?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ich | 3 | 27.3% | 7 | 10.3% | 10 | 12.7% |
| Meine Teampartnerin | 1 | 9.1% | 0 | 0.0% | 1 | 1.3% |
| Mein Teampartner | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Abwechselnd | 5 | 45.5% | 24 | 35.3% | 29 | 36.7% |

**Frage 16a: Waren Sie mit der Arbeitsteilung am Computer einverstanden?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 10 | 90.9% | 36 | 52.9% | 46 | 58.2% |
| Nein | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| Fehlend / ungültig | 1 | 9.1% | 31 | 45.6% | 32 | 40.5% |

**Frage 16b: Begründen Sie bitte Ihre letzte Antwort.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 9 | 81.8% | 25 | 36.8% | 34 | 43.0% |
| Nicht genannt | 2 | 18.2% | 40 | 58.8% | 42 | 53.2% |

**Frage 17: Wie lange haben Sie insgesamt mit dem Editor gearbeitet?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Unter 5 Stunden | 7 | 63.6% | 27 | 39.7% | 34 | 43.0% |
| 5 – 10 Stunden | 3 | 27.3% | 27 | 39.7% | 30 | 38.0% |
| 11 – 20 Stunden | 1 | 9.1% | 11 | 16.2% | 12 | 15.2% |
| über 20 Stunden | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |

**Frage 18a: In welchem Umfang haben Sie die Simulationsfunktion genutzt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Einmal | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| Mehrmals | 2 | 18.2% | 14 | 20.6% | 16 | 20.3% |
| Bei jeder Aufgabe | 8 | 72.7% | 49 | 72.1% | 57 | 72.2% |
| Fehlend / ungültig | 1 | 9.1% | 2 | 2.9% | 3 | 3.8% |

**Frage 18b: In welchem Umfang haben Sie die Veranschaulichung der Simulation durch Geräte genutzt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 1 | 9.1% | 7 | 10.3% | 8 | 10.1% |
| Einmal | 0 | 0.0% | 13 | 19.1% | 13 | 16.5% |
| Mehrmals | 4 | 36.4% | 22 | 32.4% | 26 | 32.9% |
| Bei jeder Aufgabe | 6 | 54.5% | 24 | 35.3% | 30 | 38.0% |
| Fehlend / ungültig | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |

**Frage 18c: In welchem Umfang haben Sie ausgedruckt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 2 | 18.2% | 22 | 32.4% | 24 | 30.4% |
| Einmal | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |
| Mehrmals | 3 | 27.3% | 14 | 20.6% | 17 | 21.5% |
| Bei jeder Aufgabe | 5 | 45.5% | 27 | 39.7% | 32 | 40.5% |

**Frage 18d: In welchem Umfang haben Sie exportiert?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 5 | 45.5% | 22 | 32.4% | 27 | 34.2% |
| Einmal | 1 | 9.1% | 9 | 13.2% | 10 | 12.7% |
| Mehrmals | 2 | 18.2% | 16 | 23.5% | 18 | 22.8% |
| Bei jeder Aufgabe | 2 | 18.2% | 20 | 29.4% | 22 | 27.8% |
| Fehlend / ungültig | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |

**Frage 18e: In welchem Umfang haben Sie den Navigator genutzt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 3 | 27.3% | 19 | 27.9% | 22 | 27.8% |
| Einmal | 2 | 18.2% | 9 | 13.2% | 11 | 13.9% |
| Mehrmals | 2 | 18.2% | 32 | 47.1% | 34 | 43.0% |
| Bei jeder Aufgabe | 2 | 18.2% | 8 | 11.8% | 10 | 12.7% |
| Fehlend / ungültig | 2 | 18.2% | 0 | 0.0% | 2 | 2.5% |

**Frage 18f: In welchem Umfang haben Sie die Schnellansicht genutzt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 6 | 54.5% | 25 | 36.8% | 31 | 39.2% |
| Einmal | 2 | 18.2% | 8 | 11.8% | 10 | 12.7% |
| Mehrmals | 0 | 0.0% | 29 | 42.6% | 29 | 36.7% |
| Bei jeder Aufgabe | 2 | 18.2% | 3 | 4.4% | 5 | 6.3% |
| Fehlend / ungültig | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |

**Frage 18g: In welchem Umfang haben Sie die Notizfunktion genutzt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 6 | 54.5% | 24 | 35.3% | 30 | 38.0% |
| Einmal | 2 | 18.2% | 6 | 8.8% | 8 | 10.1% |
| Mehrmals | 2 | 18.2% | 22 | 32.4% | 24 | 30.4% |
| Bei jeder Aufgabe | 0 | 0.0% | 15 | 22.1% | 15 | 19.0% |
| Fehlend / ungültig | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |

**Frage 18h: In welchem Umfang haben Sie die Hilfe aufgerufen?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 7 | 63.6% | 34 | 50.0% | 41 | 51.9% |
| Einmal | 1 | 9.1% | 23 | 33.8% | 24 | 30.4% |
| Mehrmals | 3 | 27.3% | 8 | 11.8% | 11 | 13.9% |
| Bei jeder Aufgabe | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| Fehlend / ungültig | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |

**Frage 18i: In welchem Umfang haben Sie das Look and Feel verändert?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 7 | 63.6% | 38 | 55.9% | 45 | 57.0% |
| Einmal | 3 | 27.3% | 16 | 23.5% | 19 | 24.1% |
| Mehrmals | 1 | 9.1% | 12 | 17.6% | 13 | 16.5% |
| Bei jeder Aufgabe | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |

**Frage 18j: In welchem Umfang haben Sie die Farbe der Modellelemente geändert?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 2 | 18.2% | 41 | 60.3% | 43 | 54.4% |
| Einmal | 2 | 18.2% | 10 | 14.7% | 12 | 15.2% |
| Mehrmals | 6 | 54.5% | 12 | 17.6% | 18 | 22.8% |
| Bei jeder Aufgabe | 1 | 9.1% | 4 | 5.9% | 5 | 6.3% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 18k: In welchem Umfang haben Sie die DAVE-Vorlesungsfolien zu Rate gezogen?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 0 | 0.0% | 12 | 17.6% | 12 | 15.2% |
| Einmal | 5 | 45.5% | 22 | 32.4% | 27 | 34.2% |
| Mehrmals | 4 | 36.4% | 29 | 42.6% | 33 | 41.8% |
| Bei jeder Aufgabe | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |
| Fehlend / ungültig | 1 | 9.1% | 0 | 0.0% | 1 | 1.3% |

**Frage 18l: In welchem Umfang haben Sie die Homepage konsultiert?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 4 | 36.4% | 32 | 47.1% | 36 | 45.6% |
| Einmal | 2 | 18.2% | 22 | 32.4% | 24 | 30.4% |
| Mehrmals | 5 | 45.5% | 13 | 19.1% | 18 | 22.8% |
| Bei jeder Aufgabe | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 19a: Die Möglichkeit, meine Modelle zu simulieren, hat mich beim Lösen der Aufgaben unterstützt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |
| Stimme weniger zu | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| Stimme eher zu | 1 | 9.1% | 15 | 22.1% | 16 | 20.3% |
| Stimme voll zu | 9 | 81.8% | 49 | 72.1% | 58 | 73.4% |
| Weiß nicht | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |

**Frage 19b: Ich fand es gut, ein unmittelbares Feedback zu meinem Modell zu bekommen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 9.1% | 0 | 0.0% | 1 | 1.3% |
| Stimme weniger zu | 1 | 9.1% | 4 | 5.9% | 5 | 6.3% |
| Stimme eher zu | 0 | 0.0% | 6 | 8.8% | 6 | 7.6% |
| Stimme voll zu | 9 | 81.8% | 56 | 82.4% | 65 | 82.3% |
| Weiß nicht | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |

**Frage 19c: Durch die Simulation habe ich ein besseres Verständnis der Statecharts gewonnen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Stimme weniger zu | 1 | 9.1% | 9 | 13.2% | 10 | 12.7% |
| Stimme eher zu | 2 | 18.2% | 28 | 41.2% | 30 | 38.0% |
| Stimme voll zu | 7 | 63.6% | 27 | 39.7% | 34 | 43.0% |
| Weiß nicht | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 19d: Die Simulation verleitet dazu, die Aufgaben per Trial and Error zu lösen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 27.3% | 5 | 7.4% | 8 | 10.1% |
| Stimme weniger zu | 2 | 18.2% | 27 | 39.7% | 29 | 36.7% |
| Stimme eher zu | 2 | 18.2% | 17 | 25.0% | 19 | 24.1% |
| Stimme voll zu | 4 | 36.4% | 18 | 26.5% | 22 | 27.8% |
| Weiß nicht | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 19e: Ich halte die Simulation für überflüssig, weil ich von meinem Übungsgruppenleiter ein Feedback bekomme.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 9 | 81.8% | 53 | 77.9% | 62 | 78.5% |
| Stimme weniger zu | 2 | 18.2% | 14 | 20.6% | 16 | 20.3% |
| Stimme eher zu | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Stimme voll zu | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Weiß nicht | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 20a: Ich habe mich durch die beiden Beispiele angesprochen gefühlt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| Stimme weniger zu | 0 | 0.0% | 13 | 19.1% | 13 | 16.5% |
| Stimme eher zu | 3 | 27.3% | 23 | 33.8% | 26 | 32.9% |
| Stimme voll zu | 8 | 72.7% | 25 | 36.8% | 33 | 41.8% |
| Weiß nicht | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 20b: Durch die Beispiele wurden die abstrakten Probleme deutlicher.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 9.1% | 8 | 11.8% | 9 | 11.4% |
| Stimme weniger zu | 2 | 18.2% | 13 | 19.1% | 15 | 19.0% |
| Stimme eher zu | 1 | 9.1% | 23 | 33.8% | 24 | 30.4% |
| Stimme voll zu | 6 | 54.5% | 20 | 29.4% | 26 | 32.9% |
| Weiß nicht | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 20c: Durch die Beispiele wurde mir der praktische Nutzen des Formalismus deutlich.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 18.2% | 11 | 16.2% | 13 | 16.5% |
| Stimme weniger zu | 3 | 27.3% | 20 | 29.4% | 23 | 29.1% |
| Stimme eher zu | 3 | 27.3% | 19 | 27.9% | 22 | 27.8% |
| Stimme voll zu | 3 | 27.3% | 15 | 22.1% | 18 | 22.8% |
| Weiß nicht | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 20d: Mir war nicht klar, wie ich mit meinem Modell das Gerät ansteuern kann.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 9 | 81.8% | 41 | 60.3% | 50 | 63.3% |
| Stimme weniger zu | 2 | 18.2% | 15 | 22.1% | 17 | 21.5% |
| Stimme eher zu | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| Stimme voll zu | 0 | 0.0% | 6 | 8.8% | 6 | 7.6% |
| Weiß nicht | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 20e: Die Beispiele waren nicht hilfreich.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 10 | 90.9% | 38 | 55.9% | 48 | 60.8% |
| Stimme weniger zu | 1 | 9.1% | 13 | 19.1% | 14 | 17.7% |
| Stimme eher zu | 0 | 0.0% | 9 | 13.2% | 9 | 11.4% |
| Stimme voll zu | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| Weiß nicht | 0 | 0.0% | 5 | 7.4% | 5 | 6.3% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 21: Hatten Sie im Zusammenhang mit DAVE technische Probleme?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 5 | 45.5% | 37 | 54.4% | 42 | 53.2% |
| Einmal | 4 | 36.4% | 8 | 11.8% | 12 | 15.2% |
| Mehrmals | 1 | 9.1% | 24 | 35.3% | 25 | 31.6% |
| Nein | 6 | 54.5% | 27 | 39.7% | 33 | 41.8% |

