

DIPLOMARBEIT

Vergleich von modellgetriebenen Entwicklungsansätzen für Graphical User Interfaces

ausgeführt zur Erlangung des akademischen Grades
eines Diplom-Ingenieurs unter der Leitung von

Univ.Prof. Dipl.-Ing. Dr. Hermann Kaindl
Univ.Ass. Dipl.-Ing. Dr.techn. Roman Popp
Dipl.-Ing. David Raneburger

am

Institut für Computertechnik (E384)
der Technischen Universität Wien

durch

Simon Badalians Gholi Kandi
Matr.Nr. 0626305
Prinz Eugen Straße 46, 1040 Wien

Wien, am 30.4.2014

Kurzfassung

Das Erstellen von grafischen Benutzungsschnittstellen ist eine zeit- und kostenintensive Aufgabe. Eine mögliche Lösung bieten modellgetriebene Ansätze, die von einer abstrakten Spezifikation in Form von Modellen ausgehend Benutzungsschnittstellen für verschiedene Endgeräte generieren können. Diese Diplomarbeit präsentiert einen Vergleich etablierter Ansätze aus diesem Bereich.

Im Rahmen dieser Arbeit wurden konkret die drei Ansätze MARIA, UsiXML und UCP verglichen. Diese sind, nach einer Einführung in die Geschichte der modellgetriebenen Benutzungsschnittstellen, kurz vorgestellt und die entsprechenden Metamodelle erläutert. Damit der Leser die Theorie besser nachvollziehen kann, ergänzt zu jedem Ansatz ein praktisches Beispiel die vorgestellte Theorie. Anschließend folgt ein Vergleich der unterschiedlichen Ansätze. Dieser Vergleich setzt sich aus einem konzeptionellen Vergleich und einem praktischen Vergleich zusammen. Für den konzeptionellen Vergleich sind, neben der Klassifizierung anhand des bekannten Cameleon Reference Framework (CRF), neue Kriterien definiert, anhand derer die Ansätze ebenfalls verglichen sind. Diese zusätzlichen Kriterien sollen dazu beitragen, Eigenschaften aufzuzeigen bzw. zu vergleichen, die nicht im CRF-Schema dargestellt werden können. Im Anschluss an dem Kriterien-basierten Vergleich folgt ein praktischer Vergleich der Ansätze, bei dem auch ihre Ähnlichkeiten aufgezeigt werden. Für den praktischen Vergleich wurde im Rahmen dieser Arbeit eine zweite Anwendung implementiert, die auf einem W3C Use Case aufbaut.

Als Ergebnis dieser Vergleiche liegt eine Klassifizierung der untersuchten Ansätze basierend auf den eigens dafür definierten Vergleichskriterien vor. Diese basiert neben den Erkenntnissen aus der Überprüfung der Modelle und Werkzeuge auf jenen aus der Untersuchung der generierten Applikationen. Das Reengineering des W3C Use Cases vervollständigt den Vergleich dieser Ansätze.

Abstract

User Interface Development is one of the most time-consuming activities in Software Development. Novel approaches resulting from research activities in the last two decades are able to generate User Interfaces based on a specification of the Human-Machine Interaction. This master's thesis presents a comparison of established approaches. These approaches were MARIA, UsiXML and UCP, which are briefly introduced. This thesis also gives a snapshot of the Model-Driven User Interface Development (MDUID) history. The comparison of the selected approaches has two parts. The first part, a conceptual comparison, consists of a CRF-based and a criteria-based comparison. The second part contains practical aspects. While introducing these approaches, a running example is used to explain these and the available tools. The practical comparison also implies a W3C Use Case, which is implemented in MARIAE and UCP.

This work results in a classification of the approaches based on the defined criteria. This classification builds on the results of the comparative analysis, which relies on the evaluation of the models, tools and the generated Graphical User Interfaces. The Car Rental Use Case and its reengineering based on the given ConcurTaskTree completes the comparison.

Danksagung

Ich möchte hier die Gelegenheit nutzen und mich bei allen bedanken, die mich bei dieser Abschlussarbeit unterstützt haben.

Mein erster Dank gilt meinen Eltern und Geschwistern, die mich nicht nur finanziell, sondern auch moralisch jederzeit unterstützt haben.

Besonders bedanken möchte ich mich bei Herrn Prof. Hermann Kaindl und Herrn Dr. Roman Popp für die Ermöglichung dieser Arbeit. Ihre Kommentare und professionellen Anregungen haben sehr zur Bereicherung dieser Arbeit beigetragen. Allen voran bedanke ich mich bei Herrn Dipl.-Ing. David Raneburger, der mir jederzeit mit Rat und Tat zur Seite stand.

Einen großen Dank an meine Freundin Tamara, die mir immer zur Seite stand und mich bei allem unterstützte. Vielen Dank für deine Unterstützung und deinem Zuspruch!

INHALTSVERZEICHNIS

1	Einleitung	1
1.1	Motivation	1
1.2	Forschungsfragen und Ziel der Arbeit	2
1.3	Aufbau der Arbeit	3
2	Modellgetriebene Generierung von GUIs	4
2.1	Modellbasierte GUI-Entwicklung	4
2.2	Historischer Hintergrund	5
2.3	CAMELEON Reference Framework	6
3	CTT, MARIA und unterstützende Werkzeuge	10
3.1	Metamodelle	10
3.1.1	ConcurTaskTree (CTT)	10
3.1.2	MARIA-AUI	15
3.1.3	MARIA-CUI	19
3.2	Werkzeuge	19
3.2.1	CTTE	19
3.2.2	MARIAE	22
4	USer Interface eXtensible Markup Language (UsiXML)	27
4.1	Metamodelle	27
4.1.1	Task-Modell	27
4.1.2	Domain-Modell	30
4.1.3	Context-Modell	31
4.1.4	AUI-Modell	32
4.1.5	CUI-Modell	32
4.2	Werkzeuge	36
5	Unified Communication Plattform (UCP)	38
5.1	Metamodelle	39
5.1.1	Domain-of-Discourse Model	39
5.1.2	Action-Notification Model	40
5.1.3	Discourse Model	40
5.1.4	Generierungs-Framework für die GUIs (UCP:UI)	44
5.2	Werkzeuge	47

5.2.1	Editoren	47
5.2.2	UCP:UI Werkzeuge	49
5.2.3	UCP Runtime	49
6	Vergleich der unterschiedlichen Ansätze	52
6.1	Konzeptioneller Vergleich	52
6.1.1	CRF-basierter Vergleich	52
6.1.2	Vergleichskriterien	55
6.1.3	Kriterien-basierter Vergleich	57
6.2	Praktischer Vergleich	69
6.2.1	Login	70
6.2.2	Car Rental	70
7	Diskussion und Zusammenfassung	78
7.1	Diskussion	78
7.2	Zusammenfassung	79
A	Car Rental Task-Baum	81
	Wissenschaftliche Literatur	87

ABKÜRZUNGEN

ANM	Action-Notification Model
AUI	Abstract User Interface
CAMELEON	Context Aware Modelling for Enabling and Leveraging Effective interactiON
CRF	CAMELEON Reference Framework
CTT	ConcurTaskTrees
CTTE	CTT Environment
CUI	Concrete User Interface
DoD	Domain of Discourse
DSL	Domain-Specific Language
EMF	Eclipse Modeling Framework
FUI	Final User Interface
GOMS	Goals, Operators, Methods and Selection
GUI	Graphical User Interface
HCI	Human Computer Interaction
HTA	Hierarchical Task Analysis
IU	Interaction Unit
MARIA	Model-based lAnAge foR Interactive Applications
MARIAE	MARIA Environment
MBUID	Model-Based User Interface Development
MDA	Model Driven Architecture
MDSD	Model-Driven Software Development
MDUID	Model-Driven User Interface Development
OMG	Object Management Group
PIM	Platform-Independent Model
UCP	Unified Communication Plattform
UI	User Interface
UIMS	User Interface Management System
UML	Unified Modeling Language
UsiXML	User Interface eXtended Markup Language
WADL	Web Application Description Language
WIMP	Windows, Icons, Menus und Pointer
WSDL	Web Services Description Language

1 EINLEITUNG

1.1 Motivation

Software ist ein wichtiger Innovationsfaktor in technischen Produkten. Seit dem Erscheinen der ersten höheren Programmiersprachen, welche die Entwicklung von komplexerer Software auf einer abstrakten Ebene erlaubten, wurden Softwaresysteme immer komplexer. Die Entwicklung solcher komplexer Softwaresysteme ist eine sehr zeit- und kostenaufwendige Arbeit. Die Wartung und Fehlerbehebung von Softwareprodukten ist mit Abstand der aufwendigste und teuerste Teil des Softwarelebenszyklus. Hierbei trägt das User Interface (UI) einen hohen Anteil bei [MR92]. In Zusammenhang mit UIs bilden die Heterogenität der heutigen IT-Landschaft (z.B. Endgeräte, Interaktions-Modalitäten, Programmiersprachen, ...) und deren Benutzer einen zusätzlichen Komplexitätsfaktor.

Als eine mögliche Lösung für diese Probleme bietet sich eine modellgetriebene Vorgehensweise an. Modellgetriebene Paradigmen erlauben dem Anwender auf einer hohen Abstraktionsebene eine Software-Anwendung zu entwickeln. Dies kann z.B. mit Hilfe einer domänenspezifischen Sprache erfolgen, mit der Softwaremodelle definiert werden können. Diese Modelle ermöglichen es dem Entwickler, die Lösung einer Problemstellung auf einer Ebene zu spezifizieren, auf der die Details der Implementierung unwesentlich sind. Im nächsten Schritt können diese Modelle mit unterschiedlichen Transformationsansätzen auf andere Modelle abgebildet werden und damit inkrementell an die spezifischen Vorgaben bezüglich Anwender und Plattform angepasst werden [MCC14].

Die Softwaretechnik bewegt sich, als eine relativ junge Ingenieurwissenschaft, immer mehr in Richtung modellgetriebener Ansätze. Untermuert wurde dies, als im Jahre 2003 die Object Management Group einen Framework zum Thema Model Driven Architecture (MDA) publizierte [MM03]. Immer mehr moderne Anwendungen nutzen dieses Framework als Basis ihrer Architektur.¹ Modellgetriebene Ansätze haben potenziell zahlreiche Vorteile gegenüber traditionellen Entwicklungsmethoden. Nachdem die Entwicklung der Software basierend auf sogenannten Eingangsmodellen und den auf diesen Modellen ausgeführte Transformationen erfolgt, ermöglichen diese Ansätze, eine Spezifikation der Software auf einer höheren Abstraktionsebene und unterstützen eine bessere Wiederverwendbarkeit. Zusätzlich erreicht man eine höhere Qualität des Quell-Codes, da dieser automatisiert durch einen Code-Generator von einem vordefinierten Modell abgeleitet wird. Ein Bereich der Software-Entwicklung, der sich für eine modellgetriebene Vorgehensweise eignet, ist die Entwicklung von User Interfaces (UI).

¹http://www.omg.org/mda/products_success.htm

In den letzten zwei Dekaden haben sich viele Wissenschaftler damit beschäftigt, geeignete Modelle für die automatische Generierung von UIs zu definieren. Als Ergebnis dieser Bemühungen entstanden verschiedene Ansätze, von denen es jedoch bis heute wenige zu einem industriellen Einsatz schafften. Mendix², WebRatio [ABB⁺08] und OutSystems³ zählen zu den Werkzeugen, die in der Industrie Einsatz finden. Diese erlauben eine Daten-basierte Generierung von GUIs. Die GUI-Generierung basiert in diesen Ansätzen auf gerätespezifischen GUI-Modellen und nicht auf geräteunabhängigen Interaktionsmodellen. Die aktuell leistungsfähigsten Lösungen bieten die Möglichkeit, mit domänenspezifische Sprachen (sogenannten Domain Specific Languages, auch DSL) die Interaktionen zwischen Mensch und Maschine zu modellieren, um basierend auf diesen Modellen die zu generierenden UIs auf verschiedenen Abstraktionsebenen zu beschreiben.

1.2 Forschungsfragen und Ziel der Arbeit

Die modellgetriebenen Ansätze für die Generierung von UIs erfreuen sich eines immer weiterwachsenden Interesses, und Technologie-Forschungseinrichtungen rechnen damit, dass diese Lösungen in den kommenden Jahren, auch in der Industrie breiteren Einsatz finden. Beispiele hierfür sind z.B. die Automotive-Branche⁴. Obwohl in den letzten Jahren eine Vielzahl an verschiedensten Ansätzen vorgestellt worden ist, fehlt es dem Anwender noch an einem Vergleich dieser Ansätze. Gemeinsam ist all den hier untersuchten Ansätzen, dass sich alle zum Ziel gesetzt haben, eine UI in erster Instanz auf einer abstrakten Ebene zu definieren, um hiermit eine Technologie-unabhängige Spezifikation der Interaktion zu erlauben. Welche Ähnlichkeiten bzw. Unterschiede zwischen diesen Ansätzen vorliegen, ist die **Forschungsfrage**, mit der sich diese Arbeit auseinandersetzt.

Ziel dieser Arbeit ist ein Vergleich verschiedener Ansätze für die modellgetriebene GUI-Entwicklung. Dieser Vergleich besteht einerseits aus einem konzeptionellen Teil, in dem unter anderem das Cameleon Reference Framework (CRF) herangezogen wird und andererseits aus einem praktischen Teil. Hiermit soll eine Antwort auf die Forschungsfrage gefunden werden.

Für den Vergleich wurden die Ansätze MARIA, UsiXML und UCP herangezogen. MARIA und UsiXML gehören zu den am meisten eingesetzten Ansätzen⁵ und bilden die Basis für einen W3C-Standard der Model-Based UI Working Group [PSSR]. Sie basieren beide auf einer Task-basierten Modellierung der Interaktion zwischen Mensch und Maschine. Der UCP-Ansatz setzt im Gegensatz zu den anderen zwei Ansätzen auf ein Diskurs-basiertes Kommunikationsmodell und bietet hiermit eine Alternative zu Task-Modellen. Diskurs-basierte Kommunikationsmodelle erlauben die Modellierung der Interaktion zwischen zwei Kommunikationspartnern. Dabei kann es sich bei der Kommunikation um eine Mensch-Maschine oder eine Maschine-Maschine Kommunikation handeln.

In dieser Arbeit wird bei der Vorstellung der Ansätze auf CRF gesetzt. Da basierend auf diesem Framework nicht alle Eigenschaften der Ansätze aufgezeigt werden können, werden zusätzliche Vergleichskriterien benötigt. Eine der wichtigsten Fragen in diesem Zusammenhang ist, welche Kriterien zum Vergleich der Ansätze in Betracht gezogen werden sollen. Bei dem konzeptionellen Vergleich wurden die Eingangsmodelle, Methoden und Werkzeuge als wichtigste Kriterien festgelegt. Da die Methoden in der Praxis stark mit den Werkzeugen verknüpft sind, wird in dieser

²<http://www.mendix.com>

³<http://www.OutSystems.com>

⁴<http://www.automotive-hmi.org/>

⁵<http://giove.isti.cnr.it/tools/CTTE/external>

Arbeit explizit nur auf die Modelle und Werkzeuge eingegangen. Zusätzlich wurden noch zwei weitere Merkmale in dem Vergleich mit berücksichtigt, nämlich die allgemeinen Eigenschaften der Ansätze und die Eigenschaften der erzeugten GUIs. Als ein weiteres Ergebnis der Arbeit wird erwartet, dass die Leser in die Lage versetzt werden, die verglichenen Ansätze so eindeutig wie möglich zu klassifizieren und für eine bestimmte Anwendung den bestgeeigneten Ansatz zu wählen.

Für den praktischen Vergleich der Ansätze werden zwei Beispiele herangezogen. Das erste Beispiel stellt ein „Login“-Beispiel dar und wird als Running Example eingesetzt. Dieses Beispiel soll die Vorstellung und die Erklärung der einzelnen Ansätze unterstützen. Das zweite Beispiel besteht aus einem Car Rental-Szenario, das im Zuge des SERENOA⁶ Projects definiert und im W3C-Standard für Modell-basierte User Interface Entwicklung als Use Case verwendet wurde.

1.3 Aufbau der Arbeit

Nach der Einleitung in Kapitel 1 wird dem Leser in Kapitel 2 eine theoretische Einführung in die modellgetriebene Generierung von UIs (Model-Driven User Interface Development, MDUID) und deren Evolution gegeben. In diesem Abschnitt wird auf die historische Entwicklung in diesem Bereich eingegangen und auf das CRF. CRF bildet heute das Standard-Framework im Bereich der MDUID. Es kategorisiert die verschiedenen Möglichkeiten, die im Design Time Process und im Run Time Process dem Entwickler eines UIs zur Verfügung gestellt werden müssen.

In den drei darauffolgenden Kapiteln (3-5) werden die drei zu vergleichenden Ansätze vorgestellt und deren Funktionalitäten anhand des Login-Beispiels erläutert. Hierbei werden im ersten Schritt die Metamodelle dieser Ansätze vorgestellt, um darauffolgend die Transformationen zu erläutern und die Werkzeuge vorzustellen, die diese Ansätze bereitstellen. Die Grundlage dieser Vorstellung bildet das CRF. Es bietet einen ausgezeichneten Ansatzpunkt zur Vorstellung der Ansätze an, da es auf eine übersichtliche Art und Weise die Struktur und teilweise auch die Dynamik der Ansätze wiedergibt.

Kapitel 6 beschäftigt sich mit dem Vergleich der Ansätze. Dabei werden diese aus zwei verschiedenen Blickwinkeln verglichen: Einerseits erfolgt ein konzeptioneller Vergleich, welcher auf CRF und zusätzlichen Kriterien basiert, die in diesem Kapitel definiert werden. Andererseits wird ein praktischer Vergleich durchgeführt. Bei diesem werden die Erfahrungen aus der Implementierung des Login- und des Car Rental-Beispiels präsentiert. Das Car Rental-Beispiel wurde durchgeführt, um neben den Unterschieden der Ansätze auch deren Gemeinsamkeiten darzustellen.

In Kapitel 7 werden die erzielten Ergebnisse zusammengefasst und eine Analyse des weiteren Forschungsbedarfes im Bereich der modellgetriebenen UI-Entwicklung durchgeführt.

⁶<http://www.serenoa-fp7.eu/>

2 MODELLGETRIEBENE GENERIERUNG VON GUIs

Da wir uns in dieser Arbeit mit modellgetriebenen Ansätze für die Entwicklung von GUIs beschäftigen, soll hier eine Einführung in die Konzepte gegeben werden, auf denen die in dieser Arbeit untersuchte Ansätze aufbauen.

2.1 Modellbasierte GUI-Entwicklung

In einer Studie von Myers und Rosson [MR92] nahm die Entwicklung des UIs ungefähr 45% der Entwicklungszeit, 50% der Implementierungszeit und 37% der Wartungszeit in Anspruch. Laut dieser Studie sahen sich die Entwickler von UIs während der Entwicklung und Implementierung unter anderem mit den folgenden Problemen konfrontiert:

- Analyse von nicht-funktionalen Anforderungen
- Implementierung von Hilfe-Texten
- Erlernen der angewandten Werkzeuge
- Performance

Diese Probleme sind heute im gleichen Ausmaß wiederzufinden. Verschärft werden diese Probleme, wenn man sich vor Augen hält, in welchem Ausmaß sich die Interaktion zwischen Mensch und Maschine im täglichen Leben der Menschen etabliert hat. Die vielfältigen, modernen Einsatzbereiche der Computertechnik [MCC14] zwingen die Entwickler, sich immer mehr mit neuen Lösungsansätzen zu beschäftigen, die sowohl entwicklerfreundlich als auch kostengünstig und qualitätsbezogen sind.

Als eine Lösung für die oben erwähnten Problemstellungen, bieten sich die modellbasierte UI-Entwicklung („Model-Based User Interface Development(MBUID)“) bzw. die modellgetriebene UI-Entwicklung („Model-Driven User Interface Development(MDUID)“) an.

Bei modellbasierter UI-Entwicklung wird das Modell nur zwecks Spezifikation eingesetzt und muss nicht direkt mit der Implementierung übereinstimmen. Dies ist gleichzeitig einer der Nachteile dieses Ansatzes. Dadurch kommt es schnell zu Inkonsistenzen zwischen der Spezifikation und der

Implementierung, da die implementierte Logik nicht aus dem Eingangsmodell abgeleitet wird. Das bedeutet, dass bei dieser Vorgehensweise die Transformationen fehlen, die für die Konsistenz zwischen dem Eingangsmodell und den generierten Zwischenartefakten sorgen. Im Gegensatz zur modellbasierten Software-Entwicklung, sind bei einem modellgetriebenem Ansatz das aufgestellte Modell und die Transformationen, die dieses Modell in bestimmte andere Modelle überführen, der Kern der Vorgehensweise und das generierte Code basiert vollkommen auf dem Eingangs-Modell. Das bedeutet, dass das Modell durch Transformationen in den Quell-Code übergeführt wird. Man erwartet sich von MDUID folgende Vorteile gegenüber traditionellen Ansätzen:

- Bessere Handhabung von Komplexität durch Abstraktion
- Bessere Wartbarkeit und Wiederverwendbarkeit der generierten Anwendungen
- Höhere Qualität des generierten Codes, da die Generierung der unterschiedlichen Module immer auf einem Generator basiert
- Verkürzte Entwicklungszyklen

Im konkreten Fall der modellgetriebenen Entwicklung von GUIs wird es dem Entwickler ermöglicht, seine Konzentration hauptsächlich auf die abstrakte Beschreibung der Interaktion zu setzen und weniger auf die Herausforderungen, die durch die Implementierung entstehen.

Im nächsten Abschnitt soll dem Leser, bevor die modellbasierten und modellgetriebenen Ansätze genauer erläutert werden, ein Überblick über die Plattformen bzw. Frameworks für die Entwicklung von GUIs gegeben werden.

2.2 Historischer Hintergrund

Die User Interface Management Systems (UIMS) waren die erste Generation an Tools, die den Entwickler bei seiner Arbeit unterstützen sollten. Die ersten Ansätze zur Entwicklung von UIMS stammen aus den 1980er Jahren. Laut einer Definition von [BBF⁺87] [MPV11] ist ein UIMS ein Tool, welches die interdisziplinäre Arbeit der Entwicklung, Anpassung und Management von Interaktionen für eine Applikationsdomäne unterstützen soll, in der verschiedene Endgeräte und Interaktionsarten zum Einsatz kommen.

Die UIMS konnten sich laut Myers [B.87] hauptsächlich aus drei Gründen nicht durchsetzen:

1. UIMS waren für Nichtprogrammierer schwer verständlich und dementsprechend waren die Einlernzeiten zu lang.
2. Viele UIMS haben nur ein begrenztes Spektrum an „Windows, Icons, Menus, Pointer (WIMP)“-GUIs unterstützt.
3. Da UIMS auf einer sehr niedrigen Abstraktionsebene arbeiteten (die Tools mussten sich um plattformabhängige Themen wie IO-Zugriffe kümmern), war keine Portabilität der GUIs gegeben.

Anfang der 1980er Jahre fanden die ersten Model-Based User Interface Development (MBUID) Plattformen ihren Einsatz. Eine der treibenden Kräfte hinter der Evolution von UIMS zu MBUID-Systemen war die Entwicklung von Programmiersprachen, die eine Definition von komplexeren UIs erlaubten. Objekt-orientierte Programmiersprachen sind ein Beispiel für solche Sprachen. Diese Sprachen wurden in den MBUID-Systemen eingesetzt um komplexere UIs zu entwerfen [PS94][Sze96][MPV11].

Eine MBUID-System muss folgende zwei Voraussetzungen erfüllen [E.96]:

1. Es muss ein abstraktes, explizites Modell definieren, womit man die Interaktion mit einem System modellieren kann.
2. Es muss die notwendigen Transformationen zur Verfügung stellen, um aus dem abstrakten Modell die lauffähige UI zu generieren.

In [MPV11] werden die MBUID-Systeme in vier Generationen unterteilt. Die erste Generation hatte sich die Erkennung und Abstrahierung der wesentlichen Aspekte von UIs zum Ziel gesetzt. Diese Ansätze versuchten eine vollkommen automatische Generierung von UIs zu ermöglichen. UIDE [JF89] und HUMANOID [SLN92] sind zwei Beispiele für die Ansätze aus der ersten Generation. Die Ansätze der zweiten Generation beschäftigten sich hauptsächlich mit der Erweiterung von bestehenden UI-Modellen. Dabei strukturierten sie die UIs in mehrere Sub-Modelle wie z.B. ein Task-Modell, Dialog-Modell oder Präsentations-Modell. Beispiele für die Ansätze aus der zweiten Generation sind TRIDENT [BHL+95] und MASTERMIND [BDRS96]. Die Anforderungen an die verschiedenen Generationen waren nicht selten auch durch unterschiedliche technologische Trends getrieben. Da zum Beispiel die dritte Generation an MBUIDs in der Zeitspanne weiterentwickelt wurde, als Smartphones die mobilen Rechnermärkte eroberten, beschäftigten sich diese hauptsächlich damit die UIs unabhängig von dem eingesetzten Gerät zu generieren und einmal entwickelte Modelle, für verschiedenste Geräte wie z.B. PDAs, Smartphones und andere Geräte einsetzbar zu machen. TERESA [MPS04] gehört zu den Ansätzen aus dieser Generation. Laut [MPV11] bezeichnet man die Ansätze, die ab dem Jahr 2004 entsandt sind als die vierte Generation der MBUID-Ansätze. Der Fokus der vierten Generation liegt an sogenannte „Context-Sensitive“ UIs für eine Mehrzahl an Endgeräten und Modalitäten.

In dieser Arbeit beschäftigen wir uns hauptsächlich mit den Ansätzen aus der vierten Generation und der Evaluierung jener Funktionalitäten, die durch die Evolution der MBUID-Systemen, in der vierten Generation enthalten sein müssen. So wie auch in [MPV11] erwähnt, spricht man heute im Bereich der modellbasierten Benutzungsschnittstellen von MBUID als ein Äquivalent zum MDUID. Dies hat den Grund, dass die vierte Generation der MBUID-Systeme schon den Eigenschaften einer MDUID genügen und man in diesem Fall eher von modellgetrieben sprechen muss als von modellbasiert. Trotzdem halten wir uns in dieser Arbeit durchgehend an die Bezeichnung „MDUID“.

2.3 CAMELEON Reference Framework

Das CAMELEON Reference Framework (CRF) wurde im Zuge des CAMELEON-Projektes¹ spezifiziert. Dieses Projekt beschäftigte sich hauptsächlich mit ontologischen Aspekten des „Multi-Targeting“ für UIs. Ziel des CRFs war ein Standard für die Designer und Entwickler von UIs.