**Frage 21a: Welche(s) Problem(e) hatten Sie?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 5 | 45.5% | 40 | 58.8% | 45 | 57.0% |
| Nicht genannt | 6 | 54.5% | 28 | 41.2% | 34 | 43.0% |

**Frage 22: Haben Sie sich, um das Problem zu lösen, Hilfe geholt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 2 | 18.2% | 14 | 20.6% | 16 | 20.3% |
| Nein | 4 | 36.4% | 30 | 44.1% | 34 | 43.0% |
| Fehlend / ungültig | 5 | 45.5% | 24 | 35.3% | 29 | 36.7% |

**Frage 22a: Wenn Sie sich keine Hilfe geholt haben, wieso nicht?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ich konnte das Problem alleine lösen. | 2 | 18.2% | 6 | 8.8% | 8 | 10.1% |
| Ich halte die universitären Ansprechpartner nicht für fähig, mir bei technischen Problemen zu helfen. | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Ich hielt das Problem nicht für relevant genug. | 2 | 18.2% | 15 | 22.1% | 17 | 21.5% |
| Ich befürchtete, mich als nicht kompetent bloßzustellen. | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Freie Antwort | 1 | 9.1% | 9 | 13.2% | 10 | 12.7% |

**Frage 22b-i: Wenn Sie sich Hilfe geholt haben, an wen haben Sie sich gewandt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Kommilitonin(nen) | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |
| Kommilitone(n) | 2 | 18.2% | 12 | 17.6% | 14 | 17.7% |
| Übungsgruppenleiter | 1 | 9.1% | 7 | 10.3% | 8 | 10.1% |
| Entwickler | 2 | 18.2% | 15 | 22.1% | 17 | 21.5% |
| Dozierende | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| Freie Antwort | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |

**Frage 22b-ii: Wenn Sie sich Hilfe geholt haben, wie haben Sie sich an die entsprechende Person gewandt?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Telefon | 0 | 0.0% | 9 | 13.2% | 9 | 11.4% |
| E-Mail | 2 | 18.2% | 13 | 19.1% | 15 | 19.0% |
| Face to face | 4 | 36.4% | 22 | 32.4% | 26 | 32.9% |
| Freie Antwort | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |

**Frage 23: Konnten Sie mit der Unterstützung der angegebenen Personen das Problem lösen?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 3 | 27.3% | 7 | 10.3% | 10 | 12.7% |
| Teilweise | 0 | 0.0% | 9 | 13.2% | 9 | 11.4% |
| Nein | 1 | 9.1% | 5 | 7.4% | 6 | 7.6% |
| Fehlend / ungültig | 7 | 63.6% | 47 | 69.1% | 54 | 68.4% |

**Frage 24: Wurde Ihr Problem durch die zweite Editor-Version behoben?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 1 | 9.1% | 8 | 11.8% | 9 | 11.4% |
| Nein | 2 | 18.2% | 15 | 22.1% | 17 | 21.5% |
| Welche zweite Version? | 3 | 27.3% | 14 | 20.6% | 17 | 21.5% |

**Frage 25: Gab es Gründe, weshalb Sie sich bei der Nutzung des Editors beeinträchtigt gefühlt haben?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 2 | 18.2% | 30 | 44.1% | 32 | 40.5% |
| Fehlende Hard- oder Software zuhause | 1 | 9.1% | 0 | 0.0% | 1 | 1.3% |
| Technische Probleme am Computer zuhause | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| Technische Probleme in den Computer-Pools | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| Fehlende Zeit zur Einarbeitung in die Funktionen des Editors | 0 | 0.0% | 5 | 7.4% | 5 | 6.3% |
| Unausgereiftheit des Editors | 1 | 9.1% | 25 | 36.8% | 26 | 32.9% |
| Freie Antwort | 0 | 0.0% | 8 | 11.8% | 8 | 10.1% |
| Nein | 9 | 81.8% | 28 | 41.2% | 37 | 46.8% |
| Weiß nicht | 0 | 0.0% | 5 | 7.4% | 5 | 6.3% |

**Frage 26a: Die Oberflächengestaltung des Editors war ansprechend.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme zu | 9 | 81.8% | 61 | 89.7% | 70 | 88.6% |
| Stimme nicht zu | 2 | 18.2% | 6 | 8.8% | 8 | 10.1% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

**Frage 26b: Ich würde DAVE meinen Kommiliton(inn)en weiterempfehlen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme zu | 7 | 63.6% | 52 | 76.5% | 59 | 74.7% |
| Stimme nicht zu | 4 | 36.4% | 16 | 23.5% | 20 | 25.3% |

**Frage 26c: Ich fände ein ähnliches Werkzeug auch für andere Bereiche der Vorlesung sinnvoll.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme zu | 9 | 81.8% | 59 | 86.8% | 68 | 86.1% |
| Stimme nicht zu | 2 | 18.2% | 9 | 13.2% | 11 | 13.9% |

**Frage 26d: Die Ausführungsgeschwindigkeit von DAVE war angenehm.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme zu | 11 | 100.0% | 51 | 75.0% | 62 | 78.5% |
| Stimme nicht zu | 0 | 0.0% | 17 | 25.0% | 17 | 21.5% |

**Frage 26e: Ich habe DAVE über die Übungsaufgaben hinaus auch privat genutzt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme zu | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |
| Stimme nicht zu | 10 | 90.9% | 67 | 98.5% | 77 | 97.5% |

**Frage 27: Wie schätzen Sie den Nutzen von DAVE beim Lösen Ihrer Übungsaufgaben ein?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr gut | 4 | 36.4% | 18 | 26.5% | 22 | 27.8% |
| Gut | 5 | 45.5% | 32 | 47.1% | 37 | 46.8% |
| Zufriedenstellend | 2 | 18.2% | 11 | 16.2% | 13 | 16.5% |
| Unbefriedigend | 0 | 0.0% | 7 | 10.3% | 7 | 8.9% |

**Frage 28: Worin sehen Sie die Vorteile des Editors gegenüber dem Arbeiten mit Stift und Papier?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Arbeitserleichterung | 4 | 36.4% | 40 | 58.8% | 44 | 55.7% |
| Spaß | 5 | 45.5% | 24 | 35.3% | 29 | 36.7% |
| Bessere optische Gestaltung | 9 | 81.8% | 45 | 66.2% | 54 | 68.4% |
| Simulation gibt Feedback | 8 | 72.7% | 63 | 92.6% | 71 | 89.9% |
| Werkzeugorientiertes Arbeiten wie im professionellen Umfeld | 3 | 27.3% | 21 | 30.9% | 24 | 30.4% |
| Motivation | 3 | 27.3% | 24 | 35.3% | 27 | 34.2% |
| Besseres Verständnis der Inhalte | 4 | 36.4% | 19 | 27.9% | 23 | 29.1% |
| Es gibt keinen Vorteil | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |
| Freie Antwort | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |

**Frage 28a: Warum sehen Sie diesen Punkt als Vorteil?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt kategorisiert | 9 | 81.8% | 47 | 69.1% | 56 | 70.9% |
| Nicht genannt | 2 | 18.2% | 21 | 30.9% | 23 | 29.1% |

**Frage 29: Welche Nachteile des Editors gegenüber dem Arbeiten mit Stift und Papier sehen Sie?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Größerer Arbeitsaufwand | 5 | 45.5% | 14 | 20.6% | 19 | 24.1% |
| Technische Probleme | 6 | 54.5% | 20 | 29.4% | 26 | 32.9% |
| Verleitet zur Ästhetisierung der Ergebnisse | 2 | 18.2% | 22 | 32.4% | 24 | 30.4% |
| Ergebnis muss zusätzlich gemailt werden | 4 | 36.4% | 6 | 8.8% | 10 | 12.7% |
| Arbeit am Computer behindert Gruppenarbeit | 3 | 27.3% | 4 | 5.9% | 7 | 8.9% |
| Es gibt keinen Nachteil | 0 | 0.0% | 18 | 26.5% | 18 | 22.8% |
| Freie Antwort | 2 | 18.2% | 7 | 10.3% | 9 | 11.4% |

**Frage 29a: Warum sehen Sie diese Punkte als Nachteil?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 11 | 100.0% | 37 | 54.4% | 48 | 60.8% |
| Nicht genannt | 0 | 0.0% | 31 | 45.6% | 31 | 39.2% |

| Frage 30: Was kann an DAVE nach Ihrer Meinung verbessert werden? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 6 | 54.5% | 53 | 77.9% | 59 | 74.7% |
| Nicht genannt | 5 | 45.5% | 15 | 22.1% | 20 | 25.3% |

| Frage 31: Welche weiteren Funktionen wünschen Sie sich? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 4 | 36.4% | 34 | 50.0% | 38 | 48.1% |
| Nicht genannt | 7 | 63.6% | 34 | 50.0% | 41 | 51.9% |

| Frage 32: In die Hilfe aufgenommen werden sollte: | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 2 | 18.2% | 32 | 47.1% | 34 | 43.0% |
| Nicht genannt | 9 | 81.8% | 36 | 52.9% | 45 | 57.0% |

| Frage 39: Besitzen Sie zur Zeit einen eigenen Computer? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 11 | 100.0% | 68 | 100.0% | 79 | 100.0% |
| Nein | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |

| Frage 39a: Alter des eigenen Computers | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| 0 – 1 Jahre alt | 6 | 54.5% | 30 | 44.1% | 36 | 45.6% |
| 2 – 3 Jahre alt | 2 | 18.2% | 25 | 36.8% | 27 | 34.2% |
| 4 – 5 Jahre alt | 2 | 18.2% | 10 | 14.7% | 12 | 15.2% |
| 6 – 7 Jahre alt | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| Nicht beantwortet | 1 | 9.1% | 2 | 2.9% | 3 | 3.8% |

| Frage 39b: Prozessorgeschwindigkeit des eigenen Computers | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Weniger als 500 MHz | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| 500 MHz – 999 MHz | 0 | 0.0% | 20 | 29.4% | 20 | 25.3% |
| 1000 MHz – 1499 MHz | 1 | 9.1% | 14 | 20.6% | 15 | 19.0% |
| 1500 MHz – 1999 MHz | 4 | 36.4% | 14 | 20.6% | 18 | 22.8% |
| 2000 MHz – 2499 MHz | 1 | 9.1% | 7 | 10.3% | 8 | 10.1% |
| 2500 MHz – 2999 MHz | 0 | 0.0% | 7 | 10.3% | 7 | 8.9% |
| 3000 MHz und mehr | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| Fehlend / ungültig | 5 | 45.5% | 1 | 1.5% | 6 | 7.6% |

**Frage 39b: Hauptspeichergröße des eigenen Computers**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| 128 MB | 1 | 9.1% | 6 | 8.8% | 7 | 8.9% |
| 256 MB | 2 | 18.2% | 18 | 26.5% | 20 | 25.3% |
| 384 MB | 0 | 0.0% | 5 | 7.4% | 5 | 6.3% |
| 512 MB | 2 | 18.2% | 28 | 41.2% | 30 | 38.0% |
| 640 MB | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| 768 MB | 0 | 0.0% | 5 | 7.4% | 5 | 6.3% |
| 1024 MB | 1 | 9.1% | 3 | 4.4% | 4 | 5.1% |
| Fehlend / ungültig | 5 | 45.5% | 0 | 0.0% | 5 | 6.3% |

**Frage 39c: Festplattengröße des eigenen Computers**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Weniger als 20 GB | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| 20 GB – 39 GB | 1 | 9.1% | 12 | 17.6% | 13 | 16.5% |
| 40 GB – 59 GB | 4 | 36.4% | 16 | 23.5% | 20 | 25.3% |
| 60 GB – 79 GB | 0 | 0.0% | 6 | 8.8% | 6 | 7.6% |
| 80 GB – 99 GB | 1 | 9.1% | 15 | 22.1% | 16 | 20.3% |
| 100 GB – 119 GB | 0 | 0.0% | 4 | 5.9% | 4 | 5.1% |
| 120 GB und mehr | 1 | 9.1% | 10 | 14.7% | 11 | 13.9% |
| Fehlend / ungültig | 4 | 36.4% | 1 | 1.5% | 5 | 6.3% |

**Frage 41: Welches Betriebssystem verwenden Sie?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Windows | 10 | 90.9% | 61 | 89.7% | 71 | 89.9% |
| Linux | 2 | 18.2% | 35 | 51.5% | 37 | 46.8% |
| MacOS | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| Solaris | 1 | 9.1% | 6 | 8.8% | 7 | 8.9% |
| Freie Antwort | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |

**Frage 42: Haben Sie zuhause einen Internetzugang?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 11 | 100.0% | 65 | 95.6% | 76 | 96.2% |
| Modem | 5 | 45.5% | 10 | 14.7% | 15 | 19.0% |
| ISDN | 2 | 18.2% | 7 | 10.3% | 9 | 11.4% |
| DSL | 5 | 45.5% | 32 | 47.1% | 37 | 46.8% |
| Freie Antwort | 2 | 18.2% | 18 | 26.5% | 20 | 25.3% |
| Nein | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |

**Frage 58: Studiengang**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Kerninformatik | 6 | 54.5% | 53 | 77.9% | 59 | 74.7% |
| Angewandte Informatik | 4 | 36.4% | 14 | 20.6% | 18 | 22.8% |
| Nebenfach Informatik | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |

| **Frage 60: Fachsemester** | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| 2 Semester | 1 | 9.1% | 0 | 0.0% | 1 | 1.3% |
| 4 Semester | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |
| 6 Semester | 6 | 54.5% | 30 | 44.1% | 36 | 45.6% |
| 8 Semester | 2 | 18.2% | 22 | 32.4% | 24 | 30.4% |
| 10 Semester | 1 | 9.1% | 6 | 8.8% | 7 | 8.9% |
| 12 Semester | 0 | 0.0% | 3 | 4.4% | 3 | 3.8% |
| 14 Semester | 1 | 9.1% | 1 | 1.5% | 2 | 2.5% |
| 16 Semester | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| 18 Semester | 0 | 0.0% | 2 | 2.9% | 2 | 2.5% |
| 26 Semester | 0 | 0.0% | 1 | 1.5% | 1 | 1.3% |

# E Fragebogen der zweiten Evaluation


**Fragebogen für die Teilnehmer der SWT-Vorlesung WS 2003/2004**

Liebe Studierende,

Sie haben während der Vorlesung das Modellierungswerkzeug DAVE kennen gelernt, das innerhalb des Projektes „MuSofT – Multimedia in der Softwaretechnik“ für Sie entwickelt worden ist. Sie haben mit Hilfe von DAVE die Aufgaben zu Zustandsdiagrammen bearbeitet.

Da es sich bei DAVE um ein relativ neues Programm handelt, ist es für uns wichtig zu erfahren, wie Sie damit zurecht kommen und welche Verbesserungen Sie sich wünschen. Bitte gehen Sie die folgenden Seiten durch und kreuzen Sie jeweils die Aussage(en) an, die Sie als zutreffend empfinden.

Das Ausfüllen des Fragebogens sollte nicht mehr als 10 Minuten in Anspruch nehmen. Die Ergebnisse der Befragung werden für die Weiterentwicklung von DAVE und zu Forschungszwecken genutzt. Die Teilnahme an der Befragung geschieht freiwillig. Ihre Angaben werden selbstverständlich anonym ausgewertet.

Vielen Dank für Ihre Mitarbeit!

Jörg Pleumann <pleumann@ls10.cs.uni-dortmund.de>

1

**1. Bitte bewerten Sie den Schwierigkeitsgrad der einzelnen Themen der Vorlesung:**

| | Sehr schwierig | Schwierig | Einfach | Sehr einfach | Weiß nicht |
|---|---|---|---|---|---|
| Anwendungsfalldiagramme | ❑ | ❑ | ❑ | ❑ | ❑ |
| Klassendiagramme | ❑ | ❑ | ❑ | ❑ | ❑ |
| Objektdiagramme | ❑ | ❑ | ❑ | ❑ | ❑ |
| Zustandsdiagramme | ❑ | ❑ | ❑ | ❑ | ❑ |
| Sequenzdiagramme | ❑ | ❑ | ❑ | ❑ | ❑ |

**2. Bitte bewerten Sie den Schwierigkeitsgrad folgender Tätigkeiten:**

| | Sehr schwierig | Schwierig | Einfach | Sehr einfach | Weiß nicht |
|---|---|---|---|---|---|
| Einen gegebenen Sachverhalt durch ein Klassendiagramm ausdrücken. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ein Objektdiagramm zu einem gegebenen Klassendiagramm erstellen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Aus einem Objektdiagramm ablesen, welche Objekte wie interagieren können. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Aus einem Zustandsdiagramm das Verhalten des Systems ablesen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Das Verhalten eines Systems mit einem Zustandsdiagramm beschreiben. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Aus einem Sequenzdiagramm die Aufrufreihenfolge der Methoden ablesen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Eine Interaktion von Objekten in einem Sequenzdiagramm beschreiben. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Den aktuellen Zustand eines Programms durch ein Objektdiagramm darstellen. | ❑ | ❑ | ❑ | ❑ | ❑ |

**3. Bitte bewerten Sie folgende Aussagen zu den verschiedenen UML-Diagrammtypen:**

| | Stimme nicht zu | Stimme weniger zu | Stimme eher zu | Stimme voll zu | Weiß nicht |
|---|---|---|---|---|---|
| Das Zusammenspiel der verschiedenen Diagramme ist schwer zu durchschauen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die statische Natur der Diagramme erschwert das Verständnis der Dynamik, die sich dahinter verbirgt. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Der Bezug zwischen der abstrakten UML und realen Problemen ist schwer zu sehen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Der Bezug zwischen UML-Diagrammen und Java-Programmen ist schwer zu sehen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Bei einigen Diagrammtypen weiß ich überhaupt nicht, warum ich mich damit beschäftigen soll. | ❑ | ❑ | ❑ | ❑ | ❑ |

**4. Wie haben Sie die Übungsaufgaben der SWT-Vorlesung bearbeitet, die nicht mit Zustandsdiagrammen zu tun hatten?**

❑ Mit Stift und Zettel
❑ Mit dem Computer und der Software
  ❑ Together
  ❑ Rational Rose
  ❑ ArgoUML
  ❑ Poseidon
  ❑ Visio
  ❑ _____

**5. Bitte bewerten Sie folgende Aussagen zu der Vorlesung, in der DAVE vorgestellt wurde.**

| | Stimme nicht zu | Stimme weniger zu | Stimme eher zu | Stimme voll zu | Weiß nicht |
|---|---|---|---|---|---|
| Nach der Vorstellung freute ich mich darauf, mit DAVE zu arbeiten. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Der Nutzen von DAVE wurde deutlich. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Vorlesung zu DAVE und die anderen Vorlesungen zu Zustandsdiagrammen waren gut aufeinander abgestimmt. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Viele Fragen sind offen geblieben. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Der Vortrag hätte kürzer sein können. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich konnte die Details des Programms auf dem Beamer gut erkennen. | ❑ | ❑ | ❑ | ❑ | ❑ |

**6. Besitzen Sie zuhause einen Rechner?**

❑ Ja (bitte Daten angeben, falls bekannt)

_____ MHz Prozessor
_____ MB Hauptspeicher
_____ GB Festplatte

❑ Internet über Modem oder ISDN
❑ Internet über DSL oder Standleitung
❑ Kein Internet

❑ Nein (weiter bei Frage 8)

**7. Falls Sie zuhause einen Rechner besitzen, welche(s) Betriebssystem(e) verwenden Sie?**

❑ Windows
❑ Linux
❑ MacOS
❑ _____

**8. Wo haben Sie die Aufgaben zu DAVE gelöst?**

❑ Zuhause
❑ In einem Computer-Pool der Uni

**9. Bitte kreuzen Sie die zutreffende(n) Aussage(n) bezüglich der Installation von DAVE an.**

❑ Die Installation verlief problemlos.
❑ Es gab Probleme, aber ich konnte sie lösen.
❑ Die Installation klappte gar nicht.