¹<http://giove.isti.cnr.it/projects/cameleon.html>

Damit sollte ihnen die Möglichkeit gegeben werden die Schnittstellen auf unterschiedlich abstrakten Ebenen beschreiben zu können. Diese Umgebung sollte den Entwickler bei der Klassifizierung der unterschiedlichen Modellen sowohl während des Design Time Process als auch während des Run Time Process von UIs unterstützen. Zu diesem Zweck wurden für diese beiden Prozessen unterschiedliche Modelle definiert. Hierbei werden die Modelle in einer ersten, abstrakten Kategorisierung in folgende drei Gruppen unterteilt:

- „Ontological Models“: stellen die Metamodelle dar. Die Archetypal Models und Observed Models sind Modelle, die auf diesen Metamodellen basieren.
- „Archetypal Models“: Als Instanzen des ontologischen Metamodells bilden diese Modelle Klassen von effektiven Modellen, die für unterschiedliche Kontexte gelten können. Diese Modelle unterstützen den Anwender während des Design Time Process von Multi Target UIs.
- „Observed Models“: Dies sind die Modelle, die während der Laufzeit effektiv vorliegen. Im Gegensatz zu Archetypal Modellen sind diese vom Abstraktionsgrad auf einer niedrigeren Ebene und passen nur zu einem bestimmten Betrachtungsfall. Sie unterstützen damit die Adaptions-Prozesse während der Laufzeit.

Diese drei Modelle sind in der Abbildung 2.1 dargestellt.

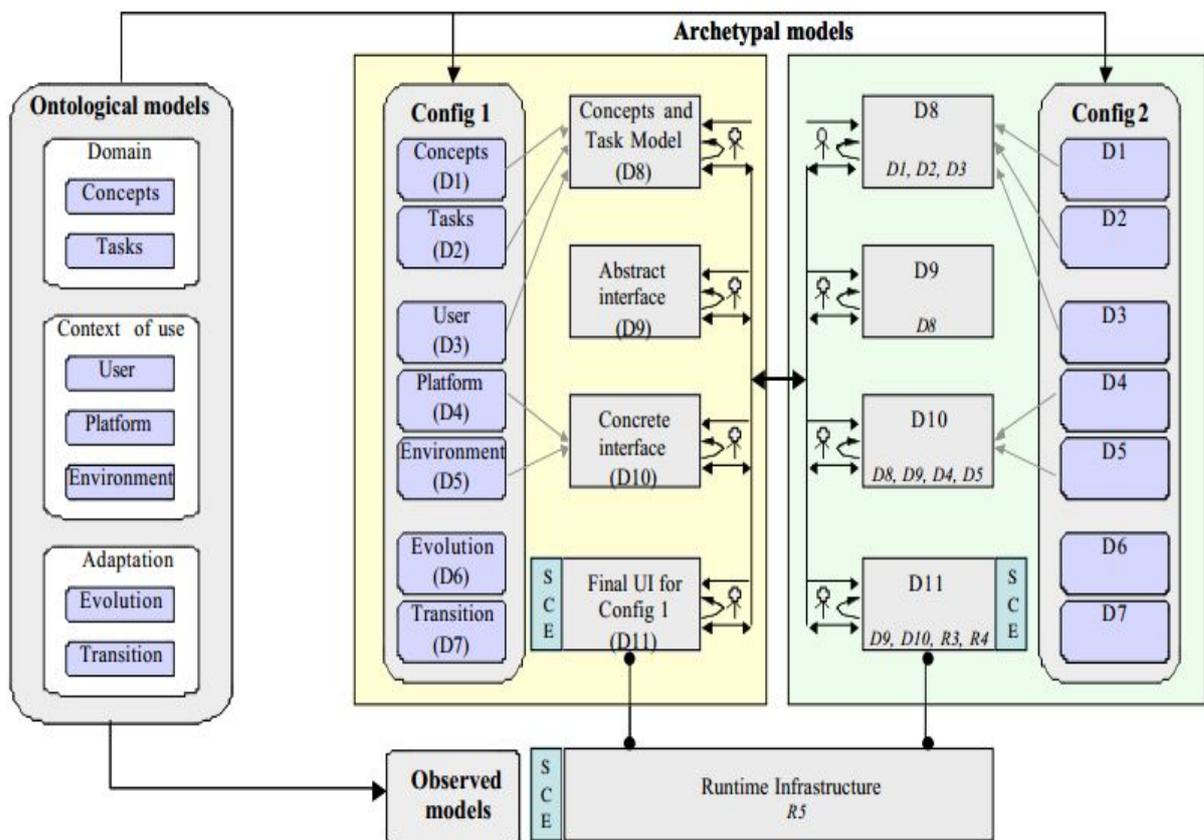


Abbildung 2.1: Modelle im CAMELEON Reference Framework (kopiert von [CClvoV+02])

Als ein Beispiel kann man sich folgenden Fall vorstellen: Ein Entwickler kann bei der Erstellung des UIs für ein mobiles Tablet-Gerät, von einem CRF-konformen Ansatz sowohl für den Design Time Process als auch für den Run Time Process entsprechende Metamodelle erwarten. Ein Archetypal Modell liefert Klassen von Modellen, die für Geräte verschiedener Hersteller gelten. Dabei kann man für ein interaktives System mehrere Archetypal Modelle instanziiieren, die verschiedene Einsatzkontexte wiedergeben. Um diese Modelle zu definieren, benutzt man Metamodell-Objekte, die man unter dem Begriff der „Ontological Models“ definiert hat. Die „Observed Models“ sind während der Laufzeit auf bestimmten Geräten, innerhalb bestimmter Kontexte, beobachtete Modelle. Diese Modelle können entweder bei einem oder bei mehreren Archetypal Kontexten verfügbar sein.

Der wichtigste Ansatz dieses Frameworks, der im Zuge dieser Arbeit als eine Referenz für den Vergleich von Strukturen der verschiedenen Ansätze herangezogen wird, ist die Konkretisierung der Modelle des Design Time Process. Ausgehend von einer abstrakten Stufe des „Archetypal Model“, werden die Modelle schrittweise konkretisiert. Diesen Prozess kann man sich als eine schrittweise erfolgende Transformation vorstellen, die in Abbildung 2.2 wiedergegeben ist.

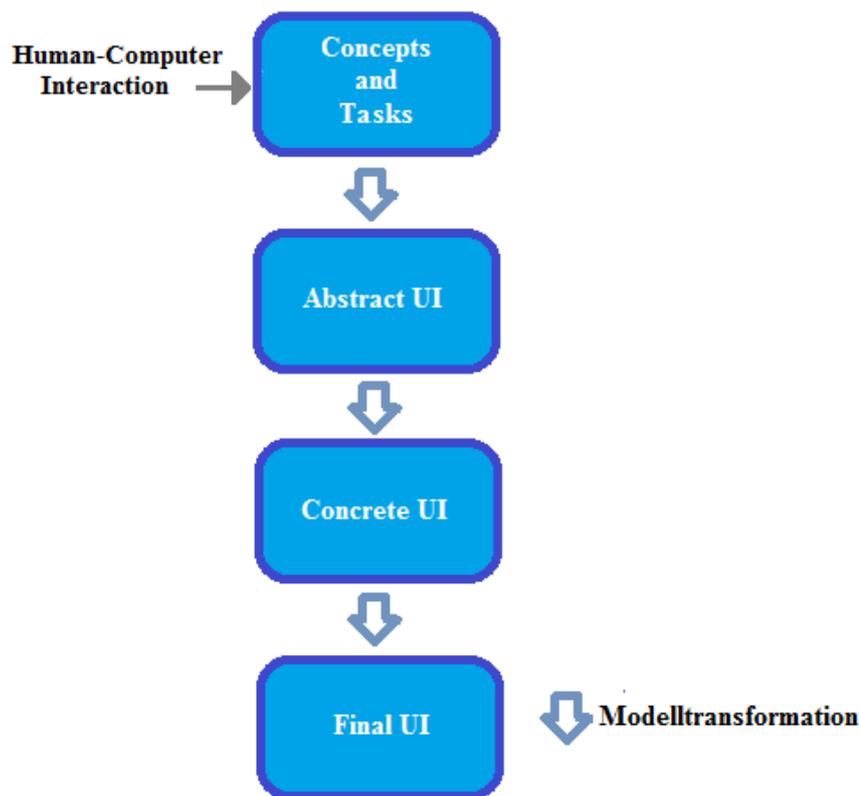


Abbildung 2.2: CAMELEON Reference Framework für einen Design Time Process

Die oben erwähnten Archetypal-Modelle bestehen aus folgenden vier Abstraktions-Ebenen:

- **Concepts and Tasks:** In diesem Teil wird abhängig vom Kontext der Anwendung ein Modell für die Aufgaben (Tasks) aufgestellt, welches die Absichten eines Anwenders wiedergibt. Diese Abstraktionsebene wird an manchen Stellen zusätzlich mit einem Domänenmodell ergänzt.

- **Abstract User Interface (AUI):** Als AUI zusammengefasst, gibt es eine erste, sehr abstrakte Realisierung des UIs wieder, die unabhängig von der Plattform ist. Sie ist eine abstrakte Darstellung von Eingaben und Ausgaben, die keine Informationen über die Modalität der Interaktionen enthält.
- **Concrete User Interface (CUI):** Als CUI zusammengefasst, konkretisiert es den im AUI definierten UI weiter und baut erste modalitätsabhängige und konkrete Objekte ein.
- **Final User Interface (FUI):** Dies ist der ausführbare Code, welcher Technologie-abhängig ist und mit der Geschäftslogik verbunden werden muss. Eine FUI kann entweder kompiliert oder interpretiert werden.

Im CRF werden auch die Relationen definiert, die diese Abstraktionsebenen miteinander verbinden. Diese sind die *Reification*, *Abstrahierung*, *Translation* und die *Reflexion*. Unter dem Begriff der *Reification* versteht man einen Übergang (Transformation) von einer abstrakteren Beschreibungsebene im „Archetypal Model“ zu einer konkreteren Beschreibung. Die *Abstrahierung* stellt eine Transformation dar, die eine Beschreibung in eine abstraktere Beschreibung im Rahmen des gleichen Kontextes umwandelt. Unter einer *Translation* jedoch wird eine Umwandlung eines Modells von einer Konfiguration bzw. von einem Kontext in einem anderen verstanden. Man kann sich eine Translation auch folgendermaßen vorstellen: Angenommen, ein Entwickler hat mit Hilfe eines Modelles eine abstrakte Definition der Aufgaben in der Ebene der „Concepts and Tasks“ aufgestellt. Mit Hilfe einer Transformation gelangt der Entwickler zu dem AUI und konkretisiert seinen Entwurf hiermit. Wünscht sich der Entwickler oder Designer das gleiche Verhalten für eine andere Konfiguration zu modifizieren, bietet ihm eine Translation die entsprechende Möglichkeit dazu. Die *Reflexion* transformiert ein bestimmtes Modell in ein anderes Modell der gleichen Ebene, für den gleichen Anwendungskontext. Dementsprechend können die oben erklärten vier Ebenen, die während des Entwurfes dem Entwickler eine bessere Strukturierung ermöglichen sollen, Operanden dieser Transformationen sein. Um Transformationen von einer Ebene der Abstraktion in eine andere Ebene eines unterschiedlichen Kontextes durchzuführen, bietet sich der „Cross Cutting“ Operator [LV09] an.

Für eine ausführliche Dokumentation des CRF und mehr Informationen wird hier auf [CCLvoV+02] und [CCT+03a] verwiesen.

3 CTT, MARIA UND UNTERSTÜTZENDE WERKZEUGE

MARIA (Model-based Language foR Interactive Applications) [PSS09] wurde im Human Interfaces in Information Systems (HIIS) Laboratory der Universität Pisa entwickelt. Folgende Abschnitte beschäftigen sich im Detail mit den Modellen und den verschiedenen Werkzeugen dieses Ansatzes. MARIA hat einen Vorgänger namens TERESA [MPS04]. Die Entwickler haben hiermit auf die Erfahrungen aus dem Vorgängerprojekt zurückgreifen können. ConcurTaskTree (CTT) umfasst die Modelle, mit denen in MARIA die Ebene der Concepts and Tasks unterstützt wird. Mit Hilfe von CTT können die Interaktionen in Form eines Task-Baumes modelliert werden. Eine Instanz des CTT-Metamodells wird in dieser Arbeit als *Task-Baum* bezeichnet. Zum Erstellen eines Task-Baumes stellt MARIA das CTTE Tool (CTT Environment) zur Verfügung. Die UIs, die in den nächsten Schritten aus dem CTT-Modell durch Transformationen (Reification) hergeleitet werden, werden in MARIA XML beschrieben. MARIA XML definiert hiermit die Sprachelemente, mit denen die AUI- und CUI-Ebene modelliert werden. Für diese Ebenen ist in MARIA das Tool MARIAE (MARIA Environment) vorgesehen, welches die benötigten Transformationen zur Verfügung stellt, mit denen die Modelle der unterschiedlichen CRF-Ebenen generiert werden können.

3.1 Metamodelle

Die folgende Abschnitte beschäftigen sich mit den Metamodellen, welche verwendet werden, um die verschiedenen Modelle in MARIA zu definieren. Dabei wird zuerst das CTT-Metamodell vorgestellt, um anschließend den Aufbau von AUIs und CUIs in MARIA zu erläutern. CTT definiert in MARIA die Elemente der Sprache, welche die Interaktionen zwischen Mensch und Maschine modellieren. MARIA XML ist die Sprache, die nach der Transformation des CTTs in Abstract UIs und Concrete UIs verwendet wird, um Benutzungsschnittstellen im Rahmen des MARIA Ansatzes zu beschreiben.

3.1.1 ConcurTaskTree (CTT)

In MARIA werden mit dem CTT-Modell die Interaktionen zwischen Mensch und Maschine modelliert. Dabei bilden zwei Elemente den Kern des Metamodells für den CTT. Diese sind die Tasks und die temporalen Operatoren. Mit Hilfe dieser Operatoren können die Tasks miteinander

verbunden werden, um letztendlich das von dem Anwender beabsichtigte abstrakte Task durchzuführen. Die abstrakten Tasks können entweder eine Änderung des Systemzustandes oder nur dessen Abfrage beinhalten. Diese abstrakten Tasks werden, basierend auf den Ideen der „Hierarchical Task Analysis (HTA)“, in Form einer Baumstruktur in mehrere kleine Teilaufgaben zerteilt [Pat03]. Hierbei stellen die Endknoten, Aktionen und die Zwischenknoten Gruppierungen der abstrakten Interaktions-Elemente dar. Im Folgenden wird das Task-Modell genauer betrachtet und die verschiedenen Kategorien der Tasks genauer erläutert.

Im CTT-Modell gibt es vier verschiedene Task-Kategorien. Diese sind die *User Tasks*, die *Abstraction Tasks*, die *Application Tasks* und die *Interaction Tasks*. Abbildung 3.1 stellt die Symbole zu diesen Tasks dar, die auch in dieser Form in CTTE vorgefunden werden.



Abbildung 3.1: Task-Kategorien in CTTE

Diese Tasks erlauben dann, die Interaktion zwischen Mensch und Maschine zu modellieren. Die Aufgabe, all diese relevanten Tasks zu identifizieren, wird auch als Task Analysis bezeichnet [Pat03]. Im Gegensatz zu Task Analysis, werden bei der Task-Modellierung die Beziehungen zwischen diesen Tasks auch mitberücksichtigt.

Jede Kategorie der oben erwähnten Tasks dient der Modellierung einer bestimmten Art von Anwenderaktion und drückt damit eine bestimmte Semantik aus. Ein User Task gibt so wie der Name schon suggeriert, eine kognitive-Aktion wieder, die nur der Anwender durchführt. Ein Application Task ist eine Aktion, die vollkommen von einer Anwendung übernommen und ausgeführt wird. Diese Task-Kategorie wird in MARIAE im Gegensatz zum CTTE als ein Task der Kategorie *system* bezeichnet. Die Ausführung eines Interaction Tasks impliziert eine direkte Interaktion des Users mit der Maschine. Ein Abstraction Task dient der Strukturierung der ConcurTaskTrees und trägt zur Gruppierung der Interaktionsobjekte in AUI bei.

Im CTT-Modell unterscheidet man in Hinsicht auf Tasks zwischen Task-Kategorien und Task-Typen. Nachdem man in einem Task-Baum die Kategorie der Tasks festgelegt hat, kann man als zusätzliche Einstellung dem Task einen Typ zuordnen. Damit kann man die semantische Implementierung eines Tasks verfeinern. Ein Beispiel für einen Task-Typ wäre z.B. der Typ „Control“, der in der Login-Anwendung dem Interaction Task „Trigger Login“ zugeordnet wird. Hiermit modelliert man, dass ein Interaktionsobjekt eine Steuerfunktion übernimmt. Zusätzlich kann man in CTT die Aufruffrequenz eines Tasks und die Zielplattform spezifizieren und feststellen, ob es sich um einen iterativen oder optionalen Task handelt. Es ist im Metamodell ein Feld für die zeitliche Performanz eines Tasks vorgesehen, mit dem der Anwender für die Ausführungszeit eines Tasks Plausibilitätsgrenzen definieren kann.

Als zweitem Bestandteil des CTT-Modells soll auf die Operatoren eingegangen werden, mit denen die Vorstellung des CTT-Modells vervollständigt wird. Das CTT-Modell [Pat03] bietet folgende Operatoren, um die in der Task-Analyse identifizierten Tasks miteinander zu verknüpfen:

- Enabling (Sequential Enabling): „Specifies second task cannot begin until first task performed“.
- Choice: „Specifies two tasks enabled, then once one has started the other one is no longer enabled“.
- Enabling with Information passing: „Specifies second task cannot be performed until first task is performed, and that information produced in first task is used as input for the second one“.
- Concurrent Tasks (Interleaving): „Tasks can be performed in any order, or at same time, including the possibility of starting a task before the other one has been completed“.
- Concurrent Communicating Tasks (Synchronization): „Tasks that can exchange information while performed concurrently“.
- Order Independence: „Tasks can be performed in any order, but when one starts then it has to finish before the other one can start“.
- Disabling: „The first task (usually an iterative task) is completely interrupted by the second task“.
- Suspend Resume: „First task can be interrupted by the second one. When the second terminates then the first one can be reactivated from the state reached before“.

Abbildung 3.2 zeigt die Symbole, die in CTTE für diese Operatoren definiert sind. In der oben ausgeführten Liste wurde eine Gruppe an Operatoren nicht berücksichtigt. Bei diesen Operatoren handelt es sich um jene, die mit der Modellierung von kooperativen Tasks zusammenhängen. Diese Operatoren wurden nicht berücksichtigt, da das Thema der kooperativen Prozesse im Rahmen dieser Arbeit nicht behandelt wird. Wir kommen am Ende dieses Abschnittes noch einmal auf die kooperativen Prozesse in CTT zurück.

[]	Choice
⌋	Order Independence
	Interleaving
[[]]	Synchronization
[>	Disabling
>	Suspend Resume
>>	Sequential Enabling
[] >>	Enabling with Information passing

Abbildung 3.2: Temporale (High Priority) Operatoren in CTTE

Abbildung 3.3 stellt einen Task-Baum für die Login-Anwendung dar. Von welchem Typ eine Task-Kategorie ist, kann einem graphischen CTT-Modell nicht entnommen werden und ist daher auch in 3.3 nicht eingezeichnet. Angefangen von einem Element *Login&DataAccess*, welches durch einen Abstraction Task dargestellt wird, verfeinert sich das Task-Modell in den Tasks *Authentication*, *Login Deny* und *Access*. Hierbei stellen *Authentication* und *Access* abstrakte Tasks dar.

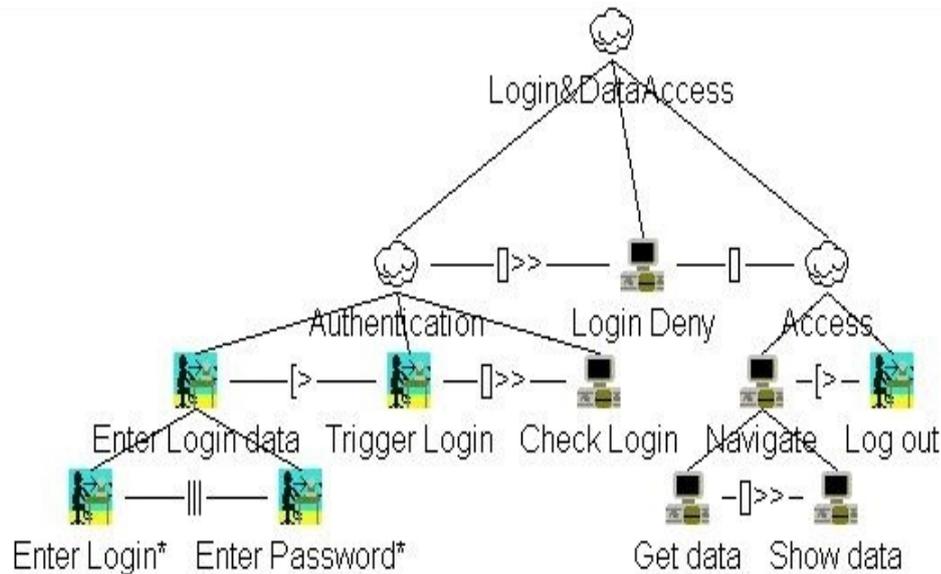


Abbildung 3.3: CTT-Modellierung der Login-Anwendung

In der ersten Abstraktion werden die Subtasks zusammengefasst, die ein Einloggen des Anwenders ermöglichen. Dabei werden die zwei Tasks zur Eingabe des Benutzernamens und des Passworts als interaktive Tasks vom Typ „Edit“ angelegt. Diese zwei Tasks werden beide mit dem Stern-Operator als iterative Tasks modelliert. Diese sind außerdem mit dem Interleaving-Operator bzw. als Concurrent Tasks verbunden. Der interaktive Task *Trigger Login* ist von Typ „Control“ und löst eine Überprüfung der Zugangsdaten aus. Da dieser durch dem Disabling-Operator mit dem vorherigen Task verbunden ist, unterbricht dieser Task die Eingabe-Tasks. Auf der rechten Seite ist *Trigger Login* über den Operator „Enabling with Information passing“ mit *Check Login* verbunden und gibt die Informationen von seinen „Vorgänger-Tasks“ an *Check Login* weiter (die Eingabedaten des Users). *Check Login* setzt die Eingangsinformationen als die Argumente jenes Webservices ein, mit dem es in MARIAE verbunden wurde. Da abhängig vom Resultat des *Check Login*-Tasks entweder zu *Login Deny* oder zu *Access* gewechselt werden soll, wird für diese Verzweigung eine Vorbedingung benötigt. Eine Vorbedingung in CTT ist zwar eine Eigenschaft des Tasks, wird jedoch nicht direkt im CTT-Baum dargestellt.

Die Definition dieser Vorbedingungen erfolgt ebenso wie die Definition des Task-Baumes in CTTE. Ein Beispiel für so eine Vorbedingung ist ein Boolean-Objekt. Einem Objekt muss in CTTE ein Name zugeordnet werden, wobei dieser Name identisch mit jener Bezeichnung sein muss, die bei der Implementierung des Back-End für die Bezeichnung des Ergebnisses einer Webservice-Methode benutzt wurde. Diese Objekte werden in der XML-Datei, welche den Task-Baum beinhaltet, mitgespeichert und ermöglichen es später MARIAE, die Reihenfolge der auszuführenden Tasks abhängig davon, ob bestimmte Bedingungen erfüllt sind oder nicht, in das Verhalten der GUIs zu übersetzen. Sollte das Login erfolgreich sein, wird zu *Access* gewechselt, wo im nächsten Schritt der Task *Get Data* ausgeführt wird, um Daten abzufragen. Die Ergebnisse dieser Abfrage

werden mit Hilfe des „Enabling with Information passing“ Operators dem Task *Show data* weitergegeben. Dieser Task ist von Typ „Display“ und zeigt dem Anwender die für ihn vorgesehenen Daten an. Im zweiten Fall, wenn das Login fehlschlägt, wird der Anwender auf eine Fehlerseite geleitet. Diese Seite wird in Abbildung 3.3 als ein Application Task mit der Bezeichnung *Login Deny* modelliert.

Bevor man aus dem Interaktions-Modell die entsprechenden Zwischenartefakte generieren kann, müssen noch die Application Tasks mit der passenden Applikation verbunden werden. In MARIA muss diese Anwendung als ein Webservice vorliegen. Diese Webservices werden im ersten Transformationsprozess durch den Anwender in einem dafür vorgesehenen Editor in MARIAE mit den Application Tasks verbunden. So eine Verbindung, auch *Binding* genannt, wird in Abbildung 3.4 dargestellt. Das Ergebnis eines Application Tasks kann nach der Ausführung als das Wert eines *Conditional Objects* verwendet werden. Auf MARIAE wird in Abschnitt 3.2 genauer eingegangen. Wie schon erwähnt, ist bei der Implementierung der Webservices darauf zu achten, dass man den Ergebnissen der Methoden mit Hilfe der „@WebResult“ Annotation die gleichen Namen wie den vorher definierten Conditional Objects zuweist. Hält man sich an dieser Regel, erkennt MARIAE bei der Verbindung der Webservices mit den Tasks die Zuordnungen zwischen den Methoden-Ausgaben und den Conditional Objects automatisch. Dadurch kann MARIAE das Ergebnis einer Webservice-Methode, nachdem diese Methode aufgerufen und ausgeführt wurde als Wert der entsprechenden Vorbedingung einsetzen.

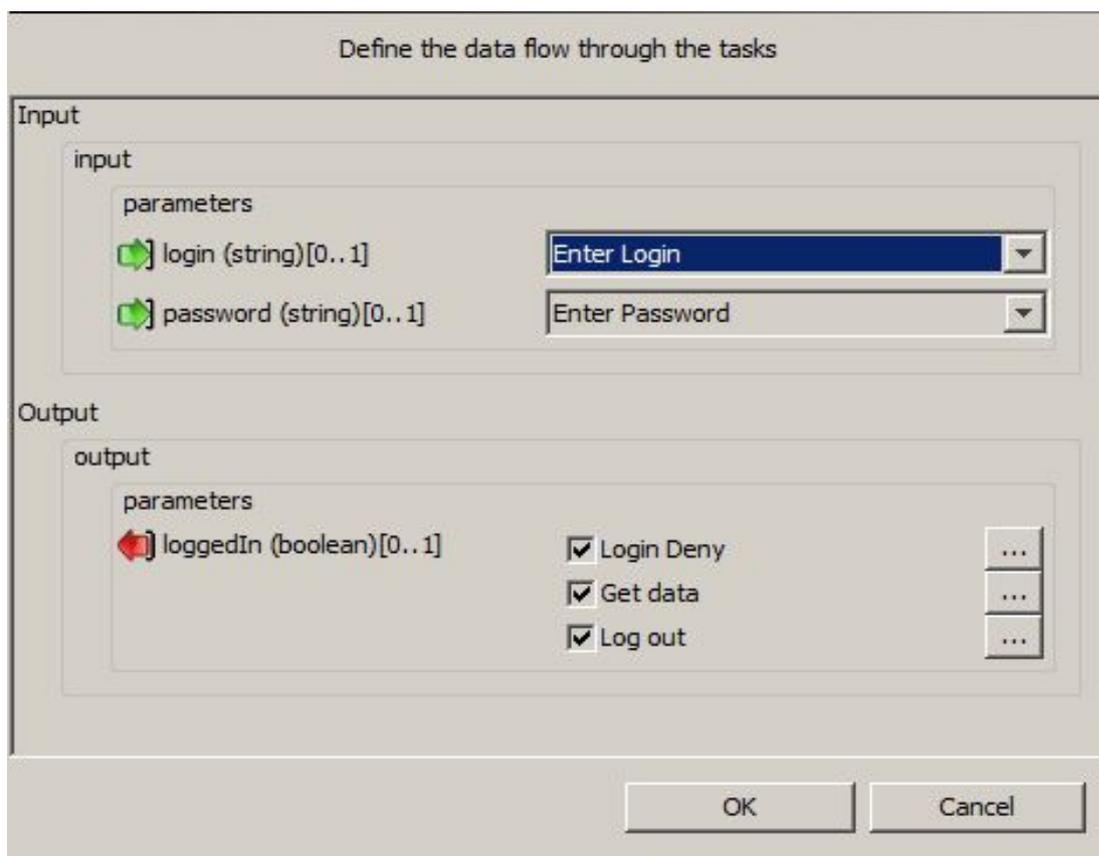


Abbildung 3.4: Beispiel für eine Binding in MARIAE

Das CTT-Modell bringt laut [Pat03] in Vergleich zu traditionellen Task-Modellen (wie z.B. GOMS [CNM83], UAN [HH93] und GTA [VdVLB96]) folgende Vorteile mit sich:

- Es bietet einen Fokus auf Aktivitäten, die für den Anwender im Vordergrund stehen.
- Es hat eine hierarchische Struktur, in der ein Task in mehrere kleinere Teile zerlegt werden kann.
- Es hat eine graphische Syntax, die einfacher zu verstehen ist als eine Text-basierte Darstellung.
- Es bietet eine breite Palette an temporalen Operatoren, mit deren Hilfe semantische Inhalte der Anwendung modelliert werden können.
- Verschiedene Objekt- und Task-Merkmale (Task-Kategorien, Task-Typen)

MARIA ist der einzige Ansatz, welcher die Modellierung von kooperativen Prozessen unterstützt. Darunter versteht man Anwendungen, bei denen mehrere Benutzer zum Erreichen eines Zieles kooperieren. Um so einen Fall modellieren zu können, definiert CTT für jeden Benutzer eine Rolle. Für alle Rollen werden Task-Bäume definiert, welche im Nachhinein miteinander verknüpft werden. Diese Verknüpfung gibt die Kooperation der unterschiedlichen Rollen wieder und erfolgt mit Hilfe der „Connection Tasks“. Diese Tasks sind in CTT mit einem blauen Strich und Pfeilen in beiden Richtungen markiert [Pat03].

3.1.2 MARIA-AUI

Für MARIA XML wurden basierend auf den Erfahrungen aus dem TERESA-Projekt eine Reihe an Erweiterungen definiert. MARIA XML setzt auf zwei Kernmodellen. Diese umfassen die Definition eines *Daten-Modells* und eines *Event-Modells*. Die Präsentationen, in denen das MARIA-AUI unterteilt wird, bauen auf diesen zwei Modellen auf.

Das Daten-Modell modelliert die dem GUI unterliegenden Daten. Dieses Daten-Modell wurde mit Hilfe der XSD SprachTyp-Definitionssprache definiert und ermöglicht es, abstrakte Interaktoren mit abstrakten Datenelementen aus dem Daten-Modell zu verbinden. Abbildung 3.5 gibt ein Beispiel für ein AUI-Modell wieder, wie es auch in MARIA definiert wird.

In Abbildung 3.5 ist die Instanz des Daten-Modells durch den Tag „data“ gekennzeichnet, welches den Wurzelknoten der XML-Struktur darstellt. Durch die *xs:complexType* Markierung werden komplexe Datenstrukturen gekennzeichnet, die aus mehreren *elementen* zusammengestellt sind. Diese Elemente, die zur Kennzeichnung simpler Datentypen verwendet werden können, werden durch *xs:element* markiert und stellen in der Sprache *xs* beliebige Objekte dar. Hiermit stellen *xs:complexType*-Knoten eine Aggregation unterschiedlicher, vordefinierter Datenstrukturen dar. Das bedeutet, dass eine Sequenz an Elementen einen *xs:complexType* darstellt. Ein Beispiel dafür, wie diese Definition einer Datenstruktur in MARIA verwendet wird, um die Eingangs-Parameter eines Webservices zu modellieren, ist das Element *xs:element name= „ns1_login“* in Abbildung 3.5. Hier werden die Eingangs-Parameter als *xs:elements* mit den Namen *login* und *password* modelliert. Diese Elemente sind von Typ String, welches in dem entsprechenden Namespace, zu den vordefinierten Datentypen gehört und daher auch als Attribut eines Elementes Verwendung finden kann.

```

<data>
  <xs:schema finalDefault="" blockDefault="" elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <xs:complexType name="ns1_Portal_PortalPortType_login" mixed="false">
      <xs:sequence minOccurs="1" maxOccurs="1">
        <xs:element name="input">
          <xs:complexType mixed="false">
            <xs:sequence>
              <xs:element name="parameters" type="ns1_login"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="output">
          <xs:complexType mixed="false">
            <xs:sequence>
            </xs:complexType>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="ns1_login">
      <xs:complexType mixed="false">
        <xs:sequence>
          <xs:element name="login" type="xs:string"/>
          <xs:element name="password" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ns1_loginResponse">
      <xs:complexType mixed="false">
        <xs:sequence>
          <xs:element name="loggedIn" type="xs:boolean"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</data>

```

Abbildung 3.5: Datenmodell der Login-Anwendung

Das Event-Modell modelliert, wie das GUI auf die Ereignisse reagiert, die der Anwender auslöst. Dafür definiert MARIA XML zwei Typen an Ereignissen: die „Property Change Events“ und „Activation Events“. Die erste Sorte an Ereignissen sind jene, die Werte für bestimmte Eigenschaften eines GUIs ändern. Der zweite Typ an Ereignissen wird durch sogenannte „Activators“ ausgelöst und führt zur Ausführung einer Anwendung (z.B. eines Webservices) [PSS09]. Abbildung 3.6 zeigt das Schema des Event-Modells für das Login-Beispiel. In diesem Beispiel wird für den *activator* „P1_Trigger_Login“ ein *script* definiert, welches ausgeführt wird sobald dieser activator ausgelöst wird. Hierbei spielen die Webservices eine zentrale Rolle. Die Informationen aus der Web Services Description Language (WSDL)-Datei, welche die Webservices beschreibt, werden in der ersten Transformation von der Concepts and Tasks-Ebene zu AUI in eine interne Darstellung, in eine sogenannte *function* transformiert. Der *script* gibt eine Sequenz von Befehlen an, die beim Triggern des activators ausgeführt werden. Unter *change_property* werden die Elemente des GUIs angegeben, die zu einer Änderung des Datenmodells führen sobald der activator ausgelöst wird. Das *data_value* „ui:P1_EnterLogin/value“ z.B. stellt die Eingabe über das Feld P1_Enter_Login dar. Diese Eingabe ändert in dem Datenmodell das Element „ns1_<Webservice>_<CheckLoginMethod>/input/parameters/login“, welches später als Eingabe-Parameter des Webservices „ns1_<Webservice>_<CheckLoginMethod>“ verwendet wird.

Der Knoten *invoke_function* referenziert die abstrakte Darstellung eines Webservices, welches den Namen „<Webservice>_<CheckLoginMethod>“ hat und wegen dem Binding in Concepts and Tasks Ebene erstellt wurde. Mit *parameter* und *results* werden erneut Teile des Datenmodells referenziert. Der Tag *parameter* referenziert die Eingangs-Parameter der function bzw. des Webservices. Das *result* stellt fest, unter welchem Namen die Ergebnisse des Webservices in dem Datenmodell abgelegt werden sollen.

```
<activator id="P1_Trigger_Login" enabled="true" focus="false" hidden="false" continuous_update="false">
  <events>
    <activation>
      <handler>
        <script>
          <change_property data_value="ui:P1_Enter_Login/value" data_reference="ns1_<Webservice>_<CheckLoginMethod>/input/parameters/login"/>
          <change_property data_value="ui:P1_Enter_Password/value" data_reference="ns1_<Webservice>_<CheckLoginMethod>/input/parameters/password"/>
          <invoke_function name="<Webservice>_<CheckLoginMethod>" namespace="<URL>">
            <parameter name="parameters" data_ref="ns1_<Webservice>_<CheckLoginMethod>/input/parameters"/>
            <result name="parameters" data_ref="ns1_<Webservice>_<CheckLoginMethod>/output/parameters"/>
          </invoke_function>
        </script>
      </handler>
    </activation>
  </events>
```

Abbildung 3.6: Event-Modell für das Login-Beispiel

Mit der ersten Transformation bzw. Konkretisierung wird das CTT-Modell in eine von der Plattform und der Modalität der Anwendung unabhängige Beschreibung der Schnittstelle, in das AUI, transformiert. Das AUI wird in mehreren sogenannten „Presentation Task Sets“ [PSS11] unterteilt. Diese beinhalten hauptsächlich zwei unterschiedliche Elemente, einerseits die Interaktionsobjekte und andererseits die Kategorisierung und Gruppierung von dieser Elemente. Unter Presentation Task Sets können die abstrakten Visualisierungen aller Tasks verstanden werden, die in der gleichen zeitlichen Periode aktiviert werden bzw. aktiv sind. Welcher Task zu welchem Zeitpunkt aktiv ist und damit in einem „Presentation Task Set“ auftaucht, wird anhand der Operatoren erkannt, die in einem CTT-Modell die Tasks miteinander verbinden. Abbildung 3.7 stellt einen Presentation Task für den Anwendungsfall „Login“ dar. Diese abstrakte Darstellung des ersten GUIs besteht in ihrem Kern aus zwei *text edit* Feldern. Diese stellen Textfelder dar, die der Anwender beliebig ausfüllen kann. Der *activator* P1_Trigger_Login löst das Login-Prozess aus und führt den Application-Task Check Login aus.

Während der Generierung der Presentation Task Sets wird auf die im vorherigen Schritt eingebundenen Webservices und Annotation-Files zugegriffen [PSS11]. Hierbei wird z.B. eine abstrakte Liste aller externen Funktionen generiert, die aufgerufen werden. Gleichzeitig wird ein entsprechender abstrakter Script definiert, der diese Funktionen aufruft. Dieser Script wird dann später in eine bestimmte Technologie übersetzt und als ausführbarer Code implementiert. Zusätzlich werden „Handler“ für die abstrakten Events generiert.

MARIA gewinnt aus den sogenannten Annotations zusätzliche Informationen für die Generierung von GUIs. In MARIA werden Annotations als Empfehlungen interpretiert, welche die Entwickler von Webservices den GUI-Entwicklern zur Verfügung stellen, um die Entwicklung von Service-basierten, interaktiven Anwendungen zu erleichtern [PSS11]. Die Wurzel der Annotations

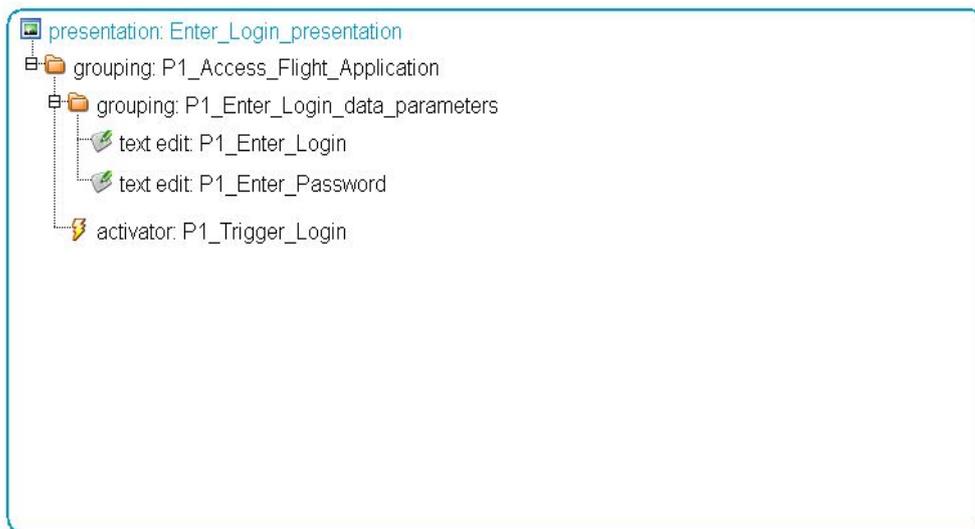


Abbildung 3.7: Ein Presentation Task aus der Login-Anwendung in MARIAE

liegt in dem Projekt ServFace,¹ wobei diese Annotations nicht mit den vordefinierten Webservice-Annotations verwechselt werden dürfen. Die Annotations fließen im Gegensatz zu den Webservices nicht direkt in die Anwendungslogik ein, sondern liefern zusätzliche Informationen zu den graphischen Eigenschaften der Interaktionsobjekten und Groupings in dem AUI. Hiermit werden die WSDL-Dateien der Webservices um zusätzliche Informationen ergänzt, erneut Deployed und können von MARIAE geladen werden. Diese Informationen können entweder mit dem Rendering einer Komponente zusammenhängen wie z.B. wie viele Spalten eine Komponente in Anspruch nehmen soll, oder ein bestimmtes Verhalten spezifizieren, z.B. welche Vorschläge dem Anwender angezeigt werden sollen, wenn Teile eines Wortes in einem Feld eingegeben worden sind. Die Annotations werden zusätzlich eingesetzt, um die richtigen Interaktoren im GUI einzubauen. Ein konkretes Beispiel für die Annotations ist die `xsi:type=SSAMA:VisualProperty` Annotation. Diese hat den Attribut `property` der verschiedene Werte annehmen kann. Zu diesen Werten gehören z.B. `disabled` oder `read-only`. Ein weiteres Beispiel ist die Validation-Annotation, die für die Validierung von Eingaben eingesetzt werden kann und in folgender Form in den Annotation-Files vorkommt (Zeile 2):

```

1 <SAM:AnnotationModel>
2 <annotations xsi:type="SAMA:Validation" languages="//@languages.1">
3   <expression expression="[0-9]+" expressionType="REGEX"/>
4   <message xsi:type="SAMA:TextFeedback" text="Invalid username" type="Error "
      languages="//@languages.1" platforms="//@platforms.1"/>
5 </annotations>
6   <platforms id="Desktop"/>
7   <platforms id="Mobile"/>
8   <languages id="French"/>
9   <languages id="English"/>
10 </SAM:AnnotationModel>

```

In diesem Beispiel wird überprüft, ob die Eingabe nur aus Ziffern zwischen 0 und 9 besteht (Zeile 3), wobei diese auch mehrfach vorkommen können. Mit *Platforms* (Zeilen 6 und 7) und *languages* (Zeilen 8 und 9) werden Identifikatoren für verschiedene Plattformen und Sprachen definiert.

¹<http://www.servface.eu/>

Referenziert werden diese Definitionen basierend auf der Reihenfolge ihrer Definition, wobei, bei Null beginnend gezählt wird. Das bedeutet, dass die „Validation“ , die in der Zeile eins definiert wird, nur für die zweite Sprachdefinition (Zeile 9) gilt. Der Feedback-Text in Zeile 4 gilt für die zweite languages-Definition und für die zweite platforms-Definition(Zeile 7).

3.1.3 MARIA-CUI

Das Concrete User Interface wird aus dem AUI generiert und stellt eine Ergänzung der Informationen dar, die schon im AUI vorhanden sind. Das Besondere an dem CUI ist, dass es die nötigen Informationen für eine Interaktion in einer bestimmten Modalität enthält. Das bedeutet, dass die abstrakten Definitionen auf der AUI-Ebene, durch konkrete Interaktionsobjekte ersetzt werden.

Als Beispiel für die Ergänzung des AUIs in MARIA soll das Element „Activator“ besprochen werden. Dieses Element stellt im Metamodell des AUIs einen Classifier dar, der bei einer Auslösung bzw. einem Triggern dazu führt, dass ein Webservice ausgeführt wird. Diese Services werden gemeinsam mit dem Activator aus den Binding Files und dem Informationsfluss im Task-Baum hergeleitet und im ersten Schritt im AUI eingebaut. Bei der Transformation des AUIs zum CUI wird das Element Activator, abhängig von der ausgesuchten Modalität, mit zusätzlichen Informationen ergänzt. Wenn es sich z.B. um ein GUI handelt, wird ein Activator als ein „Button“ markiert und um Attribute wie z.B. die Schriftgröße, die Position und der Farbe ergänzt. An dem vorgestellten Datenmodell und der abstrakten Darstellung der Webservices ändert sich bei der Transformation von AUI zu CUI nichts.

3.2 Werkzeuge

In diesem Abschnitt soll auf die Werkzeuge, die MARIA unterstützen, genauer eingegangen werden. Sowohl CTTE als auch MARIAE sind für Forschungszwecke frei verfügbar.

3.2.1 CTTE

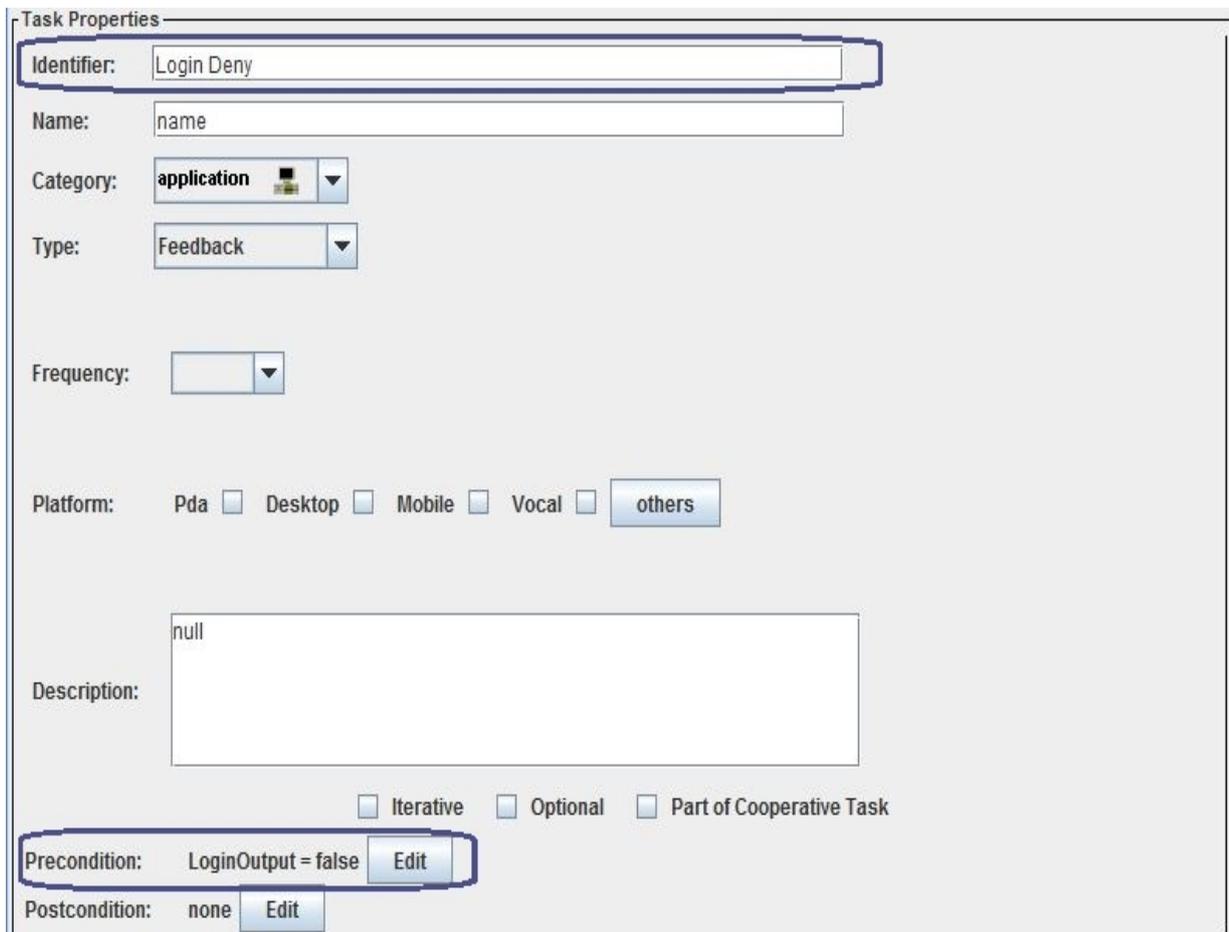
Wie oben schon kurz erwähnt, steht zur Modellierung von Tasks mit CTT das CTTE zur Verfügung. Dieses Tool bietet neben viele Möglichkeiten zur Task-Modellierung auch Funktionen zur Unterstützung der Evaluierung eines Modells. Außerdem werden zahlreiche Import- und Export-Funktionen unterstützt. Hier sollen drei der Funktionen vorgestellt werden, die in dieser Umgebung implementiert sind.

Die erste Funktion ist die des Simulators. Mit Hilfe dieser Funktion können die zuvor schon aufgebauten Task-Bäume auf ihr zeitliches Verhalten geprüft werden. Dabei ist es auch möglich, für bestimmte Tasks Conditional Objects zu definieren, welche einen bestimmten Zustand vorweisen müssen, damit ein Task ausgeführt wird. Abbildung 3.8 stellt ein Beispiel für Conditional Objects dar. Hier wurde für die Login-Anwendung ein Conditional Object namens „LoginOutput“ definiert. Dieses Objekt repräsentiert eine Vorbedingung, welche nach dem Login-Versuch die Information enthält, ob der Anwender sich richtig eingeloggt hat oder nicht. Dieses Objekt ist von Typ *Boolean*.

Die Conditional Objects werden also verwendet, um zur Ausführung von Tasks Vorbedingungen zu definieren. Auf der entsprechenden Seite zur Definition dieser Vorbedingungen gelangt man,



(a) Conditional Object



(b) Definition einer Vorbedingung in den Task Properties

Abbildung 3.8: Beispiel einer Vorbedingung in CTTE

wenn man in CTTE zwei Mal auf das Task-Symbol klickt, für welches diese Vorbedingung gelten soll. In der Abbildung 3.8(b) ist so eine Seite für den Application Task *Login Deny* der Login-Anwendung dargestellt. Im Feld *Precondition* ist die definierte Vorbedingung zu erkennen, in der das Conditional Object *LoginOutput* Verwendung findet.

Die Abbildung 3.9 stellt den Simulator in CTTE dar. Durch die entsprechende Check-Box kann der Wert eines Conditional-Objects simuliert werden. Da das dargestellte Object von Typ „Boolean“ ist, können dafür die zwei Zustände *true* und *false* simuliert werden.

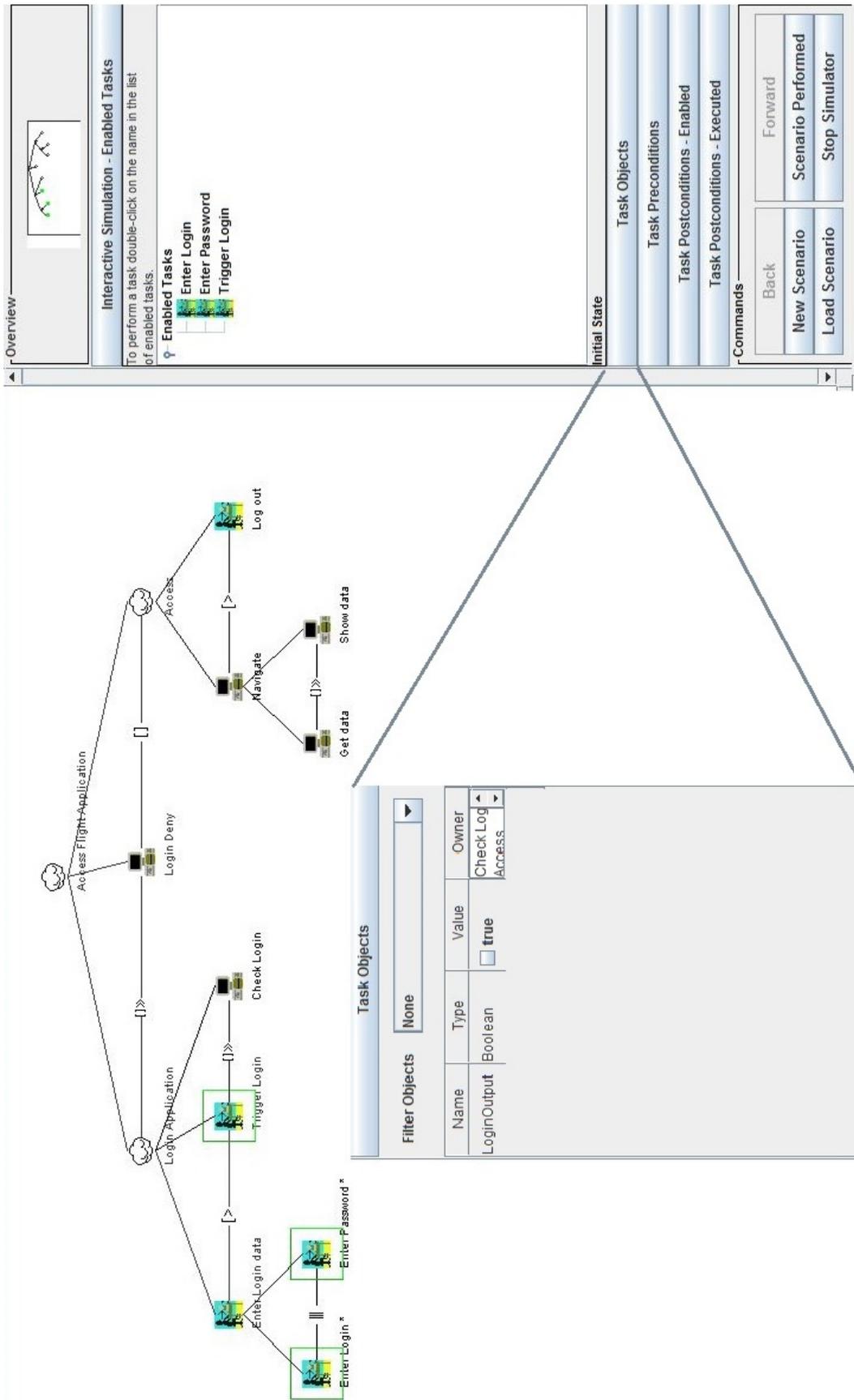


Abbildung 3.9: Der CTTE Simulator

Die zweite Funktion ist für die Überprüfung des Task-Baumes zuständig. Hiermit kann der Anwender den Baum auf syntaktische Korrektheit überprüfen. Abbildung 3.10 stellt das Ergebnis einer solchen Verifikation dar. CTTE lieferte in diesem Consistency Check eine Rückmeldung, in der bestätigt wird, dass keine Fehler im CTT-Modell gefunden worden sind.