3

**10. Bitte kreuzen Sie die zutreffende(n) Aussage(n) bezüglich der benötigten Java-Umgebung an.**

❑ Eine aktuelle Version von Java war bereits installiert.
❑ Ich musste eine neue(re) Version von Java installieren
  ❑ Das ging problemlos.
  ❑ Es gab Probleme, aber ich konnte sie lösen.
  ❑ Es klappte gar nicht.

**11. Falls Sie bei der Installation von DAVE oder Java Probleme hatten, schildern Sie diese bitte kurz.**

_____
_____
_____

**12. Wie lange haben Sie insgesamt mit DAVE gearbeitet?**

❑ unter 3 Stunden ❑ 5 – 10 Stunden
❑ 3 – 5 Stunden ❑ über 10 Stunden

**13. In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt?**

| | Nie | Einmal | Mehrmals | Ständig |
|---|---|---|---|---|
| Die Simulation von Zustandsdiagrammen | ❑ | ❑ | ❑ | ❑ |
| Die Veranschaulichung durch die Waschmaschine | ❑ | ❑ | ❑ | ❑ |
| Den Navigator (Hierarchie / Schnellansicht) | ❑ | ❑ | ❑ | ❑ |
| Den Inspektor (Eigenschaften / Notizen) | ❑ | ❑ | ❑ | ❑ |
| Die Exportfunktion | ❑ | ❑ | ❑ | ❑ |
| Die Druckfunktion | ❑ | ❑ | ❑ | ❑ |
| Die Online-Hilfe | ❑ | ❑ | ❑ | ❑ |
| Die Homepage des Programms | ❑ | ❑ | ❑ | ❑ |

**14. Bitte bewerten Sie folgende Aussagen zur Simulationsfunktion von DAVE.**

| | Stimme nicht zu | Stimme weniger zu | Stimme eher zu | Stimme voll zu | Weiß nicht |
|---|---|---|---|---|---|
| Die Simulation hat mich beim Lösen der Aufgaben unterstützt. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich fand es gut, ein unmittelbares Feedback zu meinem Modell zu bekommen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich halte die Simulation für überflüssig, weil mein Übungsgruppenleiter mein Modell bewertet. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Durch die Simulation habe ich ein besseres Verständnis der Zustandsdiagramme gewonnen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Simulation verleitet dazu, die Aufgaben per „Trial and Error“ zu lösen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Simulation macht das Lösen der Aufgaben zu einfach. | ❑ | ❑ | ❑ | ❑ | ❑ |

**15. Bitte bewerten Sie folgende Aussagen zur Untermalung der Simulation durch die Waschmaschine.**

| | Stimme nicht zu | Stimme weniger zu | Stimme eher zu | Stimme voll zu | Weiß nicht |
|---|---|---|---|---|---|
| Ich habe mich durch das Beispiel angesprochen gefühlt. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Durch das Beispiel wurde das sonst eher abstrakte Problem anschaulich. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Das Beispiel hat mich zur Beschäftigung mit der Aufgabe motiviert. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Durch das Beispiel wurde der praktische Nutzen von Zustandsdiagrammen deutlich. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Ansteuerung der Waschmaschine durch das Modell war leicht zu verstehen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Aufgabe zur Waschmaschine hat mir besser gefallen als die anderen Aufgaben. | ❑ | ❑ | ❑ | ❑ | ❑ |

**16. Traten bei der Arbeit mit DAVE technische Probleme auf?**

❑ Ja
❑ Nein (weiter bei Frage 19)

**17. Falls Sie Probleme hatten, schildern Sie diese bitte kurz.**

_____
_____
_____

**18. Falls Sie Probleme hatten, wie haben Sie diese lösen können?**

❑ Selbst ist die Frau / der Mann
❑ Frage in der Übungsgruppe
❑ E-Mail an die Support-Adresse
❑ Problem konnte nicht gelöst werden
❑ Tipp einer Kommilitonin / eines Kommilitonen
❑ Blick in die Online-Hilfe
❑ Besuch der Homepage von DAVE
❑ _____

**19. Bitte bewerten Sie folgende Aussagen zum Gesamteindruck von DAVE.**

| | Stimme nicht zu | Stimme weniger zu | Stimme eher zu | Stimme voll zu | Weiß nicht |
|---|---|---|---|---|---|
| Die Oberfläche von DAVE ist optisch ansprechend und gut strukturiert. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Bedienung von DAVE fiel mir leicht. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die Ausführungsgeschwindigkeit von DAVE war angenehm. | ❑ | ❑ | ❑ | ❑ | ❑ |
| DAVE hat mir beim Lösen der Aufgaben geholfen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich würde DAVE anderen Studierenden weiterempfehlen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ich fände ein ähnliches Werkzeug auch für andere Teile der Vorlesung sinnvoll. | ❑ | ❑ | ❑ | ❑ | ❑ |

**20. Bitte bewerten Sie folgende Aussagen zur Arbeit mit Modellierungswerkzeugen in den Übungen.**

| | Stimme nicht zu | Stimme weniger zu | Stimme eher zu | Stimme voll zu | Weiß nicht |
|---|---|---|---|---|---|
| Ich mag die dem professionellen Umfeld entsprechende Arbeitsweise mit einem Werkzeug. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Die bessere optische Gestaltung der Lösungen durch ein Werkzeug empfinde ich als positiv. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Ein Werkzeug verleitet zu einer übertriebenen Ästhetisierung der Lösungen. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Eine ausschließlich elektronische Abgabe der Lösungen fände ich praktisch. | ❑ | ❑ | ❑ | ❑ | ❑ |
| Am Werkzeug kann jeweils nur eine(r) arbeiten. Das behindert die Gruppenarbeit. | ❑ | ❑ | ❑ | ❑ | ❑ |

**21. Was kann an DAVE Ihrer Meinung nach verbessert werden?**

_____
_____
_____

**22. Welche weiteren Funktionen wünschen Sie sich?**

_____
_____
_____

**23. Welche zusätzlichen Themen sollten in die Hilfe aufgenommen werden?**

_____
_____
_____

**24. Bitte geben Sie Ihr Geschlecht an.**

❑ Männlich
❑ Weiblich

**– Das war's auch schon. Vielen Dank! –**

# F Ergebnisse der zweiten Evaluation

**Frage 1a: Bitte bewerten Sie den Schwierigkeitsgrad der einzelnen Themen der Vorlesung: Anwendungsfalldiagramme**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |
| Schwierig | 9 | 22.5% | 42 | 15.3% | 51 | 16.2% |
| Einfach | 18 | 45.0% | 154 | 56.2% | 172 | 54.8% |
| Sehr einfach | 2 | 5.0% | 25 | 9.1% | 27 | 8.6% |
| Weiß nicht | 9 | 22.5% | 42 | 15.3% | 51 | 16.2% |
| Fehlend / ungültig | 2 | 5.0% | 10 | 3.6% | 12 | 3.8% |

**Frage 1b: Bitte bewerten Sie den Schwierigkeitsgrad der einzelnen Themen der Vorlesung: Klassendiagramme**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Schwierig | 5 | 12.5% | 22 | 8.0% | 27 | 8.6% |
| Einfach | 30 | 75.0% | 196 | 71.5% | 226 | 72.0% |
| Sehr einfach | 3 | 7.5% | 48 | 17.5% | 51 | 16.2% |
| Weiß nicht | 0 | 0.0% | 6 | 2.2% | 6 | 1.9% |
| Fehlend / ungültig | 2 | 5.0% | 2 | 0.7% | 4 | 1.3% |

**Frage 1c: Bitte bewerten Sie den Schwierigkeitsgrad der einzelnen Themen der Vorlesung: Objektdiagramme**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |
| Schwierig | 6 | 15.0% | 30 | 10.9% | 36 | 11.5% |
| Einfach | 28 | 70.0% | 181 | 66.1% | 209 | 66.6% |
| Sehr einfach | 3 | 7.5% | 54 | 19.7% | 57 | 18.2% |
| Weiß nicht | 1 | 2.5% | 7 | 2.6% | 8 | 2.5% |
| Fehlend / ungültig | 2 | 5.0% | 1 | 0.4% | 3 | 1.0% |

**Frage 1d: Bitte bewerten Sie den Schwierigkeitsgrad der einzelnen Themen der Vorlesung: Zustandsdiagramme**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |
| Schwierig | 11 | 27.5% | 56 | 20.4% | 67 | 21.3% |
| Einfach | 27 | 67.5% | 182 | 66.4% | 209 | 66.6% |
| Sehr einfach | 1 | 2.5% | 27 | 9.9% | 28 | 8.9% |
| Weiß nicht | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| Fehlend / ungültig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |

**Frage 1e: Bitte bewerten Sie den Schwierigkeitsgrad der einzelnen Themen der Vorlesung: Sequenzdiagramme**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 1 | 2.5% | 15 | 5.5% | 16 | 5.1% |
| Schwierig | 23 | 57.5% | 114 | 41.6% | 137 | 43.6% |
| Einfach | 7 | 17.5% | 80 | 29.2% | 87 | 27.7% |
| Sehr einfach | 0 | 0.0% | 10 | 3.6% | 10 | 3.2% |
| Weiß nicht | 8 | 20.0% | 49 | 17.9% | 57 | 18.2% |
| Fehlend / ungültig | 1 | 2.5% | 6 | 2.2% | 7 | 2.2% |

**Frage 2a: Einen gegebenen Sachverhalt durch ein Klassendiagramm ausdrücken.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Schwierig | 7 | 17.5% | 49 | 17.9% | 56 | 17.8% |
| Einfach | 30 | 75.0% | 191 | 69.7% | 221 | 70.4% |
| Sehr einfach | 1 | 2.5% | 27 | 9.9% | 28 | 8.9% |
| Weiß nicht | 1 | 2.5% | 5 | 1.8% | 6 | 1.9% |
| Fehlend / ungültig | 1 | 2.5% | 2 | 0.7% | 3 | 1.0% |