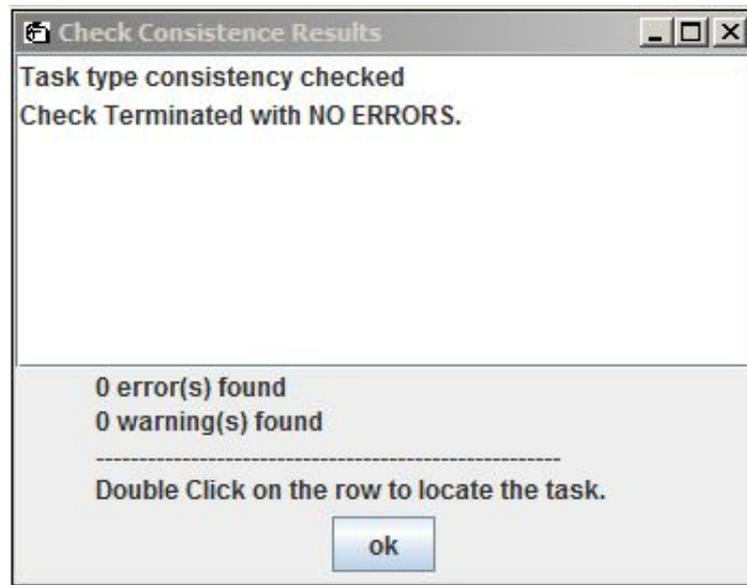


Abbildung 3.10: Consistency Check in CTTE

Die letzte hier vorgestellte Funktion bietet die Möglichkeit, aus WSDL-Dateien, die Informationen über Webservices enthalten, den entsprechenden Task-Baum zu generieren. Die WSDL-Dateien weisen für einfache Webservices einen einfachen, hierarchischen Aufbau.

3.2.2 MARIAE

MARIAE stellt dem Entwickler die benötigte Unterstützung für die Transformationen zwischen den Modellen und den Anpassungen an diesen Modellen zur Verfügung. Abbildung 3.11 zeigt einen Ausschnitt dieser Entwicklungsumgebung.

Auf der linken Seite findet der Entwickler eine Navigationsmöglichkeit, mit der er durch das Projekt navigieren kann. Nachdem der Entwickler das AUI generiert hat, findet er hier unter „Documents“ die zusammengestellten Presentation Tasks wieder. MARIAE bietet zusätzlich die Möglichkeit, über bestimmte Heuristiken die Anzahl an generierten Presentation Tasks zu minimieren bzw. redundante Informationen zu entfernen.

Im unteren Bereich zeigt das Tool, unter dem Begriff „Output“, die Log-Datei der Anwendung an.

Auf der rechten Seite des Fensters sieht man ein Panel, mit einem Webservice-Explorer und einem Annotations-Explorer. Mit der ersten Funktion kann der Anwender eine Liste von Webservices vom entfernten Server laden, um daraufhin die Tasks in einem Task-Baum mit diesen Webservices zu verknüpfen. Mit der zweiten Funktion können Annotation-Files geladen werden, die zusätzliche Informationen zur Generierung eines UI beinhalten.

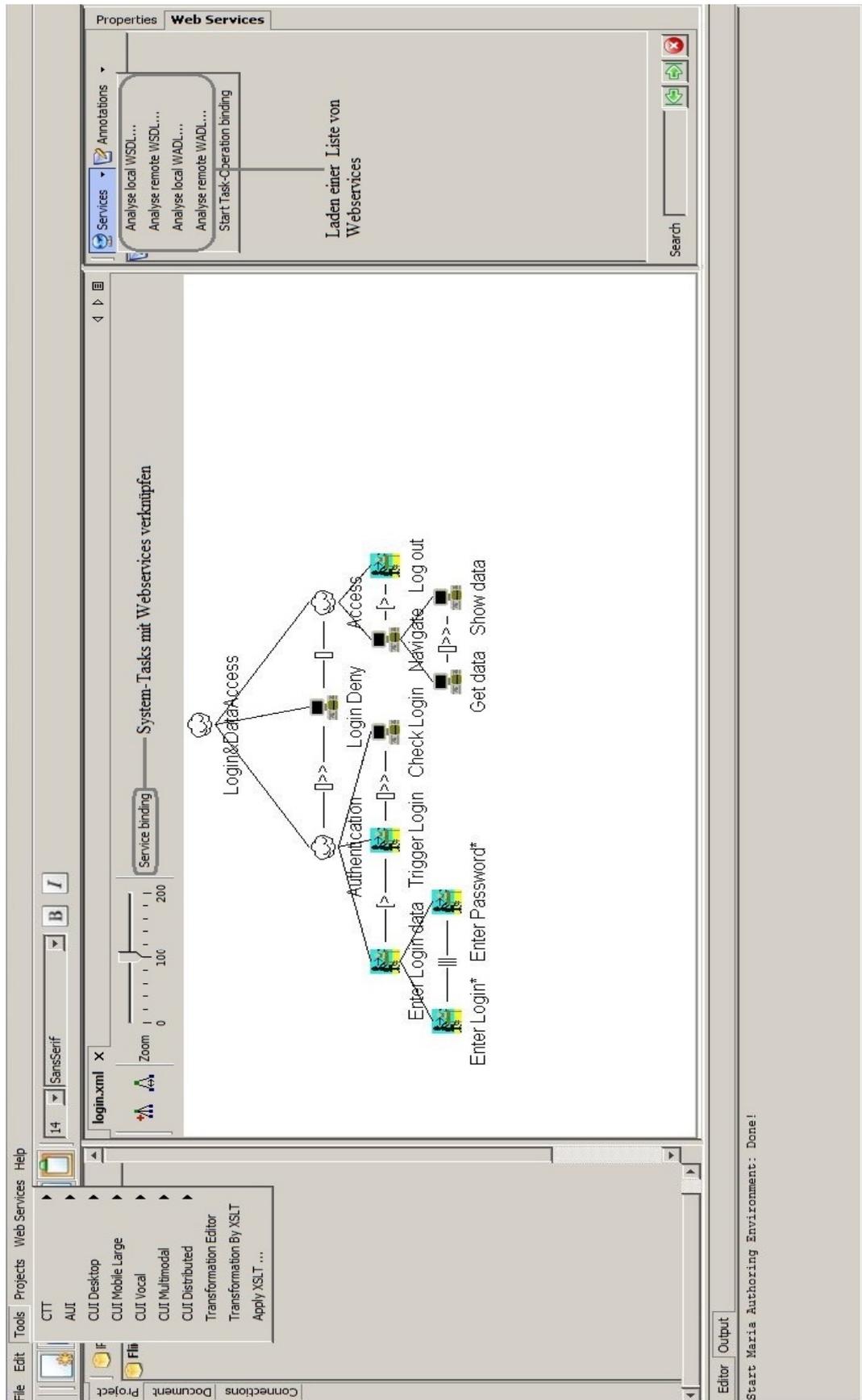


Abbildung 3.11: CTT browser und service binding in MARIAE

Der Anwender kann mit Hilfe dieser Umgebung die gesamte Entwicklungskette durchlaufen. Um den Begriff der Entwicklungskette genauer zu erläutern, soll das CRF herangezogen werden. MARIAE unterstützt einerseits auf allen Ebenen die *Reification* und stellt die notwendige Transformationen zur Verfügung. Andererseits wird durch MARIAE dem Entwickler die Möglichkeit gegeben, eine *Reflexion* durchzuführen. Vorausgesetzt, dass das Eingangsmodell in keiner proprietären Sprache definiert wurde, unterstützt MARIAE auch die Möglichkeit, die Entwicklung ausgehend von einer konkreteren Ebene als der Task-Modellierung zu starten.

Bezüglich Abstrahierung wurde in MARIA ein sogenanntes „Migration Framework“ implementiert [PSS09]. Dieses Framework unterstützt jedoch nicht den Design Time Process des CRF, sondern ermöglicht es, während der Laufzeit aus einem FUI für ein bestimmtes End-Gerät ein FUI für ein anderes Endgerät zu generieren. Da dieses Framework kein Teil von MARIAE ist, wird es hier nicht genauer beschrieben, jedoch kurz beim Vergleich der Ansätze in Abschnitt 6.1.3.1 vorgestellt.

Wie auch in Abbildung 3.11 wiedergegeben, findet der Entwickler unter „Tools“ alle Transformativmöglichkeiten, die für MARIA definiert sind. In dieser Abbildung ist noch einmal zu erkennen, dass MARIA verschiedenste Plattformen unterstützt.

Die Transformation von einem AUI in ein CUI erfolgt in MARIAE nach einer vordefinierten Vorlage. Das bedeutet, die Transformation eines Activators oder eines Textfeldes, die im Metamodell als „textedit“ bezeichnet werden, erfolgt immer auf die gleiche Weise. Dieses statische Verhalten ist jedoch aus der Hinsicht unerwünscht, dass die Kunden unterschiedliche Vorstellungen von Aufbau eines GUIs haben. Dieser Tatsache kommt MARIA damit entgegen, dass das CUI-Modell, genauso wie das AUI-Modell in einem Editor in MARIAE angepasst werden können. Dabei stellt MARIAE dem Entwickler eines GUIs eine Tool-Box mit alternativen GUI-Elementen zur Verfügung, mit der die Struktur des GUIs, angepasst und personalisiert werden kann. Der CUI Editor in MARIAE, der im Falle des Login-Beispiels die entsprechende Tool-Box für eine GUI inkludiert, ist in Abbildung 3.12 wiedergegeben. Auf der rechten Seite des dargestellten Fensters, befindet sich das Panel mit dem Titel „Desktop CUI“. Hier sind alle Presentation Task Sets, die auf der AUI-Ebene als abstrakte Beschreibungen des GUIs vorliegen nochmal dargestellt. Der Editor befindet sich im Zentrum des Fensters und zeigt das Presentation Task „Enter_Login_presentation“ als ein CUI. Das „Erdkugel“-Symbol in der Spalte über dem Editor ermöglicht es ein CUI als eine statische HTML-Seite darzustellen. Da es sich hierbei nur um einen Werkzeug zur Verifizierung der Struktur eines Presentation Tasks handelt, können in dieser Darstellung keine Webservices aufgerufen werden.

Da sich diese Arbeit mit GUIs auseinandersetzt, wurde für die Generierung des FUIs die Option „CUI Desktop“ verwendet. Diese bietet die Möglichkeit GUIs zu generieren, die Aufrufe von Webservices unterstützen. Das Layout, welches auch in der Abbildung 3.13 zu sehen ist, ist derzeit die einzige Struktur-Vorlage, welches von MARIAE für Desktop GUIs zur Verfügung gestellt wird.



Abbildung 3.12: CUI Editor in MARIAE

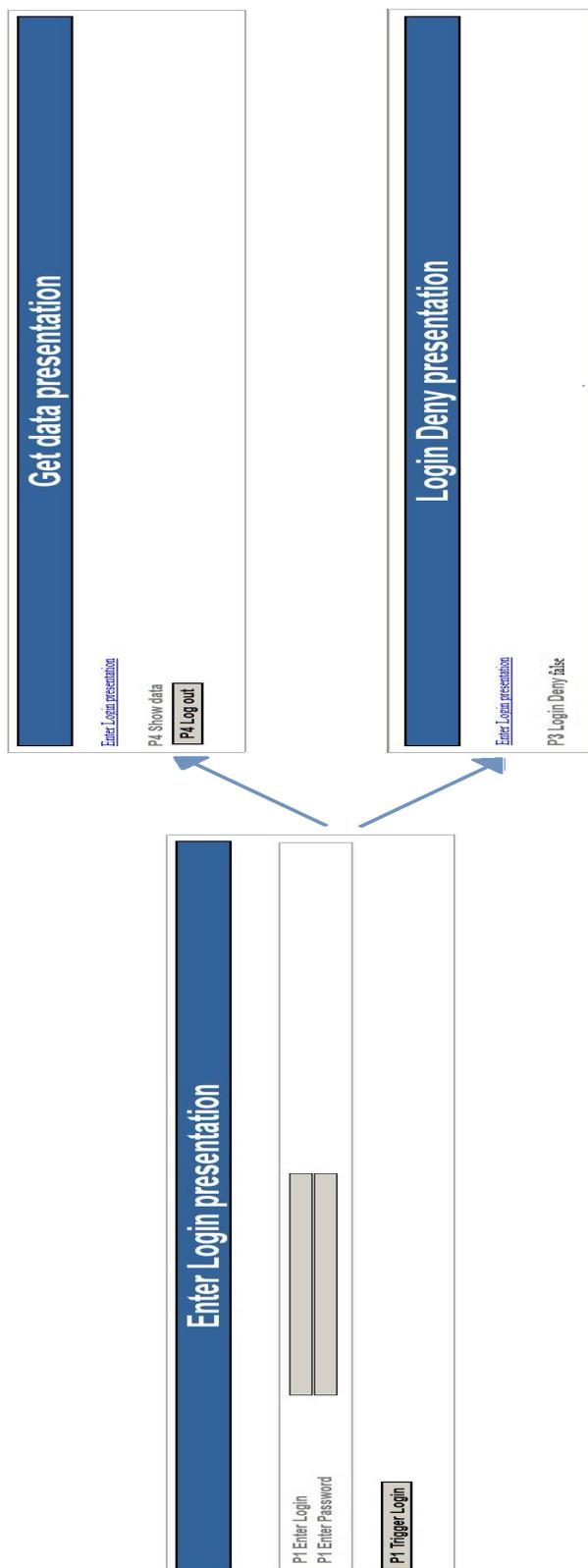


Abbildung 3.13: FUI der Login-Anwendung in MARIA

4 USER INTERFACE EXTENSIBLE MARKUP LANGUAGE (USIXML)

UsiXML 2.0 [Van08], hier als UsiXML bezeichnet, wurde an der Universität Catholique de Louvain, in Belgien¹, entwickelt. Dabei setzt UsiXML auf die OMG-Empfehlungen für MDA und unterstützt zusätzlich alle Abstraktionsebenen, die in CRF definiert werden.

4.1 Metamodelle

UsiXML definiert für die unterschiedlichen Abstraktionsebenen des CRF entsprechende Modelle. Diese Modelle sind Instanzen jener Metamodelle, in denen die Sprachelemente und die Relationen zwischen den Elementen definiert sind. Dieser Abschnitt stellt diese Metamodelle vor. Die Vorstellung beschränkt sich auf die Beschreibung jener Aspekte, die für die Implementierung des Login-Beispiels von Bedeutung sind.

4.1.1 Task-Modell

UsiXML orientiert sich bei seinem Task-Metamodell an dem „Goals, Operators, Methods and Selection“ (GOMS) Ansatz [CNM83]. GOMS ist ein spezielles Modell der menschlichen Informationsverarbeitung im Kontext der Human Computer Interaction (HCI). Basierend auf Erkenntnissen aus GOMS ist das Task-Modell in UsiXML vom Hierarchical Task Approach (HTA) inspiriert. Dies erlaubt dem Anwender, eine Baumstruktur für die Abarbeitung unterschiedlicher Tasks zu definieren. Dabei werden abstrakte Tasks festgelegt und im nächsten Schritt in Sub-Tasks zerteilt, die mit Hilfe von temporalen Operatoren miteinander verbunden werden.

Im Task-Metamodell von UsiXML definiert die Metaklasse *Decoration* Eigenschaften, die sowohl für Tasks gelten können als auch für die temporalen Relationen [VBMT12]. In UsiXML werden die aus dem CTT bekannten Task-Kategorien als sogenannte „Task-Decorations“ bezeichnet. Die Task-Decorations sind eine Spezialisierung der Decoration-Metaklasse und repräsentieren jene Decorations, die für Tasks gelten. Ein Task kann laut dem Metamodell mehrere Decorations referenzieren [VBMT12]. In dem Task-Metamodell von UsiXML, kommen neben den *System-*, *Interaction-* und *User* Decorations, die Decorations „Undefined“ und „Expression“ hinzu. Den mit dem CTT-Modell gemeinsamen Task-Decorations kommt die gleiche semantische Bedeutung zu.

¹<http://www.usixml.eu/>

Die Task-Decoration Undefined kann eingesetzt werden, wenn die Natur des Tasks nicht bekannt ist bzw. nicht genau den vordefinierten Decorations entspricht. Die Decoration *Expression* ist eine Spezialisierung der Metaklasse *Decoration*, die es dem Designer erlaubt, für temporale Relationen Decorations zu definieren. Unterschieden werden diese Task-Kategorien in dem dafür vorgesehenen Eclipse Editor durch eine Farbkodierung.

UsiXML setzt für den Kontrollfluss zwischen den Tasks, ähnlich wie beim CTT, temporale Relationen ein. UsiXML definiert folgende temporale Relationen: *ENABLING*, *CONCURRENCY*, *DISABLING*, *SUSPEND*, *ORDERINDEPENDENCE* und *CHOICE* [VBMT12]. Die Relation *ENABLING* erlaubt die Ausführung des nächsten Tasks, sobald der erstere abgeschlossen ist. Mit der *CONCURRENCY* Relation verbundene Tasks werden gleichzeitig aktiviert, wobei diese in einer beliebigen Reihenfolge oder gleichzeitig abgearbeitet werden können. Beim *DISABLING* unterbricht der zweite Task den ersten. Die Relation *SUSPEND* erlaubt es, zwei Tasks in einer Weise zu verbinden, dass der zweite Task zwar den ersten unterbrechen kann, der Zustand des ersten Tasks jedoch reproduziert werden kann. *ORDERINDEPENDENCE* implementiert das gleiche Verhalten wie *CONCURRENCY*, wobei die Reihenfolge, in der die Tasks abgearbeitet werden, nicht vorgegeben ist. Die Relation *CHOICE* wird zur Implementierung von Verzweigungen eingesetzt.

Abbildung 4.1 zeigt ein Task-Modell für die Login-Anwendung. Diese Repräsentation entspricht einer Baumstruktur, wobei der Task „LoginExample“ den Root Task bzw. den abstrakten Task darstellt. Dieser abstrakter Task, ist der erste Task, den der Designer des Task-Modells mit dem zu erreichenden Ziel assoziiert. Bei den hierarchisch unteren Tasks, die sogenannten „Leaf Tasks“, handelt es sich um Tasks, die nicht mehr weiter zerlegt werden können. Man erkennt in dem Modell drei Abstraktionsebenen. Für jede Abstraktionsebene kann der Designer eine temporale Verbindung der Unterknoten definieren. Diese temporalen Operatoren sind im unteren Bereich des Task-Modells zu erkennen, wobei sie mit den Farben grün oder blau umhüllt sind.

Im Task-Metamodell von UsiXML ist für die Task-Klasse ein Attribut mit dem Namen „canonical-TaskType“ vorgesehen. Mit Hilfe dieses Attributs kann für jeden Task eine abstrakte Definition des Vorganges gegeben werden, ohne genauer auf die Details einzugehen. UsiXML bietet hier eine Liste von vordefinierten Optionen an. Diese sollen die wichtigsten Aufgaben umfassen, die in Zusammenhang mit einem GUI durchgeführt werden können. Beispiele für mögliche Werte dieser Attribute sind: *SELECTION*, *CREATE*, *MODIFY* und *DELETE*. Diese Task-Typen definieren hiermit Aktionstypen, die im Aufgabenbereich des Anwenders liegen und mit denen der Anwender die Domänenobjekte manipulieren kann [VBMT12].

Wie schon oben erwähnt, definiert UsiXML die Task-Kategorien unter dem Namen „Task-Decorations“. Unter den Attributen der Decoration-Klasse finden sich die Attribute „maxIteration“, „minIteration“, „centrality“, „criticity“, „frequency“, „minExecutionTime“, „maxExecutionTime“ [VBMT12]. Die meisten dieser Attribute werden im Weiteren verwendet, um zusätzliche Informationen für die Generierung der folgenden Modelle zu gewinnen. Das wichtigste Attribut ist die „Nature“ eines Tasks, die bestimmt, welche Task-Kategorie zutreffend ist [LVM⁺04].

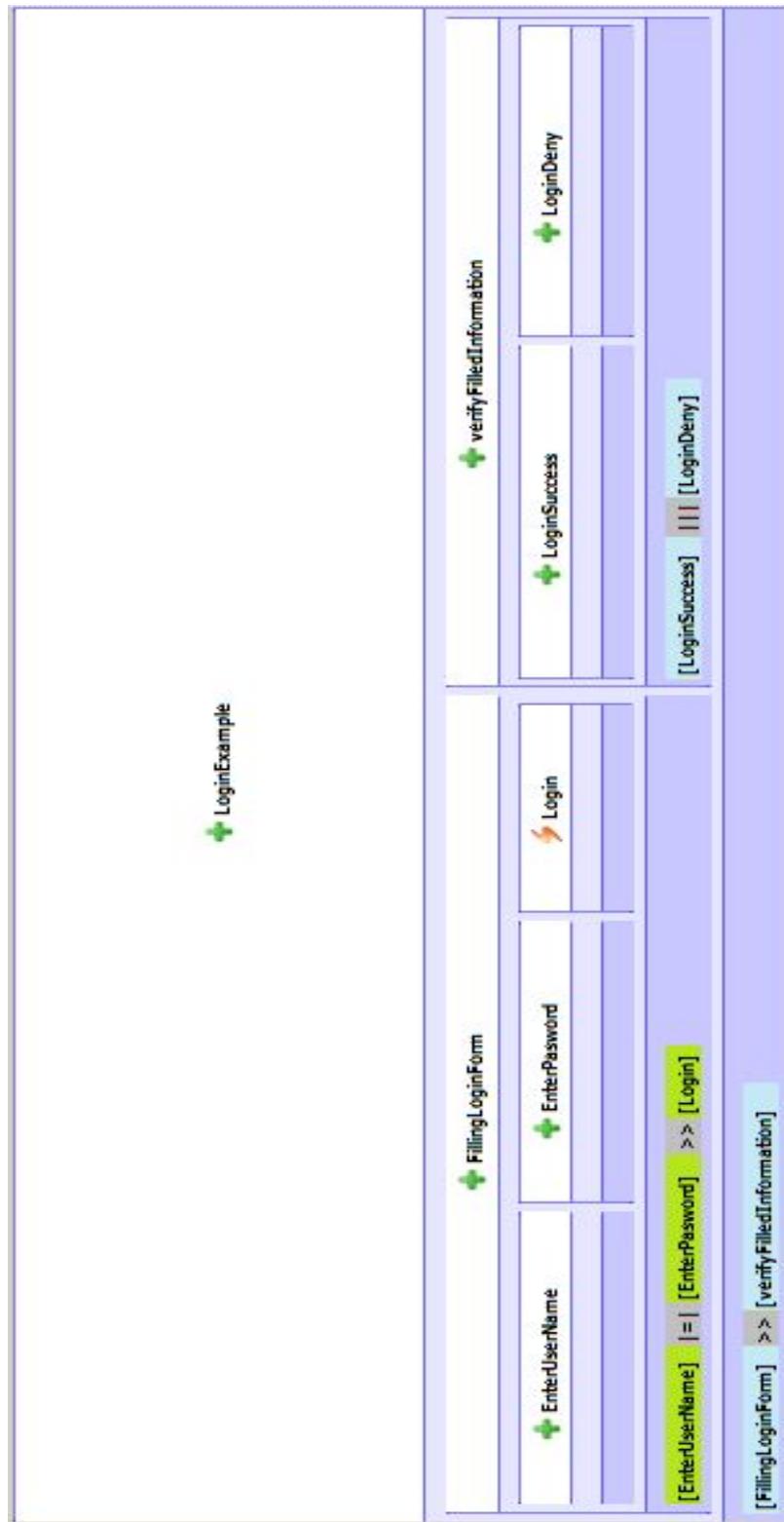


Abbildung 4.1: Task-Modell in UsiXMLs Task Editor

4.1.2 Domain-Modell

In einem Domain-Modell, wie es in UsiXML definiert wird, spezifiziert man in UsiXML all die Entitäten, die während einer Interaktion von dem Anwender manipuliert werden [Sta08]. Unter Manipulieren werden hier zwei Fälle verstanden: Entweder wird durch eine Interaktion ein Attribut eines Objekts manipuliert oder es wird eine Methode aufgerufen, die in einem Domänen-Objekt gekapselt ist und andere Aufgaben übernimmt, die zwar mit der Entität verbunden sind, jedoch keinen seiner Attribute manipulieren. Das Domain-Modell spezifiziert hiermit sowohl die Struktur als auch das Verhalten eines GUIs, da sie alle Entitäten, deren Attribute und die Beziehungen zwischen den Entitäten umfasst, die letztendlich in der GUI eingebaut werden. Damit ist die Aufgabe des Domain-Modells nicht, die gesamte Umgebung eines Systems zu definieren, sondern alle relevanten Interaktionsobjekte zu erfassen und somit die Struktur der Präsentationen zu spezifizieren.