**Frage 2b: Ein Objektdiagramm zu einem gegebenen Klassendiagramm erstellen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 0 | 0.0% | 0 | 0.0% |
| Schwierig | 6 | 15.0% | 21 | 7.7% | 27 | 8.6% |
| Einfach | 27 | 67.5% | 155 | 56.6% | 182 | 58.0% |
| Sehr einfach | 5 | 12.5% | 94 | 34.3% | 99 | 31.5% |
| Weiß nicht | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| Fehlend / ungültig | 1 | 2.5% | 1 | 0.4% | 2 | 0.6% |

**Frage 2c: Aus einem Objektdiagramm ablesen, welche Objekte wie interagieren können.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 1 | 2.5% | 1 | 0.4% | 2 | 0.6% |
| Schwierig | 7 | 17.5% | 38 | 13.9% | 45 | 14.3% |
| Einfach | 26 | 65.0% | 169 | 61.7% | 195 | 62.1% |
| Sehr einfach | 3 | 7.5% | 57 | 20.8% | 60 | 19.1% |
| Weiß nicht | 1 | 2.5% | 8 | 2.9% | 9 | 2.9% |
| Fehlend / ungültig | 2 | 5.0% | 1 | 0.4% | 3 | 1.0% |

**Frage 2d: Aus einem Zustandsdiagramm das Verhalten des Systems ablesen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |
| Schwierig | 18 | 45.0% | 47 | 17.2% | 65 | 20.7% |
| Einfach | 16 | 40.0% | 176 | 64.2% | 192 | 61.1% |
| Sehr einfach | 4 | 10.0% | 37 | 13.5% | 41 | 13.1% |
| Weiß nicht | 2 | 5.0% | 11 | 4.0% | 13 | 4.1% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |

**Frage 2e: Das Verhalten eines Systems mit einem Zustandsdiagramm beschreiben.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |
| Schwierig | 22 | 55.0% | 96 | 35.0% | 118 | 37.6% |
| Einfach | 16 | 40.0% | 152 | 55.5% | 168 | 53.5% |
| Sehr einfach | 1 | 2.5% | 12 | 4.4% | 13 | 4.1% |
| Weiß nicht | 1 | 2.5% | 10 | 3.6% | 11 | 3.5% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |

**Frage 2f: Aus einem Sequenzdiagramm die Aufrufreihenfolge der Methoden ablesen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 1 | 2.5% | 4 | 1.5% | 5 | 1.6% |
| Schwierig | 15 | 37.5% | 64 | 23.4% | 79 | 25.2% |
| Einfach | 14 | 35.0% | 122 | 44.5% | 136 | 43.3% |
| Sehr einfach | 1 | 2.5% | 39 | 14.2% | 40 | 12.7% |
| Weiß nicht | 8 | 20.0% | 41 | 15.0% | 49 | 15.6% |
| Fehlend / ungültig | 1 | 2.5% | 4 | 1.5% | 5 | 1.6% |

**Frage 2g: Eine Interaktion von Objekten in einem Sequenzdiagramm beschreiben.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 1 | 2.5% | 11 | 4.0% | 12 | 3.8% |
| Schwierig | 25 | 62.5% | 119 | 43.4% | 144 | 45.9% |
| Einfach | 4 | 10.0% | 72 | 26.3% | 76 | 24.2% |
| Sehr einfach | 1 | 2.5% | 14 | 5.1% | 15 | 4.8% |
| Weiß nicht | 8 | 20.0% | 52 | 19.0% | 60 | 19.1% |
| Fehlend / ungültig | 1 | 2.5% | 6 | 2.2% | 7 | 2.2% |

**Frage 2h: Den aktuellen Zustand eines Programms durch ein Objektdiagrammm darstellen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Sehr schwierig | 1 | 2.5% | 5 | 1.8% | 6 | 1.9% |
| Schwierig | 12 | 30.0% | 63 | 23.0% | 75 | 23.9% |
| Einfach | 15 | 37.5% | 146 | 53.3% | 161 | 51.3% |
| Sehr einfach | 2 | 5.0% | 16 | 5.8% | 18 | 5.7% |
| Weiß nicht | 7 | 17.5% | 38 | 13.9% | 45 | 14.3% |
| Fehlend / ungültig | 3 | 7.5% | 6 | 2.2% | 9 | 2.9% |

**Frage 3a: Das Zusammenspiel der verschiedenen Diagramme ist schwer zu durchschauen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 5.0% | 19 | 6.9% | 21 | 6.7% |
| Stimme weniger zu | 19 | 47.5% | 109 | 39.8% | 128 | 40.8% |
| Stimme eher zu | 14 | 35.0% | 105 | 38.3% | 119 | 37.9% |
| Stimme voll zu | 2 | 5.0% | 20 | 7.3% | 22 | 7.0% |
| Weiß nicht | 2 | 5.0% | 17 | 6.2% | 19 | 6.1% |
| Fehlend / ungültig | 1 | 2.5% | 4 | 1.5% | 5 | 1.6% |

**Frage 3b: Die statische Natur der Diagramme erschwert das Verständnis der Dynamik, die sich dahinter verbirgt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 5.0% | 25 | 9.1% | 27 | 8.6% |
| Stimme weniger zu | 9 | 22.5% | 117 | 42.7% | 126 | 40.1% |
| Stimme eher zu | 19 | 47.5% | 95 | 34.7% | 114 | 36.3% |
| Stimme voll zu | 3 | 7.5% | 8 | 2.9% | 11 | 3.5% |
| Weiß nicht | 4 | 10.0% | 24 | 8.8% | 28 | 8.9% |
| Fehlend / ungültig | 3 | 7.5% | 5 | 1.8% | 8 | 2.5% |

**Frage 3c: Der Bezug zwischen der abstrakten UML und realen Problemen ist schwer zu sehen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 7.5% | 57 | 20.8% | 60 | 19.1% |
| Stimme weniger zu | 19 | 47.5% | 155 | 56.6% | 174 | 55.4% |
| Stimme eher zu | 10 | 25.0% | 41 | 15.0% | 51 | 16.2% |
| Stimme voll zu | 5 | 12.5% | 10 | 3.6% | 15 | 4.8% |
| Weiß nicht | 1 | 2.5% | 6 | 2.2% | 7 | 2.2% |
| Fehlend / ungültig | 2 | 5.0% | 5 | 1.8% | 7 | 2.2% |

**Frage 3d: Der Bezug zwischen UML-Diagrammen und Java-Programmen ist schwer zu sehen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 7.5% | 72 | 26.3% | 75 | 23.9% |
| Stimme weniger zu | 17 | 42.5% | 115 | 42.0% | 132 | 42.0% |
| Stimme eher zu | 15 | 37.5% | 54 | 19.7% | 69 | 22.0% |
| Stimme voll zu | 4 | 10.0% | 23 | 8.4% | 27 | 8.6% |
| Weiß nicht | 1 | 2.5% | 6 | 2.2% | 7 | 2.2% |
| Fehlend / ungültig | 0 | 0.0% | 4 | 1.5% | 4 | 1.3% |

**Frage 3e: Bei einigen Diagrammtypen weiß ich gar nicht, warum ich mich damit beschäftigen soll.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 11 | 27.5% | 59 | 21.5% | 70 | 22.3% |
| Stimme weniger zu | 9 | 22.5% | 104 | 38.0% | 113 | 36.0% |
| Stimme eher zu | 13 | 32.5% | 57 | 20.8% | 70 | 22.3% |
| Stimme voll zu | 4 | 10.0% | 37 | 13.5% | 41 | 13.1% |
| Weiß nicht | 3 | 7.5% | 13 | 4.7% | 16 | 5.1% |
| Fehlend / ungültig | 0 | 0.0% | 4 | 1.5% | 4 | 1.3% |

**Frage 4: Wie haben Sie die Übungsaufgaben der SWT-Vorlesuung bearbeitet, die nicht mit Zustandsdiagrammen zu tun hatten?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Mit Stift und Zettel | 27 | 67.5% | 161 | 58.8% | 188 | 59.9% |
| Mit dem Computer und der Software... | 21 | 52.5% | 157 | 57.3% | 178 | 56.7% |
| Together | 4 | 10.0% | 22 | 8.0% | 26 | 8.3% |
| Rational Rose | 0 | 0.0% | 5 | 1.8% | 5 | 1.6% |
| ArgoUML | 0 | 0.0% | 26 | 9.5% | 26 | 8.3% |
| Poseidon | 2 | 5.0% | 12 | 4.4% | 14 | 4.5% |
| Visio | 14 | 35.0% | 63 | 23.0% | 77 | 24.5% |
| Freie Antwort | 9 | 22.5% | 73 | 26.6% | 82 | 26.1% |

**Frage 5a: Nach der Vorstellung freute ich mich darauf, mit DAVE zu arbeiten.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 7.5% | 20 | 7.3% | 23 | 7.3% |
| Stimme weniger zu | 12 | 30.0% | 55 | 20.1% | 67 | 21.3% |
| Stimme eher zu | 13 | 32.5% | 94 | 34.3% | 107 | 34.1% |
| Stimme voll zu | 8 | 20.0% | 59 | 21.5% | 67 | 21.3% |
| Weiß nicht | 3 | 7.5% | 36 | 13.1% | 39 | 12.4% |
| Fehlend / ungültig | 1 | 2.5% | 10 | 3.6% | 11 | 3.5% |

**Frage 5b: Der Nutzen von DAVE wurde deutlich.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 8 | 2.9% | 8 | 2.5% |
| Stimme weniger zu | 8 | 20.0% | 34 | 12.4% | 42 | 13.4% |
| Stimme eher zu | 20 | 50.0% | 134 | 48.9% | 154 | 49.0% |
| Stimme voll zu | 6 | 15.0% | 72 | 26.3% | 78 | 24.8% |
| Weiß nicht | 4 | 10.0% | 19 | 6.9% | 23 | 7.3% |
| Fehlend / ungültig | 2 | 5.0% | 7 | 2.6% | 9 | 2.9% |

**Frage 5c: Die Vorlesung zu DAVE und die anderen Vorlesungen zu Zustandsdiagrammen waren gut aufeinander abgestimmt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 7 | 2.6% | 7 | 2.2% |
| Stimme weniger zu | 12 | 30.0% | 52 | 19.0% | 64 | 20.4% |
| Stimme eher zu | 12 | 30.0% | 135 | 49.3% | 147 | 46.8% |
| Stimme voll zu | 5 | 12.5% | 33 | 12.0% | 38 | 12.1% |
| Weiß nicht | 8 | 20.0% | 39 | 14.2% | 47 | 15.0% |
| Fehlend / ungültig | 3 | 7.5% | 8 | 2.9% | 11 | 3.5% |

**Frage 5d: Viele Fragen sind offen geblieben.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 7.5% | 30 | 10.9% | 33 | 10.5% |
| Stimme weniger zu | 11 | 27.5% | 111 | 40.5% | 122 | 38.9% |
| Stimme eher zu | 14 | 35.0% | 66 | 24.1% | 80 | 25.5% |
| Stimme voll zu | 7 | 17.5% | 33 | 12.0% | 40 | 12.7% |
| Weiß nicht | 3 | 7.5% | 25 | 9.1% | 28 | 8.9% |
| Fehlend / ungültig | 2 | 5.0% | 9 | 3.3% | 11 | 3.5% |

**Frage 5e: Der Vortrag hätte kürzer sein können.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 12 | 30.0% | 47 | 17.2% | 59 | 18.8% |
| Stimme weniger zu | 11 | 27.5% | 109 | 39.8% | 120 | 38.2% |
| Stimme eher zu | 6 | 15.0% | 47 | 17.2% | 53 | 16.9% |
| Stimme voll zu | 3 | 7.5% | 14 | 5.1% | 17 | 5.4% |
| Weiß nicht | 5 | 12.5% | 50 | 18.2% | 55 | 17.5% |
| Fehlend / ungültig | 3 | 7.5% | 7 | 2.6% | 10 | 3.2% |

**Frage 5f: Ich konnte die Details des Programms auf dem Beamer gut erkennen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 3 | 7.5% | 13 | 4.7% | 16 | 5.1% |
| Stimme weniger zu | 8 | 20.0% | 53 | 19.3% | 61 | 19.4% |
| Stimme eher zu | 15 | 37.5% | 92 | 33.6% | 107 | 34.1% |
| Stimme voll zu | 7 | 17.5% | 64 | 23.4% | 71 | 22.6% |
| Weiß nicht | 4 | 10.0% | 44 | 16.1% | 48 | 15.3% |
| Fehlend / ungültig | 3 | 7.5% | 8 | 2.9% | 11 | 3.5% |

| Frage 6: Besitzen Sie zuhause einen eigenen Rechner? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 36 | 90.0% | 262 | 95.6% | 298 | 94.9% |
| Nein | 2 | 5.0% | 7 | 2.6% | 9 | 2.9% |
| Fehlend / ungültig | 2 | 5.0% | 5 | 1.8% | 7 | 2.2% |

| Frage 6a: Geschwindigkeit des eigenen Rechners? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| weniger als 500 MHz | 0 | 0.0% | 6 | 2.2% | 6 | 1.9% |
| 500 MHz bis 999 MHz | 4 | 10.0% | 33 | 12.0% | 37 | 11.8% |
| 1000 MHz bis 1499 MHz | 3 | 7.5% | 56 | 20.4% | 59 | 18.8% |
| 1500 MHz bis 1999 MHz | 5 | 12.5% | 76 | 27.7% | 81 | 25.8% |
| 2000 MHz bis 2499 MHz | 8 | 20.0% | 62 | 22.6% | 70 | 22.3% |
| 2500 MHz bis 2999 MHz | 2 | 5.0% | 20 | 7.3% | 22 | 7.0% |
| 3000 MHz und mehr | 0 | 0.0% | 7 | 2.6% | 7 | 2.2% |
| Fehlend / ungültig | 18 | 45.0% | 14 | 5.1% | 32 | 10.2% |

| Frage 6b: Hauptspeicher des eigenen Rechners? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| 64 MB | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |
| 128 MB | 4 | 10.0% | 16 | 5.8% | 20 | 6.4% |
| 192 MB | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| 256 MB | 9 | 22.5% | 57 | 20.8% | 66 | 21.0% |
| 384 MB | 2 | 5.0% | 7 | 2.6% | 9 | 2.9% |
| 512 MB | 6 | 15.0% | 130 | 47.4% | 136 | 43.3% |
| 640 MB | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |
| 768 MB | 0 | 0.0% | 12 | 4.4% | 12 | 3.8% |
| 1024 MB | 0 | 0.0% | 22 | 8.0% | 22 | 7.0% |
| Fehlend / ungültig | 18 | 45.0% | 24 | 8.8% | 42 | 13.4% |

| Frage 6c: Festplattengröße des eigenen Rechners? | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| weniger als 20 GB | 2 | 5.0% | 8 | 2.9% | 10 | 3.2% |
| 20 GB bis 39 GB | 8 | 20.0% | 12 | 4.4% | 20 | 6.4% |
| 40 GB bis 59 GB | 5 | 12.5% | 49 | 17.9% | 54 | 17.2% |
| 60 GB bis 79 GB | 2 | 5.0% | 30 | 10.9% | 32 | 10.2% |
| 80 GB bis 99 GB | 5 | 12.5% | 43 | 15.7% | 48 | 15.3% |
| 100 GB bis 119 GB | 0 | 0.0% | 14 | 5.1% | 14 | 4.5% |
| 120 GB bis 159 GB | 1 | 2.5% | 32 | 11.7% | 33 | 10.5% |
| 160 GB bis 199 GB | 2 | 5.0% | 25 | 9.1% | 27 | 8.6% |
| 200 GB bis 239 GB | 0 | 0.0% | 17 | 6.2% | 17 | 5.4% |
| 240 GB und mehr | 0 | 0.0% | 18 | 6.6% | 18 | 5.7% |
| Fehlend / ungültig | 15 | 37.5% | 26 | 9.5% | 41 | 13.1% |

**Frage 6d: Internetzugang**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Über Modem oder ISDN | 7 | 17.5% | 47 | 17.2% | 54 | 17.2% |
| Über DSL oder Standleitung | 21 | 52.5% | 199 | 72.6% | 220 | 70.1% |
| Kein Internet | 9 | 22.5% | 13 | 4.7% | 22 | 7.0% |
| Fehlend / ungültig | 3 | 7.5% | 15 | 5.5% | 18 | 5.7% |

**Frage 7: Falls Sie zuhause einen Rechner besitzen, welche(s) Betriebssystem(e) verwenden Sie?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Windows | 38 | 95.0% | 247 | 90.1% | 285 | 90.8% |
| Linux | 4 | 10.0% | 117 | 42.7% | 121 | 38.5% |
| MacOS | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| Freie Antwort | 0 | 0.0% | 4 | 1.5% | 4 | 1.3% |

**Frage 8: Wo haben Sie die Aufgaben zu DAVE gelöst?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Zuhause | 36 | 90.0% | 255 | 93.1% | 291 | 92.7% |
| In einem Computer-Pool der Uni | 3 | 7.5% | 14 | 5.1% | 17 | 5.4% |
| Fehlend / ungültig | 1 | 2.5% | 5 | 1.8% | 6 | 1.9% |

**Frage 9: Bitte kreuzen Sie die zutreffende(n) Aussagen bezüglich der Installation von DAVE an.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Die Installation verlief problemlos. | 30 | 75.0% | 234 | 85.4% | 264 | 84.1% |
| Es gab Probleme, aber ich konnte sie lösen. | 8 | 20.0% | 35 | 12.8% | 43 | 13.7% |
| Die Installation klappte gar nicht. | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| Fehlend / ungültig | 1 | 2.5% | 2 | 0.7% | 3 | 1.0% |

**Frage 10: Bitte kreuzen Sie die zutreffende(n) Aussage(n) bezüglich der benötigten Java-Umgebung an.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Eine aktuelle Version von Java war bereits installiert. | 25 | 62.5% | 215 | 78.5% | 240 | 76.4% |
| Ich musste eine neue(re) Version von Java installieren. | 13 | 32.5% | 51 | 18.6% | 64 | 20.4% |
| Das ging problemlos. | 9 | 22.5% | 42 | 15.3% | 51 | 16.2% |
| Es gab Probleme, aber ich konnte sie lösen. | 5 | 12.5% | 11 | 4.0% | 16 | 5.1% |
| Es klappte gar nicht. | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |

**Frage 11: Falls Sie bei der Installation von DAVE oder Java Probleme hatten, schildern Sie diese bitte kurz.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 4 | 10.0% | 24 | 8.8% | 28 | 8.9% |
| Nicht genannt | 36 | 90.0% | 250 | 91.2% | 286 | 91.1% |

**Frage 12: Wie lange haben Sie insgesamt mit DAVE gearbeitet?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Unter 3 Stunden | 3 | 7.5% | 32 | 11.7% | 35 | 11.1% |
| 3 bis 5 Stunden | 13 | 32.5% | 82 | 29.9% | 95 | 30.3% |
| 5 bis 10 Stunden | 14 | 35.0% | 110 | 40.1% | 124 | 39.5% |
| Über 10 Stunden | 8 | 20.0% | 49 | 17.9% | 57 | 18.2% |
| Fehlend / ungültig | 2 | 5.0% | 1 | 0.4% | 3 | 1.0% |

**Frage 13a: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Die Simulation von Zustandsdiagrammen**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| Einmal | 3 | 7.5% | 13 | 4.7% | 16 | 5.1% |
| Mehrmals | 22 | 55.0% | 126 | 46.0% | 148 | 47.1% |
| Ständig | 14 | 35.0% | 131 | 47.8% | 145 | 46.2% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |

**Frage 13b: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Die Veranschaulichung durch die Waschmaschine**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 3 | 7.5% | 32 | 11.7% | 35 | 11.1% |
| Einmal | 11 | 27.5% | 73 | 26.6% | 84 | 26.8% |
| Mehrmals | 19 | 47.5% | 135 | 49.3% | 154 | 49.0% |
| Ständig | 7 | 17.5% | 32 | 11.7% | 39 | 12.4% |
| Fehlend / ungültig | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |

**Frage 13c: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Den Navigator (Hierarchie / Schnellansicht)**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 15 | 37.5% | 86 | 31.4% | 101 | 32.2% |
| Einmal | 5 | 12.5% | 52 | 19.0% | 57 | 18.2% |
| Mehrmals | 13 | 32.5% | 111 | 40.5% | 124 | 39.5% |
| Ständig | 4 | 10.0% | 17 | 6.2% | 21 | 6.7% |
| Fehlend / ungültig | 3 | 7.5% | 8 | 2.9% | 11 | 3.5% |

**Frage 13d: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Den Inspektor (Eigenschaften / Notizen)**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 13 | 32.5% | 54 | 19.7% | 67 | 21.3% |
| Einmal | 4 | 10.0% | 30 | 10.9% | 34 | 10.8% |
| Mehrmals | 15 | 37.5% | 101 | 36.9% | 116 | 36.9% |
| Ständig | 5 | 12.5% | 85 | 31.0% | 90 | 28.7% |
| Fehlend / ungültig | 3 | 7.5% | 4 | 1.5% | 7 | 2.2% |