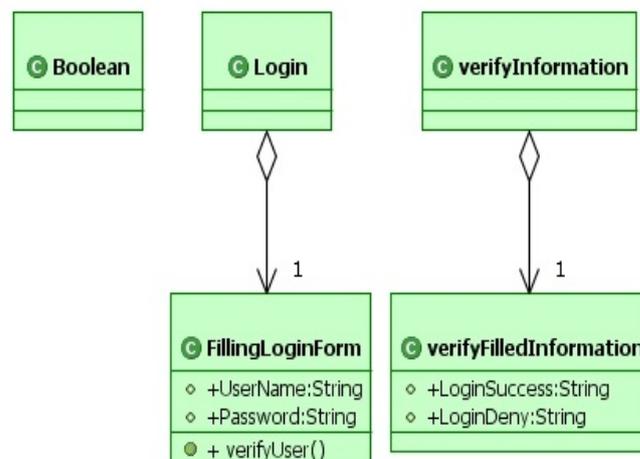


Abbildung 4.2: Ein Domain-Modell für die Login-Anwendung gemäß UsiXML

Die oben erwähnten Aspekte lassen sich mit Hilfe von UML-Klassendiagrammen sehr gut modellieren. Aus diesem Grund verwendet der entsprechende UsiXML-Editor in Eclipse zur Modellierung von Domänen-Objekten das UML-Metamodell für ein Klassendiagramm [LVM⁺04].

Das Domain-Metamodell besteht aus den vier Kernkonzepten: *Class*, *Attribute*, *Method* und *Domain Relationship*. Abbildung 4.2 zeigt das Domain-Modell der Login-Anwendung und beinhaltet für jedes Kernkonzept ein Beispiel. Diese Elemente sind entsprechend Ihrer Definitionen im UML-Standard [Lar02] zu verstehen. Die grünen Kreise, die einen „c“ umfassen, kennzeichnen eine *Class*. Diese Classes werden im nächsten Schritt in Fenster eines UsiXML AUI transformiert, die Interaktionsobjekte eines GUIs beinhalten. Für die Login-Anwendung sind das die beiden Fenster *Login* und *verifyInformation*. Das Login-Fenster umfasst das innere Fenster *FillingLoginForm* und *verifyInformation* umfasst das Fenster *verifyFilledInformation*. Die Attribute und die Methoden sind in den grünen Vierecken dargestellt. Dabei kennzeichnet ein Kreis eine *Method* und ein Diamant ein *Attribut*. Um die Information zu berücksichtigen, welcher Task welche Domänen-Objekte beeinflusst, muss mithilfe eines „Mappings“ die Verbindung zwischen den Tasks und den entsprechenden Methoden in den Domänen-Klassen erstellt werden.

4.1.3 Context-Modell

Ein Context-Modell kann man in die folgenden drei Sub-Modellen unterteilen [LVM⁺04]:

- User-Modell: In diesem Modell wird der User durch Attribute modelliert.
- Platform-Modell: modelliert die plattformspezifischen Attribute, die zum Zustand des Contexts beitragen können. Ein Beispiel für so ein Attribut ist die Bildschirmgröße.
- Environment-Modell: beschreibt Eigenschaften der physikalischen Umgebung, in der das GUI verwendet wird.

Die Aufgabe des Context-Modelles ist es, die Anwender und Plattformen, die aktiv bei einer Interaktion mitmachen zu modellieren und gleichzeitig den Einfluss der Umgebung auf der Interaktion mitberücksichtigen.

In Abbildung 4.3 ist das Context-Modell der Login-Anwendung dargestellt. In der rechten Spalte ist das Tool-Box in Eclipse-Plugin gezeigt. In diesem Modell wurden vier Observables definiert. Die *Observables* spezifizieren konkrete Akteure. *User* modelliert einen Anwender. Dieser hat ein Attribut mit dem Namen *User*. *Device*, *Screen* und *ConnectedToInternet* modellieren die in diesem Kontext involvierten Systeme. Ein *Device* z.B. hat die Attribute *Model*, *type* und *size*.

LoginTester, *Desktop* und *ConnectedToInternet* stellen Instanzen der Klasse *State* dar und dienen dazu, einen möglichen Zustand zu modellieren, den die Plattform annehmen kann. Ein Beispiel für so einen Zustand, ist der *Desktop*, in dem festgelegt wird, welche Eigenschaften ein *Device* und der *Screen* annehmen.

Die Klassen *SimpleRoom* und *Room* sind Instanzen des dritten Bausteines eines Context-Modells, das Environment-Modell. Damit beschreiben sie eine physikalische Umgebung in der ein GUI Anwendung finden kann.

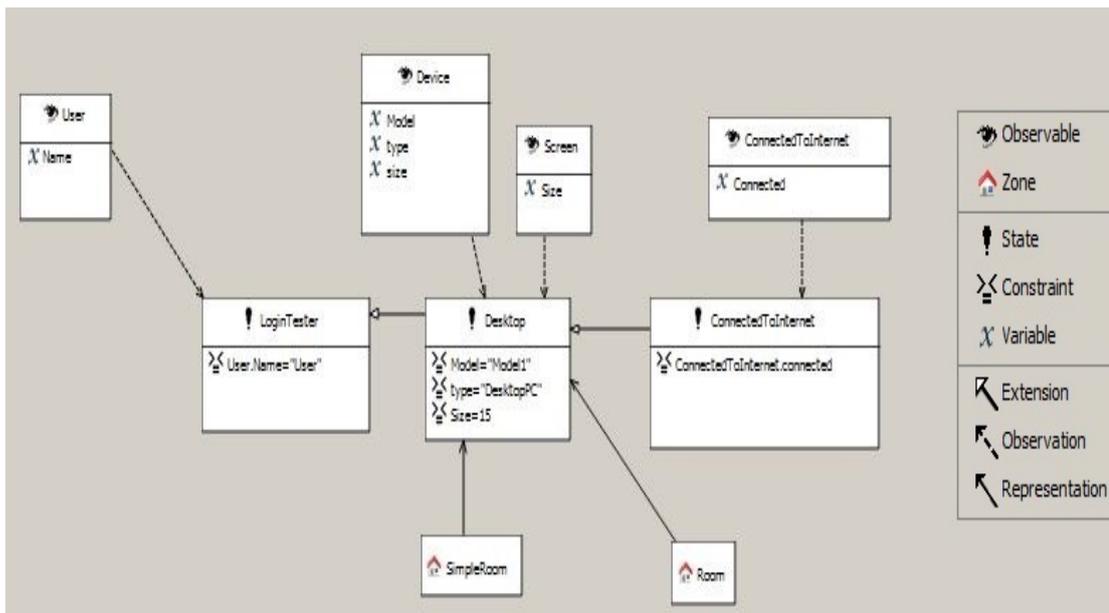


Abbildung 4.3: Context-Modell der Login-Anwendung

4.1.4 AUI-Modell

Das AUI entspricht in UsiXML der PIM-Ebene in MDE [Van05] und besteht in seinem Kern aus den Elementen „abstract interaction unit“ und „abstract relationship“ [LVM⁺04].

Unter den Abstract Interaction Units versteht man Elemente, die eine Abstraktion der herkömmlichen Widgets darstellen. Als Beispiel sind hier Fenster und Buttons für GUIs zu nennen. Da UsiXML mit seinem AUI-Metamodell die Modellierung von Multimodalen User Interfaces ermöglicht, stellen die Interaktoren im AUI gleichzeitig Abstraktionen für Widgets aller Modalitäten dar. Abbildung 4.4 stellt einen Ausschnitt jenes Metamodells dar, welches laut [VBMT12] für ein AUI in UsiXML definiert wurde. In diesem Klassendiagramm stellen die gelben Klassen die AUI-Metaklassen dar, während die blauen Klassen die vordefinierten Optionen der Attribute darstellen. Wir verwenden dieses Metamodell, um bei der Erklärung des AUIs für die Login-Anwendung die Möglichkeiten des UsiXML AUI besser zu erläutern.

Abbildung 4.5 stellt ein AUI-Modell in UsiXML dar. Dieses AUI repräsentiert eine abstrakte Realisierung des GUIs für die Login-Anwendung. Die äußeren Rahmen sind Instanzen der Klasse *AbstractCompoundIU*, die eine Kombination von mehreren *AbstractInteractionUnit* erlauben. Die eingebauten *InteractorUnits*, über denen eine Eingabe von Daten bzw. Ausgabe von Ergebnissen ermöglicht wird, sind Instanzen der Klasse *AbstractDataIU*. Das abstrakte Interaktionsobjekt mit dem Namen Login, stellt eine Instanz der Klasse *AbstractTriggerIU* dar.

Die oben erwähnten Objekte des AUI werden aus dem Domain-Modell hergeleitet. Dabei wird jede Klasse in dem Domain-Modell auf einen *AbstractCompoundIU* abgebildet. Die Properties der einzelnen Klassen im Domain-Modell werden im AUI durch *AbstractDataIUs* ersetzt.

UsiXML erlaubt es auf der AUI-Ebene Listener zu definieren. *verifyInformation*, *LoginSuccess* und *LoginDeny* in Abbildung 4.5 sind Beispiele für sogenannte *Rules*, die zusammen einen *AbstractListener* bilden. Das bedeutet, sie lauschen auf bestimmte Ereignisse und evaluieren bei deren Auftritt eine bestimmte Bedingung. Dementsprechend definiert die Klasse *AbstractListener* des Metamodells eine *Justification*, einen *EventExpression*, eine *Condition* und einen *ActionExpression*. Sobald das *AbstractEvent* aufgetreten ist, wird überprüft, ob die spezifizierte *Condition* erfüllt ist. Dieses Event kann z.B. eine Eingabe von Daten oder eine Selektion aus einer Liste sein. Wenn die *Condition* erfüllt ist, wird eine bestimmte *Action* ausgeführt.

Zu den Actions, definiert das Metamodell verschiedene Typen an *AtomicActions*. Zu diesen Typen gehören unter anderem die *InteractionUnitOpen(IUOpen)*, *IUClose*, *modelSearch* und *modelCreate*. Abbildung 4.4 enthält eine vollständige Liste der Optionen für die erwähnten Klassen, aus deren Komposition eine Rule entsteht. Diese sind in 4.4 als Instanzen der Metaklassen, als blaue *enumeration*-Klassen dargestellt.

4.1.5 CUI-Modell

Dem CRF entsprechend, liefert UsiXML in seinem Transformations-Metamodell die nötigen Transformationen, die eine Definition des CUIs aus dem AUI ermöglichen. Bei der Generierung des CUIs werden die abstrakten Objekte aus dem AUI auf konkrete, von der Modalität der Interaktion abhängige Elemente des CUI-Metamodells abgebildet. Im Prozess der CUI-Generierung aus dem AUI wird als Eingangsmodell neben dem AUI auch das Domain-Modell benötigt. Dies ist dadurch bedingt, dass das Transformationsprozess bestimmte Informationen zu der Transformation der abstrakten Actions in konkrete Actions nur aus dem Domain-Modell gewinnen kann.

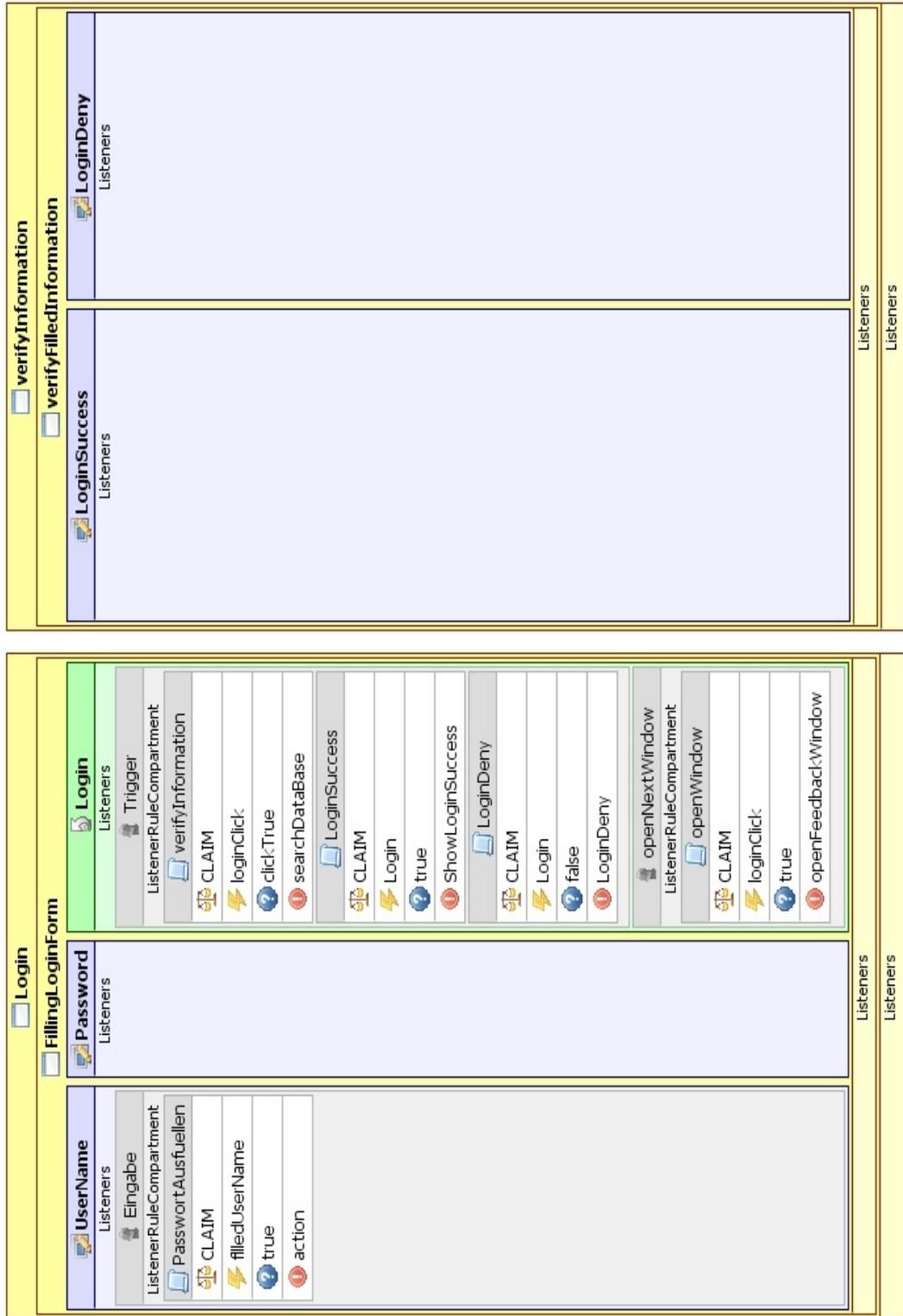


Abbildung 4.5: AUI-Modell der Login-Anwendung^a

^a<http://www.usixml.eu/>

FUIs zu evaluieren, wurde im Zuge dieser Arbeit eine Realisierung dieser Ebene mit Java Swing implementiert. Hierbei sollte überprüft werden, ob mit Hilfe der Informationen, die in dem CUI-Modell vorliegen, die Möglichkeit besteht, alle Widgets zu erkennen und in einer Endtechnologie zu implementieren.

Als Technologie wurde Java Swing eingesetzt. Die hauptsächlichen Gründe für diese Entscheidung waren die Bekanntheit dieser Technologie und das Vorliegen zahlreicher Dokumentationen. Die Ergebnisse dieser Implementierung sind in Abbildung 4.7 zu sehen.

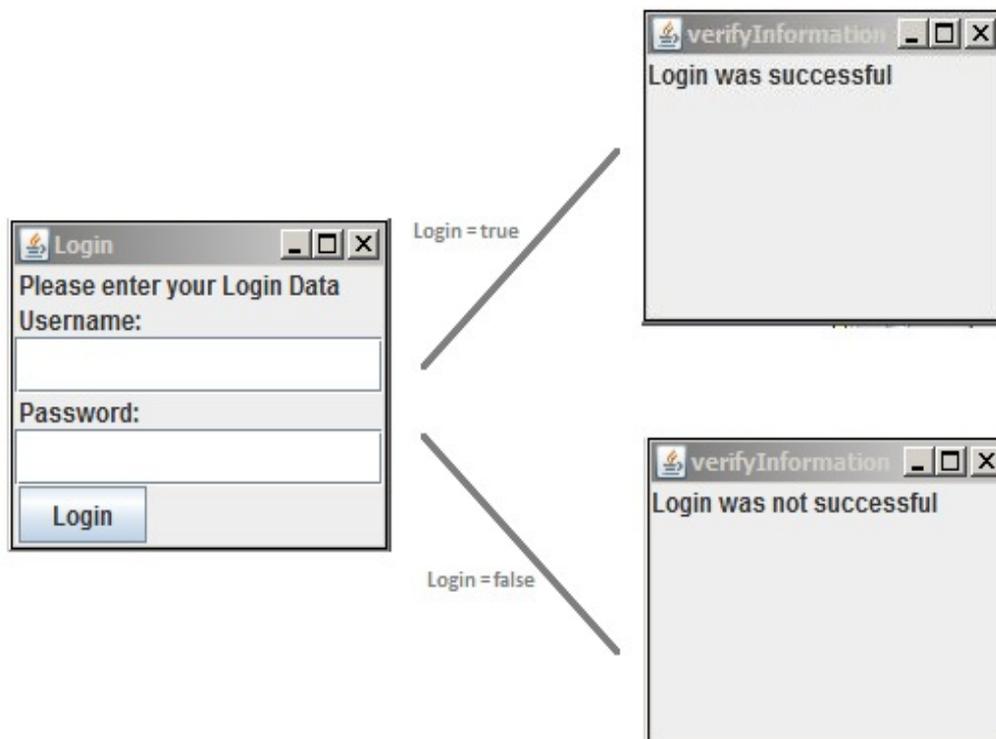


Abbildung 4.7: FUI für die Login-Anwendung

4.2 Werkzeuge

Laut [Van08] wurden für verschiedene Transformationen in UsiXML unterschiedliche Werkzeuge konzipiert und aufgebaut. Diese sind jedoch für UsiXML 2.0 nicht verfügbar. Für UsiXML 2.0 stehen nur Editoren zur Bearbeitung der Modelle zur Verfügung auf die wir weiter unten näher eingehen. Das Forschungsteam rund um UsiXML 1.0 hat laut [Van08] mit den entwickelten Werkzeugen Lösungen für die folgenden Fragestellungen gefunden:

- Einlernzeiten: Um diese zu verkürzen, wurden graphische Editoren gebaut.
- Unvorhersagbarkeit der Ergebnisse: Diesem Aspekt kommt UsiXML mit einer ausführlichen Dokumentation und mit Werkzeugen entgegen, in welchen der Entwickler die Zwischenergebnisse visualisieren kann.

- Interoperabilität der Modelle: Da UsiXML bei der Beschreibung seiner UIs auf XML-basierte Sprachen setzt und XML zu den W3C-Standards gehört, ist eine Portabilität und damit eine bessere Möglichkeit der Interoperabilität gegeben.

Beispiele für Werkzeuge aus UsiXML 1.0 sind FlowiXML [GVGC08] für die Concepts und Tasks-Ebene, IdealXML [MLJ06] für die AUI-Ebene und SketchiXML, GrafiXML [MV08] und VisiXML für die CUI-Ebene.

In UsiXML ist die Erstellung eines ersten Prototyps relativ aufwendig. So wie schon erwähnt, ist in den Plugins für UsiXML keine Transformationslogik inkludiert. Abgesehen von den Eclipse Plugins gibt es laut [Van08] noch mehrere Werkzeuge, die die Entwickler während der Entwicklungsphase mit UsiXML unterstützen sollen². Die Transformationen sind als extra Werkzeuge implementiert, die jedoch nicht verfügbar sind. Damit steht dem Entwickler keine vordefinierte Transformation zur Verfügung.

In UsiXML 2.0 wurden für die Entwicklungsphase basierend auf EMF ein Editor und entsprechende Plugins definiert.³ Diese decken jedoch die Entwicklung der Modelle nur auf der Concepts and Tasks- und der AUI-Ebene. Die Abbildungen 4.1 und 4.5 stellen für die Login-Anwendung die Modelle dar, die mithilfe dieser Werkzeuge aufgestellt worden sind. Zu den Eclipse-Editoren sind online verschiedene Showcases zu finden.⁴

²<http://www.youtube.com/channel/UCERmu0eqGI2dkeKU3CKRtZw>

³http://www.usixml.eu/usixml_tools

⁴<http://www.youtube.com/user/tesorieror?feature=watch>

5 UNIFIED COMMUNICATION PLATFORM (UCP)

Die UCP,¹ entwickelt am Institut für Computertechnik der Technischen Universität Wien, setzt für die Spezifikation einer Interaktion auf einen Diskurs-basierten Ansatz. UCP besteht nicht nur aus einem Generierungs-Framework bzw. einer Rendering Engine (UCP:UI) für die GUIs, sondern umfasst zusätzlich ein Kommunikationsmodell (UCP:CM) und eine Laufzeitumgebung (UCP:RT). Das Kommunikationsmodell ermöglicht es, die Kommunikation zwischen zwei Partner zu spezifizieren. Die Laufzeitumgebung integriert im letzten Schritt das generierte GUI und die Applikationslogik.

In Abbildung 5.1 sind die verschiedenen Komponenten der UCP dargestellt.

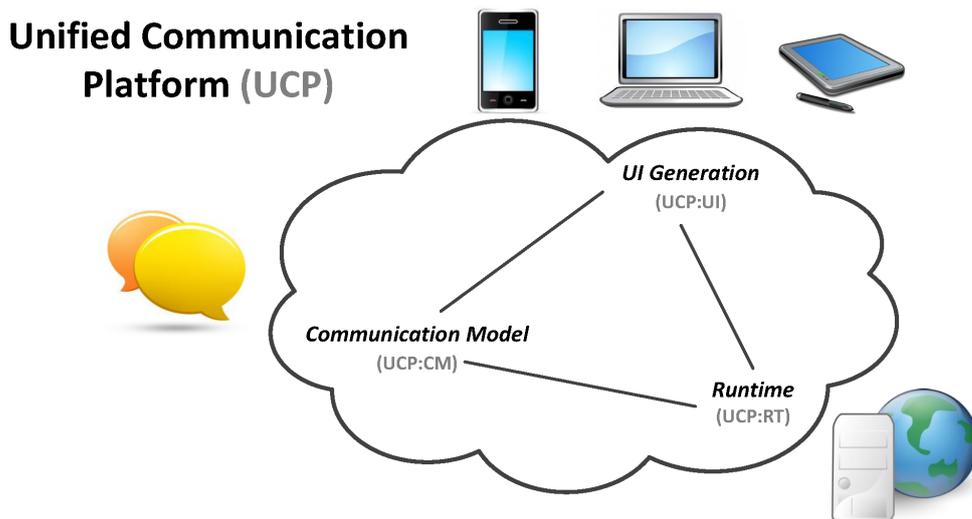


Abbildung 5.1: Die Komponenten der Unified Communication Plattform (UCP) (kopiert von der Homepage der UCP²)

¹<http://ucp.ict.tuwien.ac.at/wordpress/>

5.1 Metamodelle

Das Discourse-based Communication Model (UCP:CM) erlaubt die Modellierung von kommunikativer Interaktion zwischen zwei Kommunikationspartnern [EK12]. Im UCP muss zur Generierung eines GUI ein *Discourse Model*, ein *Action-Notification Model (ANM)* und ein *Domain of Discourse Model (DoD)* erstellt werden. Das einzige der drei Modellen welches zur Entwicklung eines ersten Prototyps nicht explizit definiert werden muss, ist das ANM. Dieses liegt als eine Basis Version vor, in der oft verwendete *Actions* und *Notifications* (wie z.B. set und get) bereits definiert sind. Zusätzlich müssen noch die *Applikationslogik* und der *Application Adapter* implementiert werden. Diese werden ausführlich in Abschnitt 5.1.4.3 besprochen. Das Generierungs-Framework, welches unter anderem auch die Transformationen und die *Devicespecification* umfasst, bietet für die Generierung von ersten Prototypen vordefinierte Konfigurationen. Die einzelnen Modelle im Communication Model bilden den Inhalt der folgenden Abschnitte.

5.1.1 Domain-of-Discourse Model

Das DoD Model spezifiziert jene Objekte, über die zwei Partner während einer Interaktion „sprechen“ können. Dabei müssen diese nicht mit dem Domain Model der Applikationslogik übereinstimmen. Im DoD Model werden zusätzlich auch die Relationen zwischen den enthaltenen Dialog-Objekten spezifiziert.

Zur Modellierung des DoD wird ein *Ecore Model* verwendet. Der Eclipse Editor ermöglicht es dem Anwender, inspiriert durch Konzepte des objektorientierten Paradigmas, Klassen anzulegen. Da für bestimmte Fälle die vordefinierten Datentypen (wie z.B. String) nicht ausreichend sind, um die Eigenschaften dieser Elemente zu spezifizieren, erlaubt der UCP- Editor die Definition von eigenen Datenstrukturen.

Das DoD der Login-Anwendung ist in Abbildung 5.2 dargestellt. Da in dem Login Use Case die Informationen über den User die einzigen sind, die zwischen dem Anwender und der Maschine ausgetauscht werden, beschränken sich die DoD-Objekte in dieser Anwendung auf die Klasse des Users und der User braucht, damit er sich in dem System einloggen kann, einen *userName* und ein *Password*. Zusätzlich kann jeder User ein persönliches Profil mit persönlichen Informationen haben. Diese Informationen werden in der Klasse *personalInformation* zusammengefasst.



Abbildung 5.2: DoD Model der Login-Anwendung

Bevor jedoch dieses Modelle zum Einsatz kommen kann, muss der Entwickler mit Hilfe eines *genmodel*-s den entsprechenden Java Code zu den DoD-Klassen generieren. Das *genmodel*³ ist ein

³„The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model. It includes a GUI from which generation options can be specified, and generators can be invoked. (<http://www.eclipse.org/modeling/emf/?project=emf>)“

Transformationsmodell. Dieses Transformationsmodell ist eine Hauptkomponente von EMF und kann automatisch aus einem Ecore-Modell die entsprechenden Quellcode-Dateien generieren.

5.1.2 Action-Notification Model

Zusammen mit dem DoD erlaubt das Action-Notification Model (ANM) in UCP die Definition von *Propositional Content* der *Communicative Acts*. Das ANM wird verwendet, um auf Daten aus dem DoD zuzugreifen oder diese zu manipulieren. Außerdem repräsentiert es noch applikationsspezifische Aktionen. Mit dem Propositional Content wird in dem Discourse Model spezifiziert, welche Aktionen beim Halten eines Dialoges ausgeführt werden müssen. UCP liefert ein *basic.anm*-Modell für die Actions und Notifications. Dieses Modell kann, abhängig von der Anwendung, erweitert und damit vervollständigt werden.