**Frage 13e: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Die Exportfunktion**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 22 | 55.0% | 113 | 41.2% | 135 | 43.0% |
| Einmal | 3 | 7.5% | 46 | 16.8% | 49 | 15.6% |
| Mehrmals | 10 | 25.0% | 87 | 31.8% | 97 | 30.9% |
| Ständig | 1 | 2.5% | 17 | 6.2% | 18 | 5.7% |
| Fehlend / ungültig | 4 | 10.0% | 11 | 4.0% | 15 | 4.8% |

**Frage 13f: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Die Druckfunktion**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 15 | 37.5% | 79 | 28.8% | 94 | 29.9% |
| Einmal | 2 | 5.0% | 35 | 12.8% | 37 | 11.8% |
| Mehrmals | 20 | 50.0% | 134 | 48.9% | 154 | 49.0% |
| Ständig | 1 | 2.5% | 22 | 8.0% | 23 | 7.3% |
| Fehlend / ungültig | 2 | 5.0% | 4 | 1.5% | 6 | 1.9% |

**Frage 13g: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Die Online-Hilfe**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 29 | 72.5% | 113 | 41.2% | 142 | 45.2% |
| Einmal | 6 | 15.0% | 57 | 20.8% | 63 | 20.1% |
| Mehrmals | 4 | 10.0% | 87 | 31.8% | 91 | 29.0% |
| Ständig | 0 | 0.0% | 12 | 4.4% | 12 | 3.8% |
| Fehlend / ungültig | 1 | 2.5% | 5 | 1.8% | 6 | 1.9% |

**Frage 13h: In welchem Umfang haben Sie folgende Funktionen von DAVE genutzt? – Die Homepage des Programms**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Nie | 16 | 40.0% | 111 | 40.5% | 127 | 40.4% |
| Einmal | 18 | 45.0% | 113 | 41.2% | 131 | 41.7% |
| Mehrmals | 4 | 10.0% | 46 | 16.8% | 50 | 15.9% |
| Ständig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |
| Fehlend / ungültig | 2 | 5.0% | 3 | 1.1% | 5 | 1.6% |

**Frage 14a: Die Simulation hat mich beim Lösen der Aufgaben unterstützt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 9 | 3.3% | 10 | 3.2% |
| Stimme weniger zu | 6 | 15.0% | 15 | 5.5% | 21 | 6.7% |
| Stimme eher zu | 20 | 50.0% | 86 | 31.4% | 106 | 33.8% |
| Stimme voll zu | 13 | 32.5% | 160 | 58.4% | 173 | 55.1% |
| Weiß nicht | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |
| Fehlend / ungültig | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |

**Frage 14b: Ich fand es gut, ein unmittelbares Feedback zu meinem Modell zu bekommen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |
| Stimme weniger zu | 4 | 10.0% | 8 | 2.9% | 12 | 3.8% |
| Stimme eher zu | 17 | 42.5% | 72 | 26.3% | 89 | 28.3% |
| Stimme voll zu | 16 | 40.0% | 181 | 66.1% | 197 | 62.7% |
| Weiß nicht | 1 | 2.5% | 10 | 3.6% | 11 | 3.5% |
| Fehlend / ungültig | 2 | 5.0% | 1 | 0.4% | 3 | 1.0% |

**Frage 14c: Ich halte die Simulation für überflüssig, weil mein Übungsgruppenleiter mein Modell, bewertet.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 20 | 50.0% | 192 | 70.1% | 212 | 67.5% |
| Stimme weniger zu | 15 | 37.5% | 58 | 21.2% | 73 | 23.2% |
| Stimme eher zu | 3 | 7.5% | 11 | 4.0% | 14 | 4.5% |
| Stimme voll zu | 1 | 2.5% | 8 | 2.9% | 9 | 2.9% |
| Weiß nicht | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |
| Fehlend / ungültig | 1 | 2.5% | 2 | 0.7% | 3 | 1.0% |

**Frage 14d: Durch die Simulation habe ich ein besseres Verständnis der Zustandsdiagramme gewonnen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 15 | 5.5% | 15 | 4.8% |
| Stimme weniger zu | 2 | 5.0% | 33 | 12.0% | 35 | 11.1% |
| Stimme eher zu | 25 | 62.5% | 111 | 40.5% | 136 | 43.3% |
| Stimme voll zu | 12 | 30.0% | 109 | 39.8% | 121 | 38.5% |
| Weiß nicht | 1 | 2.5% | 5 | 1.8% | 6 | 1.9% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |

**Frage 14e: Die Simulation verleitet dazu, die Aufgaben der Trial-and-Error zu lösen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 23 | 8.4% | 24 | 7.6% |
| Stimme weniger zu | 13 | 32.5% | 81 | 29.6% | 94 | 29.9% |
| Stimme eher zu | 13 | 32.5% | 116 | 42.3% | 129 | 41.1% |
| Stimme voll zu | 5 | 12.5% | 44 | 16.1% | 49 | 15.6% |
| Weiß nicht | 7 | 17.5% | 7 | 2.6% | 14 | 4.5% |
| Fehlend / ungültig | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |

**Frage 14f: Die Simulation macht das Lösen der Aufgaben zu einfach.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 10 | 25.0% | 80 | 29.2% | 90 | 28.7% |
| Stimme weniger zu | 19 | 47.5% | 131 | 47.8% | 150 | 47.8% |
| Stimme eher zu | 8 | 20.0% | 39 | 14.2% | 47 | 15.0% |
| Stimme voll zu | 2 | 5.0% | 17 | 6.2% | 19 | 6.1% |
| Weiß nicht | 1 | 2.5% | 6 | 2.2% | 7 | 2.2% |
| Fehlend / ungültig | 0 | 0.0% | 1 | 0.4% | 1 | 0.3% |

**Frage 15a: Ich habe mich durch das Beispiel angesprochen gefühlt.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 5.0% | 17 | 6.2% | 19 | 6.1% |
| Stimme weniger zu | 8 | 20.0% | 67 | 24.5% | 75 | 23.9% |
| Stimme eher zu | 22 | 55.0% | 129 | 47.1% | 151 | 48.1% |
| Stimme voll zu | 4 | 10.0% | 39 | 14.2% | 43 | 13.7% |
| Weiß nicht | 4 | 10.0% | 18 | 6.6% | 22 | 7.0% |
| Fehlend / ungültig | 0 | 0.0% | 4 | 1.5% | 4 | 1.3% |

**Frage 15b: Durch das Beispiel wurde das sonst eher abstrakte Problem anschaulich.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 15 | 5.5% | 16 | 5.1% |
| Stimme weniger zu | 3 | 7.5% | 49 | 17.9% | 52 | 16.6% |
| Stimme eher zu | 26 | 65.0% | 132 | 48.2% | 158 | 50.3% |
| Stimme voll zu | 8 | 20.0% | 58 | 21.2% | 66 | 21.0% |
| Weiß nicht | 2 | 5.0% | 15 | 5.5% | 17 | 5.4% |
| Fehlend / ungültig | 0 | 0.0% | 5 | 1.8% | 5 | 1.6% |

**Frage 15c: Das Beispiel hat mich zur Beschäftigung mit der Aufgabe motiviert.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 5.0% | 30 | 10.9% | 32 | 10.2% |
| Stimme weniger zu | 8 | 20.0% | 82 | 29.9% | 90 | 28.7% |
| Stimme eher zu | 18 | 45.0% | 91 | 33.2% | 109 | 34.7% |
| Stimme voll zu | 9 | 22.5% | 51 | 18.6% | 60 | 19.1% |
| Weiß nicht | 2 | 5.0% | 16 | 5.8% | 18 | 5.7% |
| Fehlend / ungültig | 1 | 2.5% | 4 | 1.5% | 5 | 1.6% |

**Frage 15d: Durch das Beispiel wurde der praktische Nutzen von Zustandsdiagrammen deutlich.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 18 | 6.6% | 18 | 5.7% |
| Stimme weniger zu | 8 | 20.0% | 53 | 19.3% | 61 | 19.4% |
| Stimme eher zu | 23 | 57.5% | 116 | 42.3% | 139 | 44.3% |
| Stimme voll zu | 9 | 22.5% | 70 | 25.5% | 79 | 25.2% |
| Weiß nicht | 0 | 0.0% | 12 | 4.4% | 12 | 3.8% |
| Fehlend / ungültig | 0 | 0.0% | 5 | 1.8% | 5 | 1.6% |

**Frage 15e: Die Ansteuerung der Waschmaschine durch das Modell war leicht zu verstehen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 8 | 2.9% | 8 | 2.5% |
| Stimme weniger zu | 4 | 10.0% | 37 | 13.5% | 41 | 13.1% |
| Stimme eher zu | 22 | 55.0% | 125 | 45.6% | 147 | 46.8% |
| Stimme voll zu | 13 | 32.5% | 82 | 29.9% | 95 | 30.3% |
| Weiß nicht | 1 | 2.5% | 15 | 5.5% | 16 | 5.1% |
| Fehlend / ungültig | 0 | 0.0% | 7 | 2.6% | 7 | 2.2% |

**Frage 15f: Die Aufgabe zur Waschmaschine hat mir besser gefallen als die anderen Aufgaben.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 36 | 13.1% | 36 | 11.5% |
| Stimme weniger zu | 11 | 27.5% | 69 | 25.2% | 80 | 25.5% |
| Stimme eher zu | 15 | 37.5% | 72 | 26.3% | 87 | 27.7% |
| Stimme voll zu | 13 | 32.5% | 64 | 23.4% | 77 | 24.5% |
| Weiß nicht | 1 | 2.5% | 28 | 10.2% | 29 | 9.2% |
| Fehlend / ungültig | 0 | 0.0% | 5 | 1.8% | 5 | 1.6% |

**Frage 16: Traten bei der Arbeit mit DAVE technische Probleme auf?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Ja | 4 | 10.0% | 82 | 29.9% | 86 | 27.4% |
| Nein | 34 | 85.0% | 189 | 69.0% | 223 | 71.0% |
| Fehlend / ungültig | 2 | 5.0% | 3 | 1.1% | 5 | 1.6% |

**Frage 17: Falls Sie Probleme hatten, schildern Sie diese bitte kurz.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 4 | 10.0% | 77 | 28.1% | 81 | 25.8% |
| Nicht genannt | 36 | 90.0% | 197 | 71.9% | 233 | 74.2% |