Für die Login-Anwendung wird, abgesehen von dem Basismodell für Actions and Notifications, ein eigenes Modell benötigt. Dieses enthält spezifische Actions und Notifications, die im Detail in der Applikationsanwendung implementiert werden. Abbildung 5.3 stellt dieses spezifische Modell dar. Wie man erkennen kann, werden zwei Klassen an Elementen definiert, nämlich die Actions die es ermöglichen, einen Befehl seitens des Anwenders zu Triggern und die Notifications, die eine Rückmeldung darstellen. Unter Anwender soll hier der Partner verstanden werden, der einen Communicative Act anstoßt.



Abbildung 5.3: Action-Notification Model in UCP

Für die Login-Anwendung benötigen wir vier, explizit definierte Actions und Notifications. Hierbei stellen die Klassen mit einem grünen Viereck im oberen linken Eck die Notifications und die, mit einem violetten Viereck die Actions dar.

5.1.3 Discourse Model

Das Discourse Model [FKH⁺06] spezifiziert alle möglichen Abläufe einer Kommunikation zwischen zwei Partnern. Dieses Modell basiert auf einige Theorien. Zu diesen gehören die *Speech Act-Theory* nach [Sea69], die *Conversation-Analysis* nach [LFG90] und der *Rhetorical Structure Theory* (RST) von [MT88]. Unter Discourse wird im Zusammenhang mit UCP das Halten eines

Dialoges verstanden. Um dieses Dialog zu spezifizieren, definiert UCP drei Klassen an Elementen: *Communicative Acts*, *Discourse Relations* und *Adjacency Pairs*.

Die Communicative Acts bilden den grundlegenden Baustein des Discourse Models. Sie können als eine Abstraktion von Speech Acts [Sea69] verstanden werden und modellieren den Austausch von Information zwischen zwei Kommunikationspartnern. Jedem Communicative Act wird ein *Agent* zugeordnet. Dieser Agent wird in Form einer Farbkodierung dargestellt. Ein Adjacency Pair wird durch einen *Opening* Communicative Act gestartet und durch einen *Closing* Communicative Act abgeschlossen, der auch mehrere Closing Acts haben kann. Ein Beispiel für so einen Fall wäre, wenn man dem Anwender einen Vorschlag in Form eines *Offers* macht und der Anwender diesen Offer entweder akzeptieren, oder ablehnen kann.

Adjacency Pairs werden durch die Discourse Relations miteinander verknüpft wobei zwei Kategorien unterschieden werden: *RST Relationen* und *Procedural Constructs*. Die RST-Relationen sind, wie ihr Name andeutet, durch dem schon erwähnte RST inspiriert. Die Procedural Constructs bieten zusätzliche Relationen zur expliziten Definition des Kontrollflusses.

Abbildung 5.4 stellt das Discourse Model der Login-Anwendung dar, welches in dem dafür konzierten Editor zusammengestellt wurde.

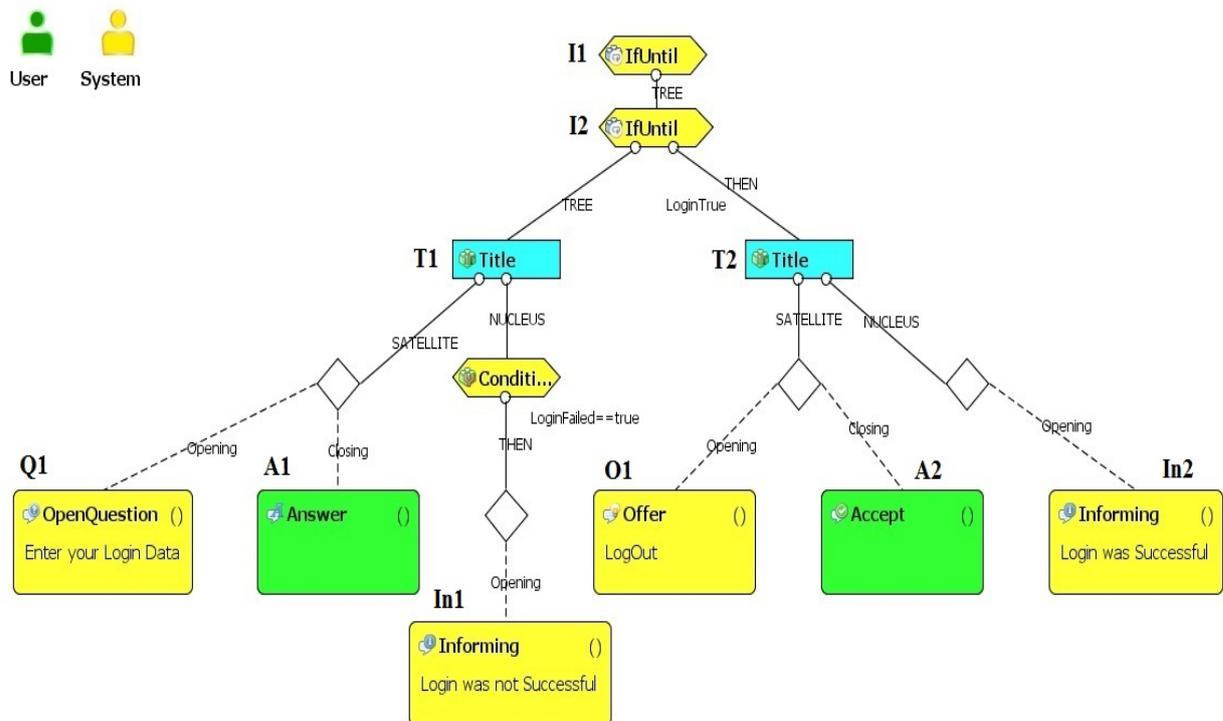


Abbildung 5.4: Discourse-Diagramm der Login-Anwendung

Die prozedurale Semantik jeder Relation wird durch eine Zustandsmaschine definiert. In dieser Arbeit beschränken wir uns auf eine textuelle Darstellung der Relationen. Für eine Spezifikation mithilfe von Zustandsmaschinen wird auf die Web-Seite von UCP⁴ verwiesen.

Die *IfUntil* [PFA⁺09] Relation kombiniert die Funktionalität einer Schleife mit der einer Verzweigung. Jedes *IfUntil* definiert drei Zweige. Diese heißen *Tree*, *Then* und *Else*. Die *IfUntil* Relation

⁴<http://ucp.ict.tuwien.ac.at/wordpress/>

wiederholt den Tree-Zweig, solange eine vordefinierte *Condition* erfüllt ist. Wenn dies der Fall ist, wechselt IfUntil zum *Then*-Zweig. Zusätzlich wird eine *RepeatCondition* definiert. Falls die RepeatCondition nicht erfüllt ist, wird die Schleife abgebrochen und der *Else*-Zweig ausgeführt. Die Ausführung einer IfUntil ist beendet, sobald der Else- oder der Then-Zweig fertig bearbeitet wurde. Die erste IfUntil (*I1*) stellt eine äußere Schleife dar, mit der die Anwendung immer wieder zu dem Anfangszustand zurückkehrt. Damit stellt sie eine unendliche Schleife dar und definiert keinen Else- oder Then-Zweig. Das zweite IfUntil (*I2*) sorgt dafür, dass die Login-Daten so oft abgefragt werden, bis die *LoginTrue* Bedingung erfüllt ist. Wenn die Bedingung erfüllt ist, wird der THEN-Zweig ausgeführt und der Anwender bekommt die für ihn vorgesehenen Daten angezeigt.

Die *Title* definiert die Zweige NUCLEUS und SATELLITE. Der NUCLEUS-Zweig stellt den Kern der Präsentation dar, während der SATELLITE-Zweig als ein Anhang dieses Kerns verstanden werden kann. Die *Title* Relation ermöglicht es hiermit, neben der Ausführung des NUCLEUS-Zweiges, der letztendlich den Endzustand vorgibt, einen zusätzlichen SATELLITE-Zweig auszuführen. Hierbei beschränkt sich der NUCLEUS-Zweig nur auf den aktuellen Zustand und spielt für den weiteren Verlauf keine Rolle. Im ersten Title (*T1*) ist dieser NUCLEUS-Zweig das Feedback zu einem fehlgeschlagenen Login-Versuch. Zusätzlich wird dieser Zweig um ein *Condition* Construct ergänzt, das überprüft, ob die Bedingung *LoginFailed==true* erfüllt ist. Damit wird erreicht, dass der Zweig erst dann ausgeführt wird, wenn das Login mindestens einmal fehlgeschlagen hat. Beim zweiten Title (*T2*), ist der NUCLEUS die Präsentation, die durch ein erfolgreiches Login dargestellt wird. Der SATELLITE-Teil kann, wie der Name auch übersetzt suggeriert, als ein Anhänger verstanden werden. Dieser Anhänger ist im Falle von T1 der erneute Aufruf zur Eingabe der Login-Daten. Die T2-Relation hängt an ihrem Kern, welcher das entsprechende Feedback zu einem erfolgreichen Login ist, einen SATELLITE-Zweig an, welches dem Kommunikationspartner die Möglichkeit eines *LogOut* bietet.

Damit das Discourse Model vollständig ist, fehlen noch die Communicative Acts. Sie stellen in der graphischen Darstellung des Discourse Models die End-Knoten und den detaillierten Inhalt der Konversation dar. Verbunden werden die Communicative Acts durch *Adjacency Pairs*. Im Login-Beispiel wurde von den Communicative Acts *OpenQuestion*, *Answer*, *Offer*, *Accept* und *Informing* Gebrauch gemacht. Bei Modellierung der Communicative Acts finden die Informationen aus dem ANM und DoD Anwendung. Damit stellt man die Verknüpfung zwischen dem Discourse Model und der Applikationslogik her. Diese Verknüpfung wird durch den *Propositional Content* dargestellt und spezifiziert hiermit, welche Information aus dem DoD Model bei einem Communicative Act von Bedeutung ist.

Im konkreten Fall der Login-Anwendung stellt *Q1* eine Frage dar, welche von dem System an dem Anwender gestellt wird. Dabei handelt es sich um die Abfrage der Login-Daten. Der Anwender antwortet auf diese Frage mit Answer *A1*. Das Propositional Content, welches bei der Modellierung von *Q1* und *A1* spezifiziert werden muss, ist in Abbildung 5.5 dargestellt und mit *Enter your Login Data* beschrieben. Dieses lautet „basic::get one LoginBeispiel::User::currentUser FOR basic::set one LoginBeispiel::User::currentUser“. Dabei sind unter *basic::get* und *basic::set* die zwei Aktionen *get* und *set* aus dem basic.anm Modell zu verstehen, durch die eine Instanz einer bestimmten Klasse aus dem DoD Model abgefragt oder gesetzt werden kann. Durch *Login-Beispiel::User::currentUser* wird festgelegt, um welche Klasse es sich dabei handelt. Diese Klasse wurde bereits im Abschnitt 5.1.1 vorgestellt. Auf *Login was not Successful* wird nachher noch eingegangen.

O1 ist ein Communicative Act des Typs *Offer*. Ein Offer stellt dem Kommunikationspartner eine vordefinierte Menge an Auswahlmöglichkeiten zur Verfügung. Im Falle von *O1* ist das die

Description	Enter your Login Data
Specification	get one User::currentUser for set User::currentUser
Specification AST	basic::get one LoginBeispiel::User::currentUser FOR basic::set one LoginBeispiel::User::currentUser
<hr/>	
Description	Login was not Successful
Specification	LoginFailed
Specification AST	LoginBeispiel::LoginFailed

Abbildung 5.5: Beispiel für Propositional Content in UCP

Möglichkeit eines LogOut dem der Kommunikationspartner zustimmen kann.

In1 und *In2* sind Beispiele für Communicative Acts vom Typ *Informing* und präsentieren dem Anwender Informationen. Diese Information kann z.B. eine Warnmeldung oder eine Feedback-Meldung sein. Für diesen Communicative Act können auch extra definierte Notifications verwendet werden, wie es auch beim *In1* und *In2* der Fall ist. Abbildung 5.5 zeigt den Propositional Content für *In1*, welcher die Beschreibung *Login was not Successful* hat. Die entsprechende Notification wurde im ANM mit dem Namen Login-Beispiel definiert und heißt *LoginFailed*.

Die Login-Anwendung kann auch durch einen einfacheren Diskurs spezifiziert werden. Die in Abbildung 5.6 dargestellte Lösung spezifiziert, dass das Login Use Case auch nach einem LoggedIn Zustand wiederholt werden kann. Voraussetzung hierfür ist, dass der Anwender sich ausloggt und hiermit in dem Anfangszustand zurückkehrt. Diese Spezifikation stellt eine erweiterte Version des Login-Beispiels dar. Die Erweiterung besteht darin, dass das gesamte Szenario iterativ spezifiziert wird. Sie soll jedoch verdeutlichen, wie einfach so ein erweitertes Verhalten in UCP spezifiziert werden kann. Wenn man auf dieses Verhalten verzichtet und sich auf einen einmaligen Login-Versuch einigt, kann das Discourse Model in Abbildung 5.6 verwendet werden. Ein Vergleich dieser beiden Spezifikationen zeigt, dass es für die erwähnte Änderung reicht, die IfUntil Relation *I1* aus dem Discourse Model in 5.6 zu streichen.

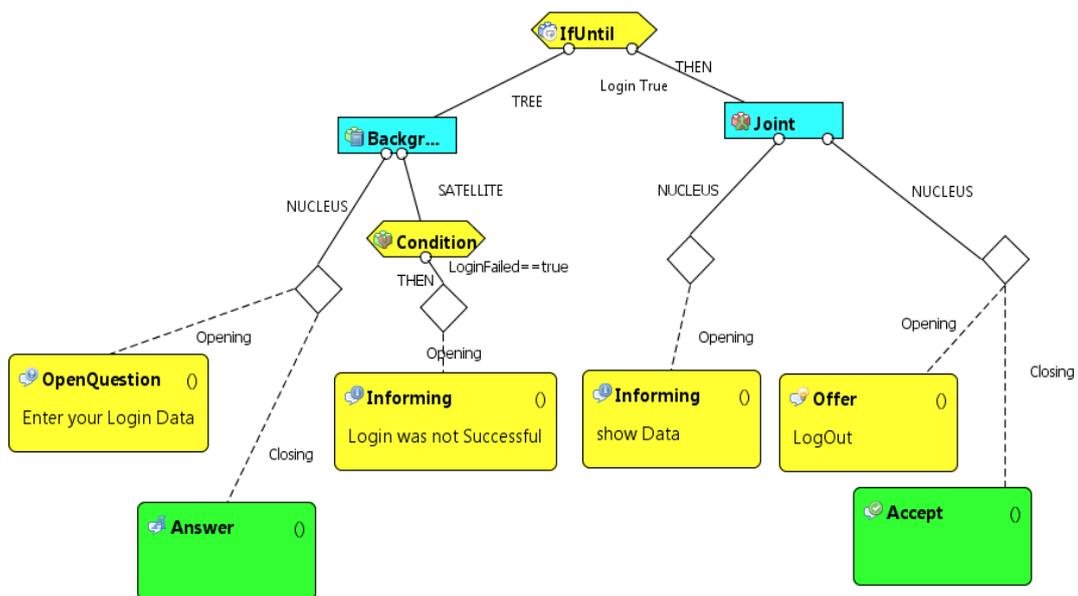


Abbildung 5.6: Alternatives Discourse Model für die Login-Anwendung

5.1.4 Generierungs-Framework für die GUIs (UCP:UI)

In diesem Abschnitt wird das Generierungs-Framework, das in Abbildung 5.7 dargestellt ist, detaillierter besprochen. Dabei werden auch die Zwischenartefakte erläutert, die von den Werkzeugen generiert werden. Auf die Werkzeuge an sich wird in Abschnitt 5.2 eingegangen. In UCP kann das Generierungs-Framework in Subteile zerlegt werden, die Eigenschaften vorweisen wie die unterschiedlichen CRF-Ebenen. Nachdem der User in der Interaktions-Spezifikation (auf der Concepts and Tasks-Ebene), die Kommunikation spezifiziert hat, erzeugt das Framework ein *WIMP UI Behavior Model* und ein *Structural UI Model* und setzt diese zu einem *Screen Model* [RPK⁺11a] zusammen. Das Screen Model wird dann in einem iterativen Prozess für ein spezifisches End-Gerät optimiert [RPK⁺11b]. Aus dem Screen Model kann der Source Code für ein GUI generiert werden, welches in der UCP-Laufzeitumgebung (UCP:RT) ausgeführt werden kann.

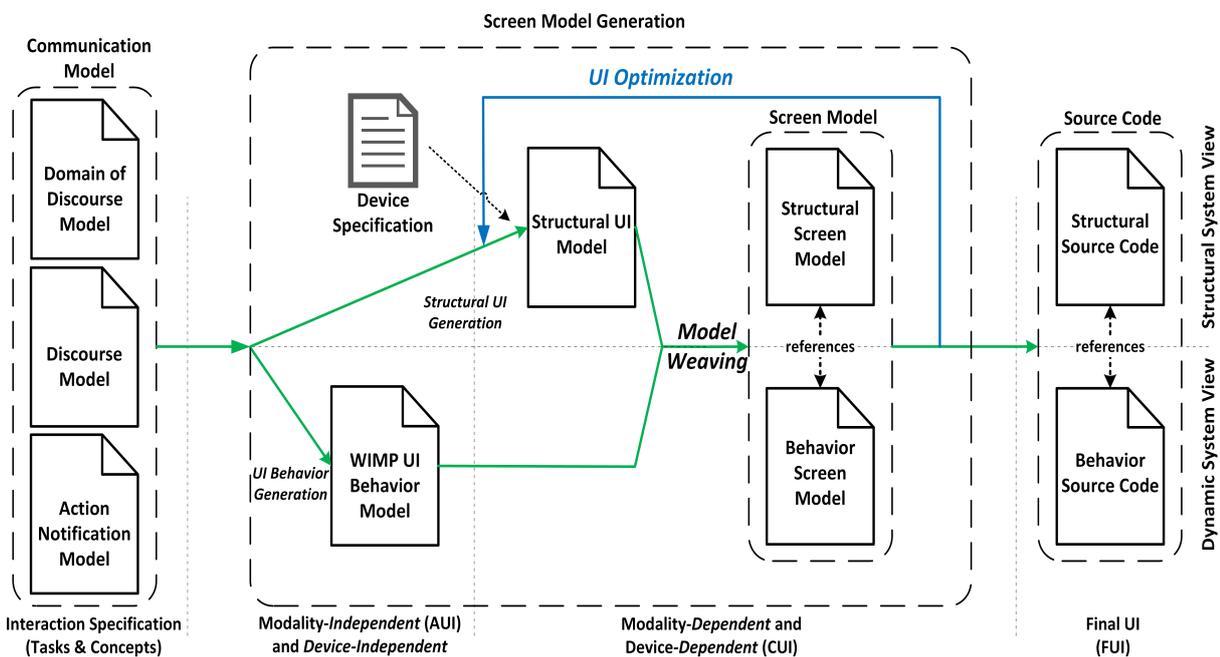


Abbildung 5.7: UCP:UI im Überblick (kopiert von [Pop12])

5.1.4.1 WIMP UI Behavior Model

Das *WIMP UI Behavior Model* definiert den Kontrollfluss der GUI Presentation Units. Dabei wird das GUI als eine Zustandsmaschine, bestehend aus mehreren Zuständen modelliert und die Transitionen zwischen diesen Zuständen in einer Controller-Klasse implementiert. Diese Zustandsmaschine wird auch als *Partitioning State Machine* (PSTM) bezeichnet [RPKF11]. Dieses GUI-Modell in UCP entspricht dem AUI im CRF, da sie unabhängig von jeder Modalität und Plattform ist. Abbildung 5.8 zeigt das Metamodell des UI Behavior Models. Die unterschiedlichen Elemente, aus denen eine Zustandsmaschine in UCP zusammengestellt ist, sind als gelbe Vierecke dargestellt.

Wie auch in Abbildung 5.7 dargestellt ist, wird mit Hilfe des PSTM aus dem Communication Model eine Zustandsmaschine gebaut, die auf dem Metamodell aus Abbildung 5.8 beruht und alle möglichen Verläufe berücksichtigt.

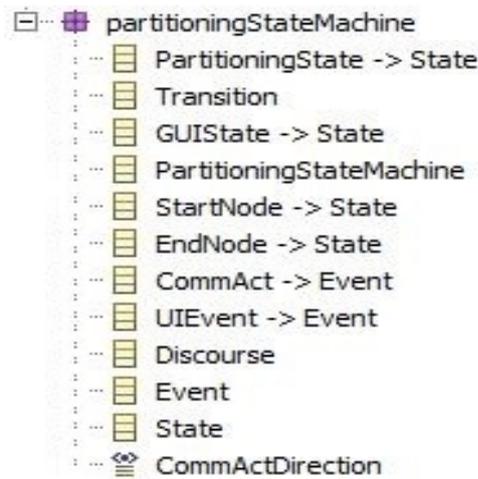


Abbildung 5.8: PSTM-Metamodell

5.1.4.2 Application-Tailored Device Specification

Bevor das Structural UI Model vorgestellt wird, soll die Device Specification erläutert werden, die in Abbildung 5.9 exemplarisch dargestellt ist. Diese erlaubt es, eine definierte Schnittstelle in eine Reihe von unterschiedlichen Plattformen umzusetzen und enthält Informationen, die das Structural UI Modell verwendet [KRF+09]. In einer *Devicespecification*-Datei kann der Entwickler die Auflösung des Bildschirmes einstellen (*Resolution* und *Dpi*), die Stylesheet-Datei angeben, die als Standard-Einstellung verwendet werden soll und angeben, ob und in welchem Ausmaß das Vielfache der Seitenlänge und Seitenbreite (*Max Scroll Height* und *Width*) zum Anzeigen der einzelnen GUI-Zustände verwendet werden soll [RPK+11b].

Property	Value
Default CSS	desktop.css
Dpi	96
Keyboard Needed	false
Max Scroll Height	1
Max Scroll Width	1
Name	Desktop
Pointing Granularity	FINE
Resolution X	1200
Resolution Y	1024

Abbildung 5.9: Devicespecification für ein Desktop-Gerät

5.1.4.3 Structural UI Model

Aus einem Communication Model wird mit Hilfe von vordefinierten Transformationen ein entsprechendes strukturelles Modell generiert. Der Structural UI darf jedoch trotz seiner Plattformabhängigkeit nicht mit einer CUI laut CRF gleichgesetzt werden, da es nur Informationen über der Struktur des GUI enthält. Erst die Zusammensetzung des WIMP UI Behavior und des Structural Model ergibt ein Modell, welches die CUI-Ebene bildet.

5.1.4.4 Screen Model und das FUI

Das WIMP UI Behavior Model und das Structural UI Model bilden zusammen das *Screen Model*.

Eine Eigenschaften des Generierungs-Frameworks ist, dass der Entwickler entscheiden kann, wie das generierte Screen Model für eine bestimmte Plattform „optimiert“ werden kann. Darunter versteht man die Anpassung eines GUI an die von der Plattform vorgegebenen Randbedingungen.

Im letzten Schritt der Generierung, wird das oben erwähnte Screen Model mit Hilfe von *JET* und *Template-Scriptsprachen* in das FUI transformiert [Ran08]. Die hierbei verwendeten Templates wurden für die PSTM mit Hilfe von *xPand* und für den Structural UI Model in der Script-Sprache *Velocity* geschrieben. Die Velocity-Vorlagen definieren hiermit die Struktur der letztendlich präsentierten UIs. Im Falle von GUIs finden sich in diesen Templates der entsprechende HTML-Code. Dieser Code wird durch einen JavaScript Teil für die dynamischen Effekte vervollständigt.

Abbildung 5.10 stellt das FUI für die Login-Anwendung dar. Das Verhalten des generierten GUIs, entspricht dem Aufbau einer Zustandsmaschine mit drei unterschiedlichen Zuständen. Ausgehend aus dem ersten Zustand, wo die Abfrage der Login-Daten stattfindet, wird nach Evaluierung des Ergebnisses in einem der beiden anderen Zustände gewechselt.

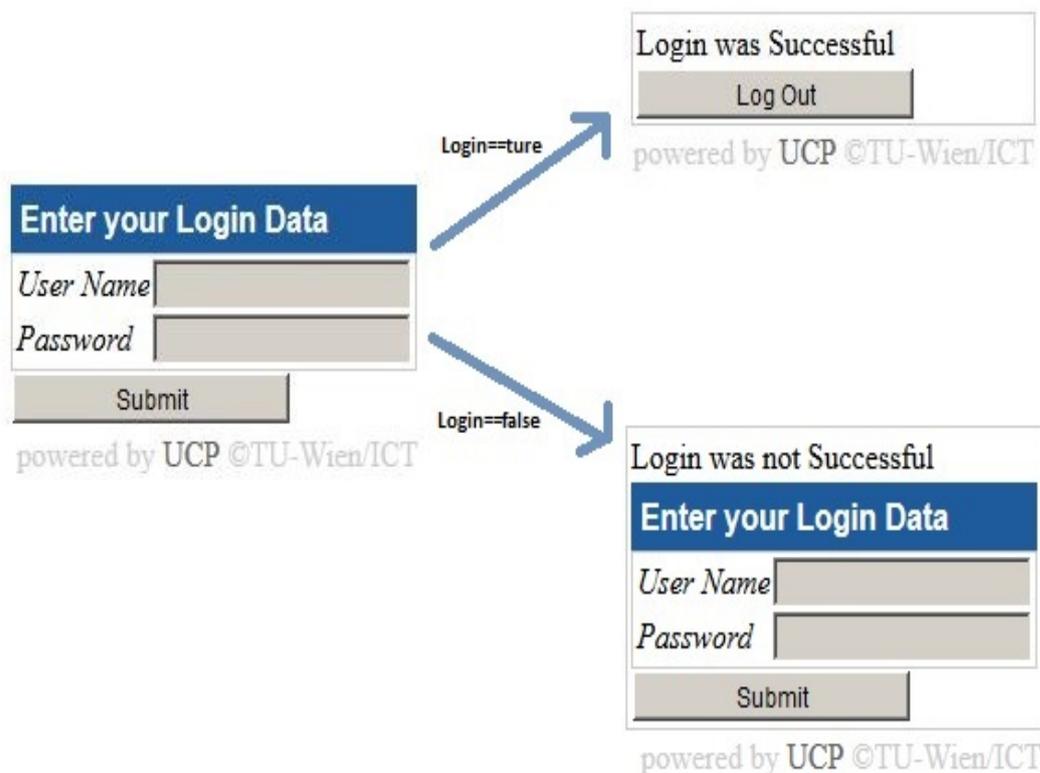


Abbildung 5.10: FUI der Login-Anwendung

Das generierte GUI kann nun ausgeführt werden und kommuniziert über die Laufzeitumgebung (UCP:RT) [Pop12] mit der Applikationslogik. Um die Applikationslogik zu definieren, muss in UCP ein eigenes Paket angelegt werden, welches die Services enthält, auf die ein GUI über UCP:RT-Komponenten zugreifen kann. Dieses Paket enthält zwei Klassen.

Die erste Klasse, die als *Application Adapter* bezeichnet wird, agiert als eine Instanz zwischen dem GUI und der Geschäftslogik. In dieser Instanz werden den Propositional Contents und damit auch den in ihnen enthaltenen Actions und Notifications entsprechende Reaktionen zugeordnet. Diese unterschiedlichen Aufrufe, entsprechen unterschiedlichen *Calls* und werden in UCP:RT auch so genannt. Diese werden, in Form von if-Abfragen im Java Code abgefragt und behandelt. In UCP:RT ist die *toString()* Methode für die Calls überschrieben worden, womit die Propositional Contents als String-Objekte bearbeitet werden können. Alle if-Abfragen im Application Adapter werden im neuesten Release der UCP-Tools automatisch von UCP:RT generiert [§14]. Hiermit muss der Entwickler nur noch die erwähnten Reaktionen bzw. die Verknüpfung zur Applikationslogik implementieren.

Die zweite Klasse enthält den Kern der Geschäftsanwendung. Damit findet man in dieser Klasse das Datenmodell und Geschäfts-spezifische Methodendefinitionen. Löst das GUI z.B. durch ein Klicken das Schicken eines Calls für einen Propositional Content aus, wird in der Adapterklasse zu der Stelle gewechselt, in der das entsprechende Propositional Content behandelt wird. An dieser Stelle kann dann auf die Applikationslogik zugegriffen werden.

5.2 Werkzeuge

In diesem Abschnitt werden die Werkzeuge in UCP vorgestellt. Diese Vorstellung stellt sich aus zwei Teilen zusammen. Im ersten Teil werden die Editoren für die unterschiedlichen Modellen in UCP besprochen. Der zweite Teil gibt dem Leser einen Überblick über die Werkzeuge des Generierungs-Frameworks (UCP:UI).

5.2.1 Editoren

Bezüglich Werkzeuge bietet UCP zur Unterstützung der Entwickler, Plugins und Editoren für Eclipse. Diese Editoren unterstützen den Aufbau von allen Eingangsmodellen, die zusammen das Discourse based Communication bilden [BKFP08].

Für das DoD Model wird der Standard Editor für das Ecore-Modell verwendet. Dieser Editor wird von Eclipse Modeling Framework zur Verfügung gestellt. Für das Discourse Model und ANM wurden eigene, grafische Editoren gebaut. Abbildung 5.11 stellt den Editor für das Discourse Model dar. Der Editor und die Tool-Box auf der rechten Seite sind als Eclipse Plugins realisiert. Der Editor ermöglicht eine interaktive Abfrage der Eigenschaften einzelner Knoten. Die hier dargestellten *Properties* gehören dem ersten IfUntil Knoten. Der Editor ermöglicht es, eine direkte Anpassung dieser Eigenschaften durchzuführen. In der Tool-Box auf der rechten Seite können beliebige UCP-Relationen ausgewählt und dem Model hinzugefügt werden.

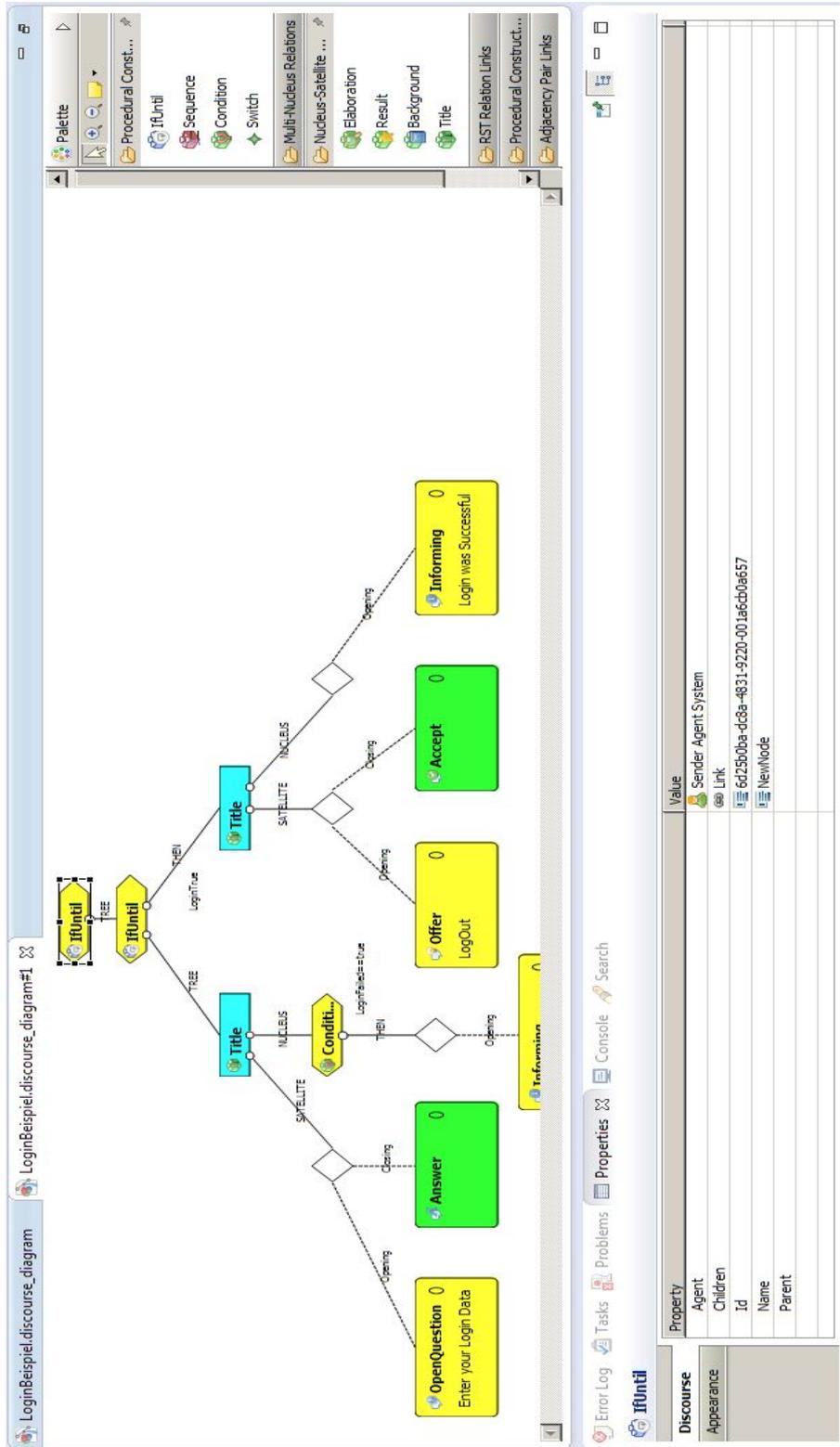


Abbildung 5.11: Discourse Model Editor

5.2.2 UCP:UI Werkzeuge

UCP stellt zur Unterstützung des Entwicklers bei der Generierung eines FUIs eine graphische Oberfläche zur Verfügung. Mit Hilfe dieser Oberfläche kann der Entwickler verschiedene Parameter des Generierungs-Frameworks einstellen. Diese Oberfläche ist als ein Eclipse-Plugin implementiert. Die Tool-Unterstützung wird in den Abbildungen 5.12 und 5.13 illustriert.

Abbildung 5.12 zeigt eines von mehreren Fenstern des Werkzeugs, nämlich jenes mit den allgemeinen Eigenschaften. In den *General* Einstellungen sind die drei Panels P1, P2 und P3 eingebaut. In P1 kann der Anwender im ersten Schritt festlegen, welches Discourse Model als Eingangsmodell verwendet werden soll. In der linken Spalte hat man die Möglichkeit die Schritte auszuwählen, die das *Transformation Engine* ausführen soll. In P2 kann der Anwender die Ziel-Dateien für die WIMP UI Behavioral und Structural UI Model festlegen, welche von dem Transformation Engine generiert werden. In P3 kann dem Tool das DoD Model angegeben werden.

Abbildung 5.13 zeigt die Einstellungen zur Generierung des Structural UI Models. Hier gibt es drei Bereiche D1, D2 und D3. In D1 kann die Devicespecification angegeben werden. Falls noch zusätzliche Regeln bei der Transformation der Eingangsmodelle zu berücksichtigen sind, können diese hier ebenfalls eingetragen werden. In D2 kann festgelegt werden, für welchen der beiden Kommunikationspartner das GUI generiert werden soll. In D3 kann eine Regel vorgegeben werden, welches von dem Transformation Engine verwendet wird, um das Screen Model zu optimieren. In Abbildung 5.13 ist die Option *ScreenBasedDeviceOptimization* ausgewählt.

Das *UI Code Generation*-Fenster ermöglicht die Plattform auszusuchen, für welche der Source Code generiert werden soll. UCP erlaubt derzeit nur die Generierung von HTML-Code. Hier können zusätzlich eigene Style Sheets eingebunden werden und der Ziel-Ordner für den generierten Code festgelegt werden.

Das Fenster mit dem Namen *Runtime* in der Tab-Bar erlaubt die Spezifikation des Services, welches vom Kommunikationspartner angeboten wird. Falls es sich um eine Mensch-Maschine Kommunikation handelt, ist darunter die Applikationslogik zu verstehen, die auf der Maschine läuft.

Im *Common*-Fenster können die eingegebenen Eigenschaften gespeichert werden.

5.2.3 UCP Runtime

Das in den vorherigen Abschnitten vorgestellte Back-End und die erzeugten GUIs in UCP setzen zur Kommunikation miteinander auf die UCP Runtime (UCP:RT) [Pop12]. Die Laufzeitumgebung übernimmt dabei Aufgaben, die im MVC Pattern der *Control*-Einheit zugeordnet sind.

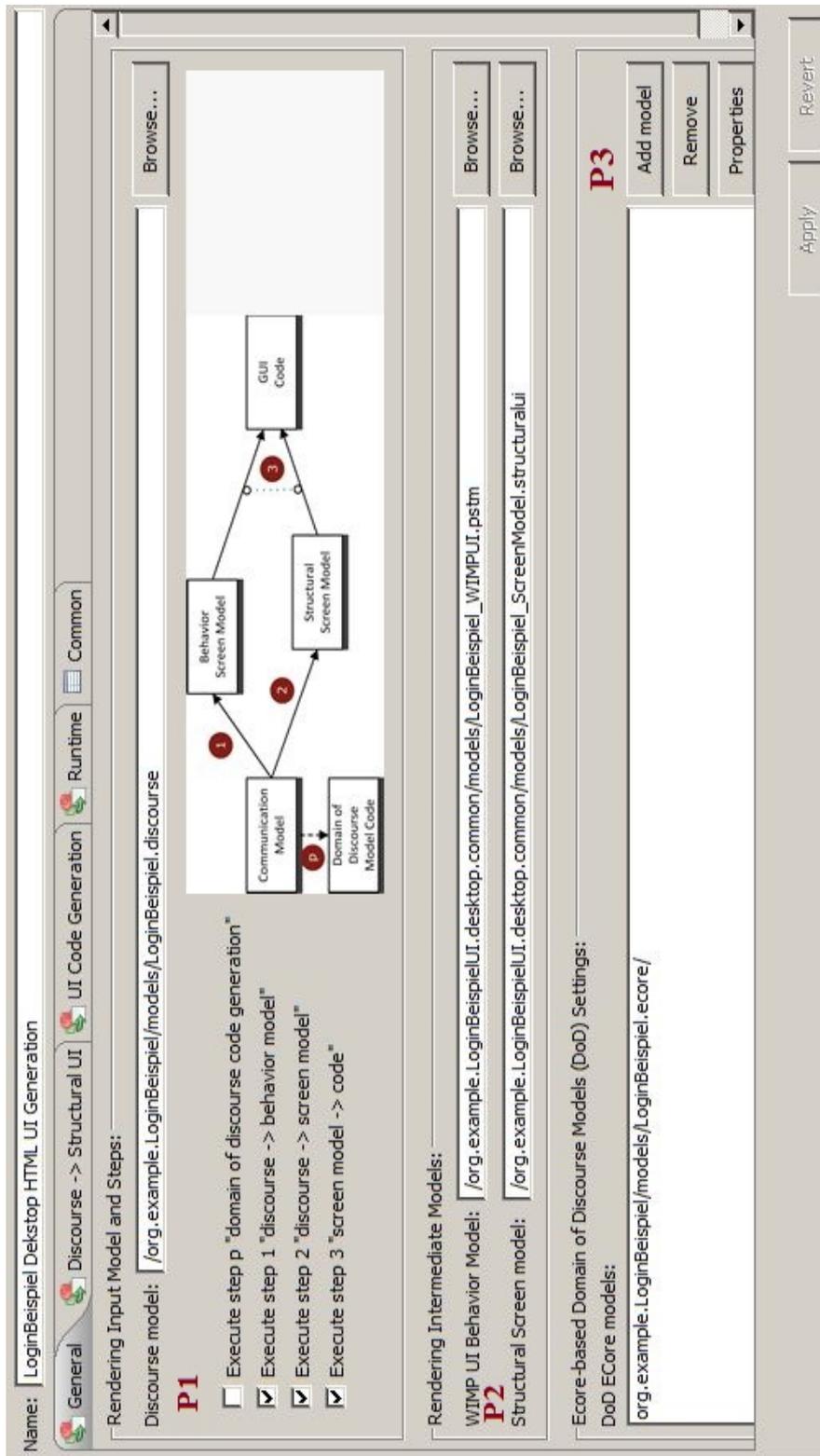


Abbildung 5.12: Tool-Unterstützung zur Generierung eines FUIs in UCP

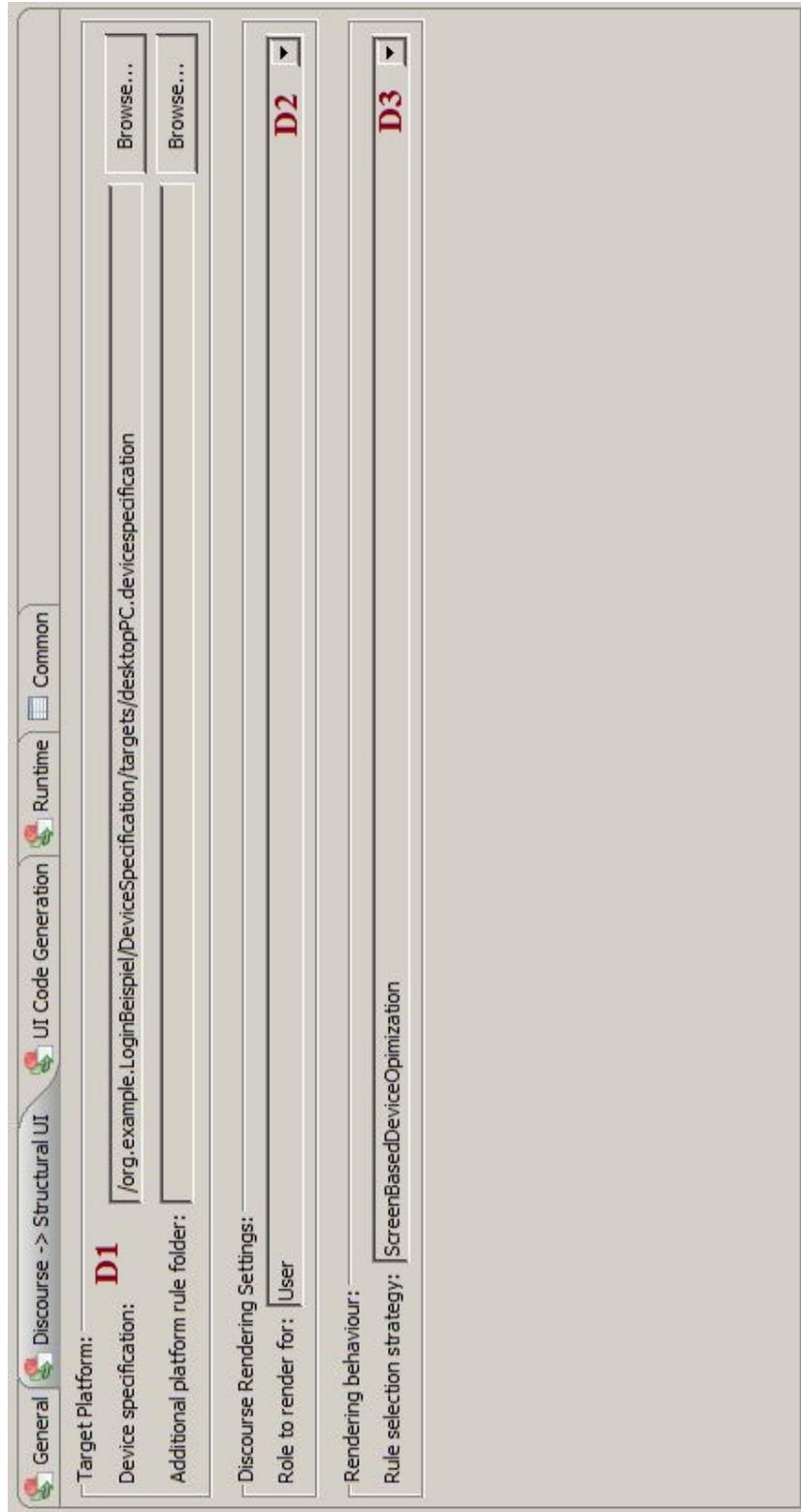


Abbildung 5.13: Weitere Tool-Unterstützung zur Generierung eines FUIs in UCP

6 VERGLEICH DER UNTERSCHIEDLICHEN ANSÄTZE

In diesem Kapitel werden die in den vorherigen Kapiteln vorgestellten Ansätze miteinander verglichen. Dieser Vergleich besteht, wie schon in der Einleitung erwähnt, aus zwei Teilen. Der erste Teil stellt einen konzeptionellen Vergleich und der zweite einen praktischen Vergleich der MDUID-Ansätze dar.

6.1 Konzeptioneller Vergleich

Der konzeptionelle Vergleich stellt den ersten Teil des Vergleichs dar. In den folgenden Abschnitten wird erläutert, warum der Vergleich mit Hilfe des CRF nicht ausreichend ist, um die Ansätze zu charakterisieren. Für diese Aufgabe, werden zusätzliche Vergleichs-*Kriterien* benötigt, die vier verschiedenen Vergleichs-*Kategorien* angehören.

6.1.1 CRF-basierter Vergleich

Die Abbildungen 6.1 bis 6.3 stellen die CRF-basierte Klassifizierung der drei vorgestellten Ansätze dar. Dargestellt sind die verschiedenen Modelle, die in den Ansätzen definiert werden. Zusätzlich wurden die Reifications und Abstractions eingezeichnet, sofern sie von den Ansätzen unterstützt werden. In diesem Abschnitt soll die CRF-basierte Klassifizierung gleichzeitig zur Wiederholung der wesentlichen Bestandteile der Ansätze dienen. Wie wir noch sehen werden, zeigt sich, dass die CRF-Klassifizierung sich als nicht ausreichend erweist, um alle Eigenschaften der Ansätze wiederzugeben. Wie schon im Abschnitt 2.3 erwähnt, gibt das CAMELEON Reference Framework, ein standardisiertes Schema zur Klassifizierung von Modellen und Prozessen wieder, die in dem jeweiligen Ansatz zur Beschreibung von GUIs auf unterschiedlichen Abstraktionsebenen verwendet werden können.

Zuerst soll kurz eine Erklärung zu den Pfeilen in den CRF-Darstellungen gegeben werden. In Zusammenhang mit CRF wurden die Pfeile, die nach unten zeigen, als „Reification“ definiert. Es soll hier erwähnt werden, dass die Darstellungen in den Abbildungen 6.1 bis 6.3 zwar dieser Definition folgen, jedoch gleichzeitig auch die Abhängigkeiten zwischen den Abstraktionsebenen wiedergeben. Diese Abhängigkeiten sind für jede Iteration während der Entwicklung gegeben. Konkret zeigen die Pfeile in der folgenden Darstellung immer auf jenes Modell, zu dem sie beitragen bzw.

in dem ihr Inhalt „einfließt“. Den grauen Pfeilen kommt dabei eine andere Bedeutung zu. Sie ordnen bestimmte Instanzen der Metamodelle den Ebenen des CRF zu. Genauer gesagt kennzeichnen sie jene Metamodelle, deren Instanzen Teil einer bestimmten Ebene des CRF sind.

Die Abbildungen 6.1 bis 6.3 stellen alle Abstraktionsebenen für die unterschiedlichen Ansätze dar. Unter *Ontological Models* werden Konzepte aus den Metamodellen verstanden. In der Spalte *Config* kommen diese Modelle noch einmal vor, wobei sie dieses Mal eine Instanz des jeweiligen Metamodells darstellen, welches für einen bestimmten Kontext zutrifft.

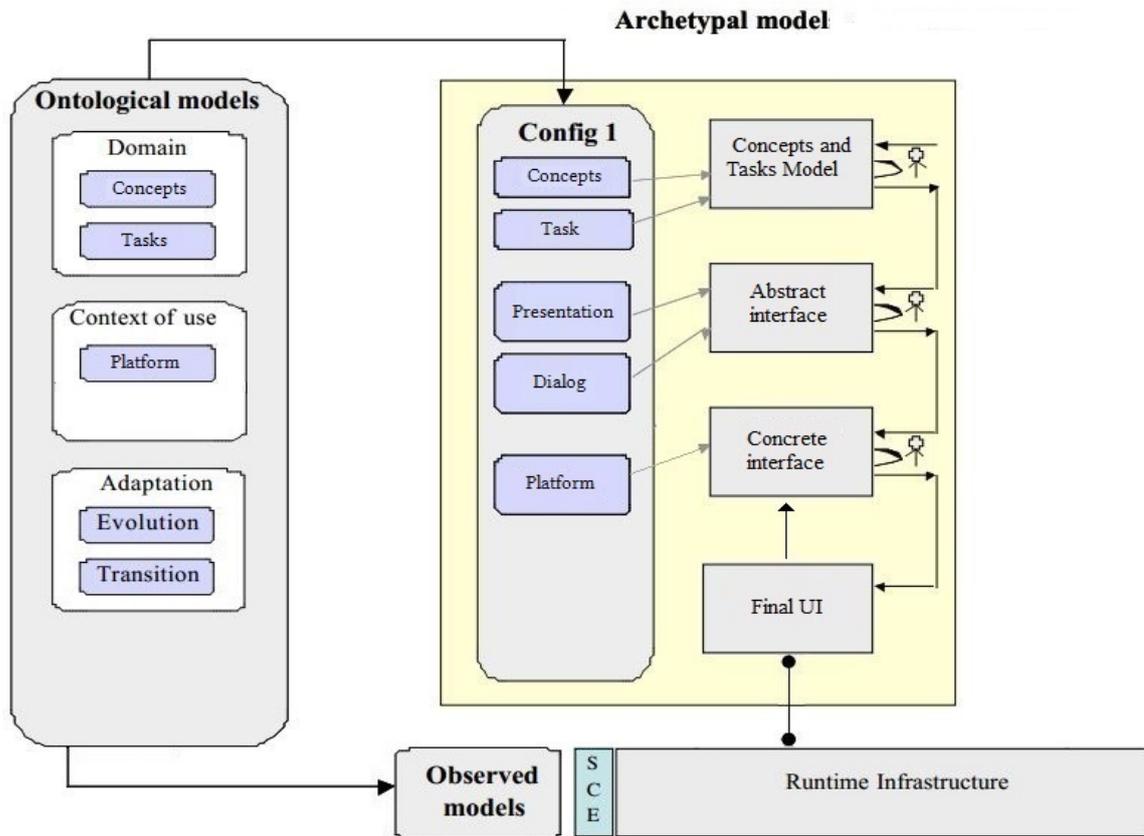


Abbildung 6.1: CRF-Darstellung für MARIA

Abbildung 6.1 zeigt die CRF-Darstellung für *MARIA* [CCT+03b]. Die *Concepts and Tasks* Ebene in *MARIA* besteht aus einem Task-Modell und den zusätzlichen Modellelementen des CTT (*Concepts*), mit denen man das Task-Modell z.B. um die Definition von Conditions ergänzen kann. Aus dem *Concepts and Tasks Model* kann ein *AUI* generiert werden. Dieses basiert auf einem Datenmodell und einem Event-Modell, welches den Informationsfluss während einer Interaktion definiert. Dieser Teil des *AUI*-Modells in *MARIA* wird aus diesem Grund auch als *Dialog*-Modell bezeichnet, obwohl es, im Gegensatz zur UCP keinen Diskurs zwischen zwei Kommunikationspartner darstellt. Das erzeugte *AUI* besteht aus mehreren *Presentation Task Sets (Presentation)*, die die Struktur der GUIs definieren. Aus dem *AUI* kann durch die entsprechende Transformation das *CUI* generiert werden. Dieses enthält modalitätsabhängige Elemente. Diese Informationen extrahiert die Transformation aus dem Modell, welches in der Abbildung als *Platform* bezeichnet wird und Teil des *CUI*-Metamodells in *MARIA* ist. Für eine detaillierte Beschreibung zu diesen Begriffen wird auf Abschnitt 3.1 verwiesen.