**Frage 18: Falls Sie Probleme hatten, wie haben Sie diese lösen können?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Selbst ist die Frau / der Mann | 8 | 20.0% | 87 | 31.8% | 95 | 30.3% |
| Tipp einer Kommilitonin / eines Kommilitonen | 9 | 22.5% | 42 | 15.3% | 51 | 16.2% |
| Frage in der Übungsgruppe | 4 | 10.0% | 21 | 7.7% | 25 | 8.0% |
| Blick in die Online-Hilfe | 2 | 5.0% | 31 | 11.3% | 33 | 10.5% |
| E-Mail an die Support-Adresse | 2 | 5.0% | 8 | 2.9% | 10 | 3.2% |
| Besuch der Homepage von DAVE | 3 | 7.5% | 19 | 6.9% | 22 | 7.0% |
| Problem konnte nicht gelöst werden | 2 | 5.0% | 22 | 8.0% | 24 | 7.6% |
| Freie Antwort | 1 | 2.5% | 9 | 3.3% | 10 | 3.2% |

**Frage 19a: Die Oberfläche von DAVE ist optisch ansprechend und gut strukturiert.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 9 | 3.3% | 10 | 3.2% |
| Stimme weniger zu | 8 | 20.0% | 28 | 10.2% | 36 | 11.5% |
| Stimme eher zu | 24 | 60.0% | 180 | 65.7% | 204 | 65.0% |
| Stimme voll zu | 7 | 17.5% | 48 | 17.5% | 55 | 17.5% |
| Weiß nicht | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |
| Fehlend / ungültig | 0 | 0.0% | 6 | 2.2% | 6 | 1.9% |

**Frage 19b: Die Bedienung von DAVE fiel mir leicht.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 2 | 5.0% | 11 | 4.0% | 13 | 4.1% |
| Stimme weniger zu | 16 | 40.0% | 75 | 27.4% | 91 | 29.0% |
| Stimme eher zu | 15 | 37.5% | 146 | 53.3% | 161 | 51.3% |
| Stimme voll zu | 5 | 12.5% | 38 | 13.9% | 43 | 13.7% |
| Weiß nicht | 2 | 5.0% | 1 | 0.4% | 3 | 1.0% |
| Fehlend / ungültig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |

**Frage 19c: Die Ausführungsgeschwindigkeit von DAVE war angenehm.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 9 | 3.3% | 10 | 3.2% |
| Stimme weniger zu | 8 | 20.0% | 57 | 20.8% | 65 | 20.7% |
| Stimme eher zu | 19 | 47.5% | 140 | 51.1% | 159 | 50.6% |
| Stimme voll zu | 10 | 25.0% | 62 | 22.6% | 72 | 22.9% |
| Weiß nicht | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |
| Fehlend / ungültig | 1 | 2.5% | 3 | 1.1% | 4 | 1.3% |

**Frage 19d: DAVE hat mir beim Lösen der Aufgaben geholfen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 4 | 1.5% | 4 | 1.3% |
| Stimme weniger zu | 7 | 17.5% | 25 | 9.1% | 32 | 10.2% |
| Stimme eher zu | 21 | 52.5% | 138 | 50.4% | 159 | 50.6% |
| Stimme voll zu | 11 | 27.5% | 103 | 37.6% | 114 | 36.3% |
| Weiß nicht | 1 | 2.5% | 2 | 0.7% | 3 | 1.0% |
| Fehlend / ungültig | 0 | 0.0% | 2 | 0.7% | 2 | 0.6% |

**Frage 19e: Ich würde DAVE anderen Studierenden weiterempfehlen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 11 | 4.0% | 12 | 3.8% |
| Stimme weniger zu | 4 | 10.0% | 24 | 8.8% | 28 | 8.9% |
| Stimme eher zu | 22 | 55.0% | 133 | 48.5% | 155 | 49.4% |
| Stimme voll zu | 11 | 27.5% | 91 | 33.2% | 102 | 32.5% |
| Weiß nicht | 2 | 5.0% | 12 | 4.4% | 14 | 4.5% |
| Fehlend / ungültig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |

**Frage 19f: Ich fände ein ähnliches Werkzeug auch für andere Teile der Vorlesung sinnvoll.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 1 | 2.5% | 7 | 2.6% | 8 | 2.5% |
| Stimme weniger zu | 7 | 17.5% | 21 | 7.7% | 28 | 8.9% |
| Stimme eher zu | 15 | 37.5% | 95 | 34.7% | 110 | 35.0% |
| Stimme voll zu | 15 | 37.5% | 136 | 49.6% | 151 | 48.1% |
| Weiß nicht | 2 | 5.0% | 12 | 4.4% | 14 | 4.5% |
| Fehlend / ungültig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |

**Frage 20a: Ich mag die dem professionellen Umfeld entsprechende Arbeitsweise mit einem Werkzeug.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 7 | 2.6% | 7 | 2.2% |
| Stimme weniger zu | 7 | 17.5% | 14 | 5.1% | 21 | 6.7% |
| Stimme eher zu | 26 | 65.0% | 150 | 54.7% | 176 | 56.1% |
| Stimme voll zu | 6 | 15.0% | 76 | 27.7% | 82 | 26.1% |
| Weiß nicht | 0 | 0.0% | 22 | 8.0% | 22 | 7.0% |
| Fehlend / ungültig | 1 | 2.5% | 5 | 1.8% | 6 | 1.9% |

**Frage 20b: Die bessere optische Gestaltung der Lösungen durch ein Werkzeug empfinde ich als positiv.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 0 | 0.0% | 5 | 1.8% | 5 | 1.6% |
| Stimme weniger zu | 4 | 10.0% | 13 | 4.7% | 17 | 5.4% |
| Stimme eher zu | 21 | 52.5% | 114 | 41.6% | 135 | 43.0% |
| Stimme voll zu | 14 | 35.0% | 136 | 49.6% | 150 | 47.8% |
| Weiß nicht | 1 | 2.5% | 2 | 0.7% | 3 | 1.0% |
| Fehlend / ungültig | 0 | 0.0% | 4 | 1.5% | 4 | 1.3% |

**Frage 20c: Ein Werkzeug verleitet zu einer übertriebenen Ästhetisierung der Lösungen.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 7 | 17.5% | 56 | 20.4% | 63 | 20.1% |
| Stimme weniger zu | 15 | 37.5% | 121 | 44.2% | 136 | 43.3% |
| Stimme eher zu | 8 | 20.0% | 51 | 18.6% | 59 | 18.8% |
| Stimme voll zu | 2 | 5.0% | 19 | 6.9% | 21 | 6.7% |
| Weiß nicht | 6 | 15.0% | 19 | 6.9% | 25 | 8.0% |
| Fehlend / ungültig | 2 | 5.0% | 8 | 2.9% | 10 | 3.2% |

**Frage 20d: Eine ausschließlich elektronische Abgabe der Lösungen fände ich praktisch.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 6 | 15.0% | 27 | 9.9% | 33 | 10.5% |
| Stimme weniger zu | 14 | 35.0% | 32 | 11.7% | 46 | 14.6% |
| Stimme eher zu | 6 | 15.0% | 88 | 32.1% | 94 | 29.9% |
| Stimme voll zu | 11 | 27.5% | 113 | 41.2% | 124 | 39.5% |
| Weiß nicht | 3 | 7.5% | 11 | 4.0% | 14 | 4.5% |
| Fehlend / ungültig | 0 | 0.0% | 3 | 1.1% | 3 | 1.0% |

**Frage 20e: Am Werkzeug kann jeweils nur eine(r) arbeiten. Das behindert die Gruppenarbeit.**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Stimme nicht zu | 9 | 22.5% | 78 | 28.5% | 87 | 27.7% |
| Stimme weniger zu | 12 | 30.0% | 109 | 39.8% | 121 | 38.5% |
| Stimme eher zu | 14 | 35.0% | 44 | 16.1% | 58 | 18.5% |
| Stimme voll zu | 1 | 2.5% | 16 | 5.8% | 17 | 5.4% |
| Weiß nicht | 3 | 7.5% | 21 | 7.7% | 24 | 7.6% |
| Fehlend / ungültig | 1 | 2.5% | 6 | 2.2% | 7 | 2.2% |

**Frage 21: Was kann an DAVE Ihrer Meinung nach verbessert werden?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 16 | 40.0% | 181 | 66.1% | 197 | 62.7% |
| Nicht genannt | 24 | 60.0% | 93 | 33.9% | 117 | 37.3% |

**Frage 22: Welche weiteren Funktionen wünschen Sie sich?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 7 | 17.5% | 118 | 43.1% | 125 | 39.8% |
| Nicht genannt | 33 | 82.5% | 156 | 56.9% | 189 | 60.2% |

**Frage 23: Welche zusätzlichen Themen sollten in die Hilfe aufgenommen werden?**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| Genannt | 3 | 7.5% | 78 | 28.5% | 81 | 25.8% |
| Nicht genannt | 37 | 92.5% | 196 | 71.5% | 233 | 74.2% |

**Studiengänge der Teilnehmer SWT 2004/2005**

| | Weiblich | | Männlich | | Gesamt | |
|---|---|---|---|---|---|---|
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| | Absolut | Relativ | | | | |
| Kerninformatik | – | – | – | – | 363 | 81% |
| Angewandte Informatik | – | – | – | – | 45 | 10% |
| Nebenfach Informatik | – | – | – | – | 22 | 4.9% |
| Lehramt Informatik | – | – | – | – | 10 | 2.2% |
| Fehlend / ungültig | – | – | – | – | 8 | 1.8% |

| **Fachsemester der Teilnehmer SWT 2004/2005** | | | | | | |
|---|---|---|---|---|---|---|
| | Weiblich | | Männlich | | Gesamt | |
| | Absolut | Relativ | Absolut | Relativ | Absolut | Relativ |
| 1 | – | – | – | – | 22 | 4.9% |
| 2 | – | – | – | – | 1 | 0.2% |
| 3 | – | – | – | – | 293 | 65.4% |
| 4 | – | – | – | – | 1 | 0.2% |
| 5 | – | – | – | – | 102 | 22.8% |
| 6 | – | – | – | – | 2 | 0.4% |
| 7 | – | – | – | – | 17 | 3.8% |
| 9 | – | – | – | – | 3 | 0.7% |
| 11 | – | – | – | – | 5 | 1.1% |
| Fehlend / ungültig | – | – | – | – | 2 | 0.4% |