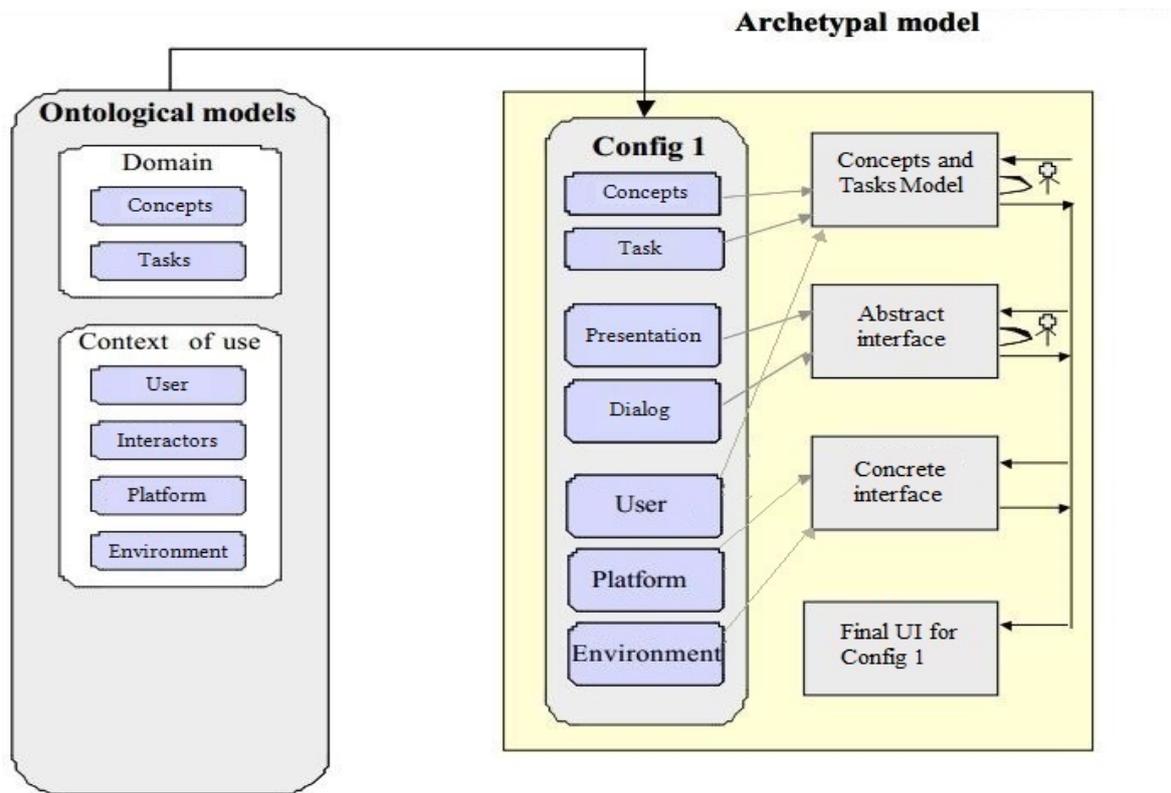


Abbildung 6.2: CRF-Darstellung für UsiXML

In Abbildung 6.2 ist das CRF für *UsiXML 2.0* zu sehen. Auf der Concepts and Tasks Ebene besteht UsiXML wie schon im Abschnitt besprochen 4.1 aus einem Task-Modell und einem Context-Modell. Auf der AUI-Ebene werden die entsprechenden Elemente aus dem AUI-Metamodell herangezogen, um die Struktur (Presentation) und den Kontrollfluss der Informationen (Dialog) darzustellen. Für die Generierung des CUIs, werden abgesehen von Informationen zu der *Platform* die im Context-Modell spezifizierten *User* und *Environment* mitberücksichtigt. Da für UsiXML 2.0 weder vordefinierte Transformationen noch Werkzeuge zur Transformation von Modellen vorhanden sind, wurden in der CRF-Darstellung die Pfeile so eingezeichnet, dass die Generierung aller Modelle theoretisch möglich ist. Dies erfordert jedoch die manuelle Implementierung der Transformations-Logik.

Abbildung 6.3 zeigt das CRF für *UCP*. Hierbei wurde versucht die Eingangsmodelle und die Modelle, die während des Generierungsprozesses entstehen, den verschiedenen CRF-Ebenen zu zuordnen. Die Diskurs-basierte Kommunikation in UCP befindet sich auf der Concepts and Tasks Ebene. Dabei besteht diese, wie schon in Kapitel 5.1 ausführlich dargestellt, aus dem DoD Modell, ANM und aus dem Discourse Model. Im nächsten Schritt wird aus dem Diskurs-basierten Kommunikationsmodell das *WIMP UI Behavioral Model* generiert, welches eine Instanz der Partitioning State Machine ist. Dieses Modell ist modalitäts- und technologieunabhängig und bildet damit das AUI in UCP. Das *Screen Model* in UCP besteht aus einer *PSTM*, einem *Structural UI Model* und einer *Application Tailored Device Specification*. Das Screen Model setzt auf dem *WIMP UI Behavioral Model* auf und ist abhängig von der Modalität einer Interaktion. Damit entspricht es laut CRF einem CUI.

Da sich diese Arbeit mit dem Vergleich der unterschiedlichen Ansätze beschäftigt, stellte sich

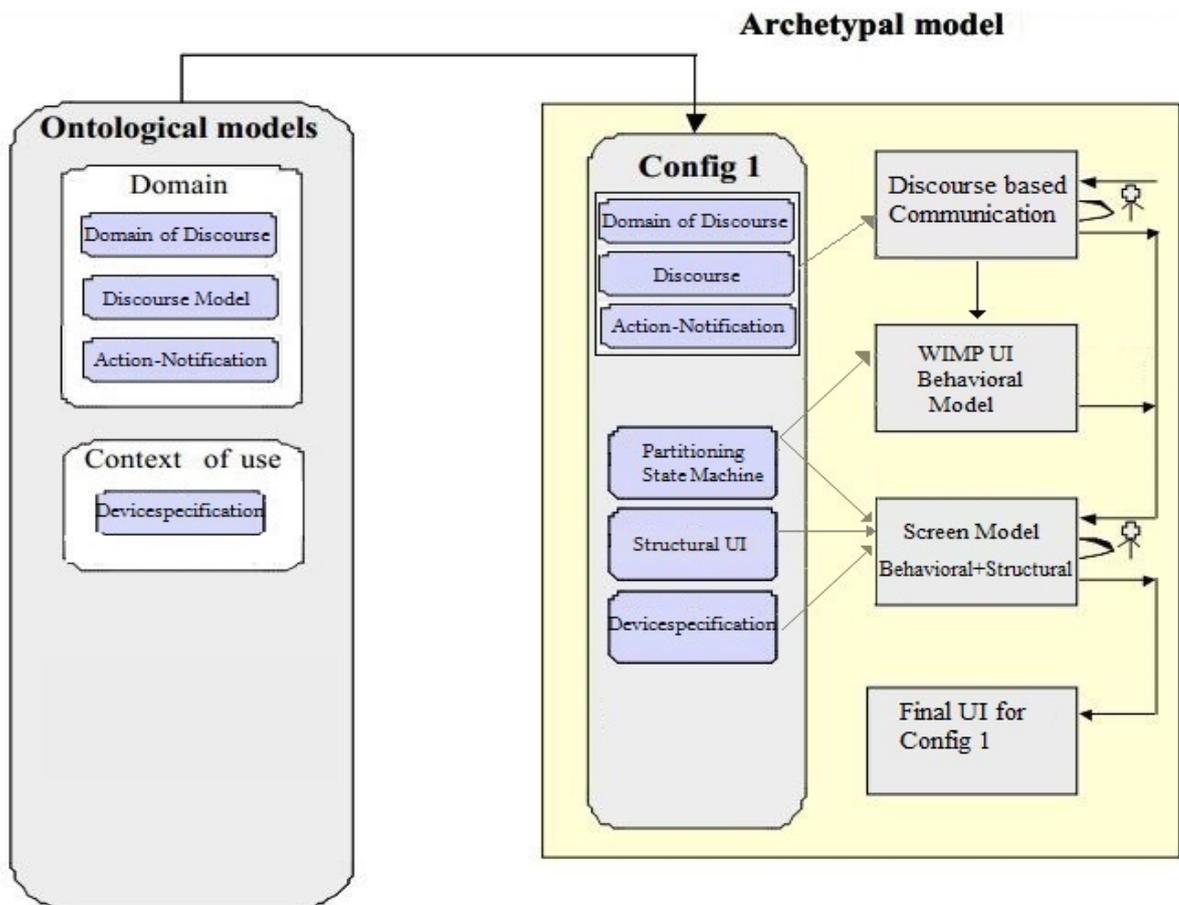


Abbildung 6.3: CRF-Darstellung für UCP

hierbei die Frage, wie geeignet das CRF für diesen Vergleich ist. Es stellte sich heraus, dass das CRF zwar zur Klassifizierung der einzelnen Ansätze und der übersichtlichen Darstellung geeignet ist, jedoch viele Eigenschaften der Ansätze nicht mitberücksichtigt bzw. keine entsprechende Darstellungsmöglichkeiten für diese Eigenschaften bietet. Aus diesen Gründen wurden zusätzliche Kriterien definiert, die genau die Eigenschaften ansprechen sollen, die nicht mit Hilfe des CRF verglichen werden können. Im nächsten Abschnitt werden diese Eigenschaften kurz vorgestellt, um sie in den darauf folgenden Abschnitten genauer zu behandeln. Ein Beispiel für so eine Eigenschaft, welche es notwendig machte, neue Kriterien zum Vergleich der Ansätze heranzuziehen, ist, dass für UsiXML 2.0 keine Werkzeuge verfügbar sind. Es fehlte in CRF die Möglichkeit, um diese Eigenschaft darzustellen.

6.1.2 Vergleichskriterien

In diesem Abschnitt soll kurz auf die Kategorien eingegangen werden, die in diesem Kapitel die Grundlage zum Vergleich der verschiedenen Ansätze bilden. Jede Kategorie setzt sich aus mehreren Kriterien zusammen. Der Vergleich der Ansätze erfolgt im Konkreten mit Hilfe dieser Kriterien.

Die erste Kategorie an Vergleichs-Kriterien ist die der *allgemeinen Eigenschaften* eines Ansatzes. Diese soll als eine Antwort auf die Frage verstanden werden, „wie es mit den Randbedingungen eines Ansatzes aussieht“. Die Bedeutung dieser Frage wird klar, wenn man sich noch einmal

vor Auge führt, dass es für ein System theoretisch viele verschiedene Einsatzkontexte gibt. Als „Kontext“ soll hier ein Tripel (*User, Platform, Environment*) verstanden werden. Man kann sich vorstellen, dass abhängig davon die Modalität der Interaktion und die Plattform, über der die Interaktion erfolgt, variieren können. Aus diesem Grund sollte das GUI für den Kontext optimiert werden. Diese Kategorie soll hiermit Klarheit darüber schaffen, ob ein Ansatz *Multimodale UIs* unterstützt, welche *Plattformen* und welche Art an *Geschäftsanwendungen* von einem Ansatz unterstützt werden. Weiterhin soll beurteilt werden, welche Arten an *Dokumentation* für einen Ansatz vorliegen und in welchem *Entwicklungsstatus* sich dieser befinde.

Als zweite Kategorie sind hier die *Modelle* zu erwähnen. Wie schon im Unterabschnitt 2.2 erwähnt, kommt den Modellen in einem modellgetriebenen Ansatz eine besondere Bedeutung zu. Diese bilden nämlich die Strukturen, auf denen die Methoden aufbauen. Unter Modelle sollen hier zwei verschiedene Arten unterschieden werden: die Eingangsmodelle und die Zwischenmodelle.

Das Eingangsmodell bildet die Grundlage für eine MDUID und ist der Ausgangspunkt einer Entwicklungs-Prozesskette. Durch Transformationen können aus einem Eingangsmodell weitere Modelle generiert werden. In CRF bieten sich z.B. als Eingangsmodelle die Modelle, die der Ebene des Concepts and Tasks zugeordnet sind. Darauf basierend können die AUI und CUI generiert werden, die Zwischenmodelle darstellen. Es soll hier erwähnt werden, dass in dieser Arbeit die Zwischenmodelle zu der Gruppe der Zwischen-Artefakte einer modellgetriebenen Vorgehensweise gezählt werden.

Jedes Modell sollte eine Reihe qualitativer Eigenschaften erfüllen. Vanderdonck [Van08] definiert verschiedene Kriterien, denen ein Modell bzw. ein Metamodell genügen sollte. Die *Vollständigkeit* und die *Kompaktheit* sind Beispiele für solche Kriterien. Diese Kriterien werden auch in [GM13] erwähnt. [VG06] beschäftigt sich mit Komplexitätsmetriken für Geschäftsprozess-Modelle und definiert einige Kriterien für die Bewertung deren Qualität. Diese Kriterien stellen jedoch Merkmale dar, die nicht die Qualität der Ansätze und Metamodelle betreffen, sondern eher die Instanzen der Modelle. Ein Beispiel solch eines Merkmales ist die Größe eines Modells. Die Größe eines Modells bestimmt sich aus der Anzahl an Knoten.

Diese Kriterien sind, obwohl sie wichtige Qualitätsmerkmale von Modellen darstellen, nicht immer einfach zu messen bzw. messbar zu machen. Daher können diese für den Vergleich der unterschiedlichen Ansätze nicht herangezogen werden. Aus diesem Grund beschränkt sich der Vergleich der Modelle in dieser Arbeit auf das Vorhandensein unterschiedlicher Modell-Typen, die für die Modellierung eines GUIs von Bedeutung sind [VJ99]. Hiermit geben diese Modelle die Kriterien dieser Kategorie an und umfassen:

- Task-Modelle
- Domain-Modelle
- Strukturelle Modelle der GUIs
- Verhaltens-Modelle
- Plattform-Modelle
- Umgebungs-Modelle
- User-Modelle

Eine weitere Kategorie bilden die *Werkzeuge*, die für die Ansätze verfügbar sind. Dieser Aspekt stellt die dritte Kategorie zum Vergleich der Ansätze dar. Die Werkzeuge für die Ansätze sollten dem Anwender helfen, mit möglichst geringem Aufwand Modelle zu erstellen. Wesentliche Kriterien in diesem Zusammenhang sind die Unterstützung von *Modellen und Transformationen*, Unterstützung unterschiedlicher *Development Paths*, die Möglichkeit der *Validierung* und der *Verifikation* der erstellten Modelle, die Unterstützung zur *Anpassung* eines Modelles und Tool-Unterstützung zur *Integration* des Back-End der Anwendung. Hierbei weisen verschiedene Ansätze, sofern die dazugehörigen Werkzeuge überhaupt frei verfügbar sind, große Unterschiede auf. Diese Unterschiede sollen hier aufgezeigt werden.

Die vierte und letzte Kategorie ist auf den generierten GUI-Source Code bezogen. Dieser Aspekt wird in vielen Studien, obwohl er einen Punkt darstellt, der unter Umständen sehr zeit- und kostenaufwendig sein kann, nicht ausreichend angesprochen. Es kann nach der Generierung der GUIs vorkommen, dass diese Fehler in der Implementierung beinhalten. In diesem Fall sollte es im MDUID nicht die Aufgabe des Entwicklers sein, sich mit der Technologie beschäftigen zu müssen, in der das GUI realisiert wurde, um vorliegende Fehler zu korrigieren. Dies könnte jedoch notwendig sein, wenn z.B. der generierte Source Code Fehler beinhaltet, die vom Code-Generator erzeugt wurden. Je mehr sich der Entwickler im letzten Schritt mit dem FUI beschäftigen muss, umso stärker verliert der generierte Source Code an Wert. Hiermit stellt die *Fehlerfreiheit der Transformationen* das erste Kriterium dieser Kategorie dar. Das Thema der generierten *Hilfefunktionen* ist das zweite Kriterium dieser Kategorie. Ein weiterer Punkt, der zwar nicht explizit in dieser Kategorie behandelt wird, aber ein sehr wichtiges Kriterium darstellt, ist Usability. Dieses Kriterium stellt eines der größten Hindernisse für ein breiteren Erfolg der MDUID-Ansätze dar, da die generierten GUIs zurzeit eine schlechte Usability vorweisen.

6.1.3 Kriterien-basierter Vergleich

Es folgt ein detaillierter Vergleich der Ansätze basierend auf den definierten Vergleichskriterien.

6.1.3.1 Allgemeine Eigenschaften

Dieser Abschnitt beschäftigt sich mit den allgemeinen Eigenschaften der Ansätze, die nicht durch CRF dargestellt bzw. charakterisiert werden können. In der Tabelle 6.1 sind die Kriterien dieser Kategorie und eine Zusammenfassung des Vergleiches dargestellt. In den folgenden Abschnitten werden diese Ergebnisse im Detail erläutert.

Multi Device und Multimodalität

MARIA unterscheidet zwischen Desktop, Mobile und Distributed als Endgeräte. Während dieser Ansatz für unterschiedliche Endgeräte und Modalitäten entsprechende Transformationen bietet, liegen diese nicht in einer Form vor, in der der Entwickler diese lesen oder beliebig bearbeiten kann. Diese transformieren das Eingangsmodell in die entsprechenden Zwischenmodelle. Während das AUI nur abstrakt, unabhängig von dem Endgerät, die Ein- und Ausgaben beschreibt, enthält das CUI die den einzelnen Endgeräten entsprechenden Interaktionsobjekte. Aus dem CUI wird in letzter Instanz das FUI generiert, welches an die Ressourcen eines Endgerätes angepasst werden muss. Wie es schon für die Transformationen der Fall war, gibt es zur Anpassung eines CUIs an die Ressourcen, die durch das Endgerät gegeben sind, keine explizite Möglichkeit bzw. Werkzeuge. In

Tabelle 6.1: Allgemeine Eigenschaften

Kriterium	MARIA	UsiXML	UCP
Multi Device	Desktop: ✓ Mobile: ✓ Multitarget: ✓	Desktop: ✓ Mobile: ✓ Multitarget: ✓	Desktop: ✓ Mobile: ✓ Multitarget: ✓
Multimodalität	✓	✓	-
Entwicklungsstatus	Forschung: ✓ Industrie: ✓	Forschung: ✓ Industrie: -	Forschung: ✓ Industrie: -
Dokumentation	Getting Started: ✓ Metamodelle: ✓ Werkzeuge: ✓ Publikationen: ✓ Screencasts: ✓	Getting Started: - Metamodelle: ✓ Werkzeuge: - Publikationen: ✓ Screencasts: ✓	Getting Started: - Metamodelle: ✓ Werkzeuge: - Publikationen: ✓ Screencasts: -
Unterstützte Anwendung	Prozess-basiert: ✓ Collaborative: ✓	Prozess-basiert: ✓ Collaborative: -	Prozess-basiert: ✓ Collaborative: -

[PZ10] definiert MARIAE im Zusammenhang mit dem dort vorgestellten „Migration Framework“, das sogenannte „Parametric Bidimensional Semantic Redesign“ [PSS10]. Dieses Werkzeug wird von dem Migration Server eingesetzt, um ein CUI an mobile Endgeräte anzupassen. Laut [PZ10] überprüft der Migration Server die CUIs auf ihren Ressourcenverbrauch, auf den sogenannten „resource costs“ und falls diese Kosten höher sind als die verfügbaren Ressourcen, werden bestimmte Interaktionsobjekte ausgewechselt, Texte umformatiert und im letzten Schritt, falls notwendig, Presentation Tasks in kleinere Presentation Tasks zerlegt. Der Migration Server setzt jedoch im Vorhinein das Vorhandensein einer Web-Version des GUIs voraus, um daraus das entsprechende mobile-GUI zu generieren.

Zwecks Anpassung eines GUI an eine Plattform setzt *UsiXML* in seinem Transformations-Modell auf das QOC Pattern. Hierbei können vordefinierte Plattform-Anpassungen als unterschiedliche Möglichkeiten betrachtet werden, aus denen basierend auf einem Entscheidungskriterium die am nächsten liegende Lösung zum Rendering des GUIs ausgesucht wird.

UCP setzt zwecks Anpassung des Strukturellen Modells an den Gegebenheiten des Endgerätes das Application Tailored Device Specification-Modell ein. In diesem Modell kann der Entwickler unterschiedliche Eigenschaften des Endgerätes spezifizieren. Zu diesen Eigenschaften gehören die Größe der Anzeige, Pointing Granularity (kann abhängig davon ob das Endgerät z.B. mit Finger oder mit der Mouse bedient wird, unterschiedlich gesetzt werden [KRF+09]) und Default Scroll Width. Zusätzlich kann die Ziel-Plattform spezifiziert werden. Im Login-Beispiel wurde als Zielplattform die HTML-Technologie spezifiziert. Bearbeitet werden diese Spezifikationen direkt in einem Eclipse Editor. *UCP* bietet hiermit als einziger Ansatz sowohl vordefinierte Spezifikationen, um das GUI an verschiedene Geräte anzupassen, als auch eine generische Möglichkeit, diese Spezifikation wiederzuverwenden und mit Hilfe von Optimierungsalgorithmen, während der Generierung an andere Endgeräte anzupassen [RPK+11b].

Es soll hier noch kurz auf die unterstützten Modalitäten eingegangen werden. Die drei Ansätze zeigen in der Unterstützung von unterschiedlichen Modalitäten und den hierfür notwendigen Entwicklungstools eine ähnliche Vorgehensweise. Es ist klar zu erkennen, dass die zwei Modalitäten der vokalen und graphischen Interaktion, die Hauptthemen bilden. MARIA und UsiXML bieten beide die Modelle, um Benutzungsschnittstellen für unterschiedliche Modalitäten zu generieren.

Während die *Concrete Style* und *Concrete Listener* Klassen es erlauben, UsiXML auf beliebige Modalitäten zu erweitern, verfolgt MARIA diesbezüglich eine weniger generische Vorgehensweise und bietet nur vordefinierte Modalitäten an. Diese umfassen eine vokale, graphische und eine multimodale Interaktion, als eine Mischung dieser beiden Modalitäten. UsiXML sieht in seinem Metamodell zusätzlich eine Gestik-basierte Interaktion vor. Außerdem sind im Metamodell bestimmte Klassen mitberücksichtigt, die die Implementierung mulitmodaler UIs erlauben. Dass die Unterstützung unterschiedlicher Modalitäten auch in UCP möglich ist, zeigt [Ert11]. Hier wurden die theoretischen Möglichkeiten aufgezeigt, mit denen man in UCP, basierend auf den Discourse-Modellen, multimodale Benutzungsschnittstellen generieren kann. Es ist derzeit jedoch für UCP keine Tool-Unterstützung für multimodale Interaktionen integriert.

Dokumentation

MARIA erlaubt es, nach einer kostenlosen Registrierung auf <http://girove.isti.cnr.it/tools/Mariae/>, Werkzeuge und Dokumentationen herunterzuladen und einzusetzen. Außerdem stehen dem Benutzer wissenschaftliche Publikationen zum Thema CTT und MARIAE zur Verfügung. [PSS11] bietet eine strukturierte und detaillierte Einführung in MARIAE.

Für *UsiXML* wurde ein sogenannter *End User Club* eingerichtet. Die Mitglieder dieses Clubs bekommen die entsprechenden Rechte für die Benutzung der UsiXML 2.0 Werkzeuge. Der Erhalt der vorliegenden Dokumentationen und Werkzeuge, zu denen auch ein *UsiXML Reference Manual* gehört, bedarf der Unterschrift eines Non-Disclosure Agreements. Es sind jedoch viele Diplomarbeiten und Dissertationen kostenfrei auf www.usixml.org zugänglich, die sich mit UsiXML beschäftigen.

Da sowohl Teile von UsiXML als auch Teile des CTT-Ansatzes zur Standardisierung bei der W3C¹ eingereicht worden sind, finden sich auf dieser Web-Seite auch einige Dokumentationen zu diesen Ansätzen. Diese beschäftigen sich hauptsächlich mit den Metamodellen in UsiXML und dem Task-Modell in CTTE. [CMP⁺] ist ein Beispiel für solch eine Dokumentation. Diese umfasst jedoch auch zahlreiche andere Ideen und Ansätze aus anderen Forschungsschwerpunkten.

UCP bietet industriellen und akademischen Partnern eine kostenlose Lizenz an. Die Dokumentation des Ansatzes befindet sich zwar in ihren Anfängen, jedoch bietet UCP eine genaue Dokumentation der Modelle und Methoden². Wie auch bei UsiXML sind einige Abschlussarbeiten, die sich mit UCP und den eingesetzten Technologien beschäftigen, frei verfügbar. Außerdem wurde für die Werkzeuge eine Eclipse-Update Site eingerichtet. Über diese Seite³ können die Plugins direkt in der Entwicklungsumgebung eingebaut werden.

Unterstützte Anwendungen

Bei MARIA ist dieser Aspekt klar festgelegt. MARIA beschäftigt sich mit Geschäftsanwendungen, die in Form von Webservices abrufbar sind. Eine wesentliche Schwachstelle von MARIA ist, dass die Maschinenentscheidungen, die basierend auf den Ergebnissen dieser Webservices getroffen werden müssen, auf der Tasks-Ebene zwar vollständig modelliert werden können, jedoch durch die Werkzeuge, die MARIAE anbietet, nicht vollständig transformiert werden. Das bedeutet im Konkreten, dass das AUI für Anwendungen, die je nach dem Vorkommen von bestimmten

¹www.w3c.org

²<http://ucp.tuwien.ac.at>

³<http://ucp.ict.tuwien.ac.at/eclipse-update/>

Vorbedingungen anders ablaufen, also Verzweigungen enthalten, noch fehlerbehaftet ist. Dieser Fehler bezieht sich im Wesentlichen auf fehlende Referenzen auf das Datenmodell im AUI. Die Eigenschaft, mit der MARIA bezüglich dieses Kriteriums, verglichen mit anderen Ansätzen einen Vorsprung vorweisen kann, ist die Unterstützung von kooperativen Tasks (*Collaborative Tasks*) [Pat03]. Hierbei können mehrere Personen, gleichzeitig an der Ausführung von Aufgaben zusammenarbeiten. MARIA ist derzeit der einzige Ansatz, welcher eine Unterstützung solcher Tasks anbietet. Wie diese Tasks im CTT modelliert werden können, ist in 3.1 dargestellt.

Abgesehen von diesem Aspekt muss man festhalten, dass MARIAE zur Generierung von GUIs für einfache *Frage- und Antwort-Szenarien* sehr gut geeignet ist. Die Werkzeuge, die mit MARIAE geliefert werden, weisen verglichen mit anderen Ansätzen, einen Vorteil bezüglich Funktionalität auf. Beispiele für besondere Funktionalitäten in MARIAE sind z.B. ein graphischer Editor für Transformationen und einfachere Werkzeuge zum starten eines Transformationsprozesses.

Für *UsiXML* kann diesbezüglich keine ausführliche Erklärung gemacht werden, da die entsprechenden Mittel zur Generierung von GUIs fehlten. Es standen keine Werkzeuge zur Verfügung, um eine Aussage über die Tool-Unterstützung zu machen, und in der vorliegenden Dokumentation wurde ebenfalls dieses Aspekt nicht angesprochen. Bei dem Versuch, eine GUI aus den UsiXML-Modellen zu generieren, wurde die Geschäftslogik als eine Standard Java-Anwendung entwickelt und die Anbindung an die Geschäftslogik folgte manuell.

UCP unterstützt all jene Back-End, die zwei Eigenschaften vorweisen. Die erste Eigenschaft ist die des Dialog-Charakters. Dies bedeutet, eine Applikation kann nur dann modelliert werden, wenn mindestens zwei Kommunikationspartner an ihrer Durchführung teilnehmen. Ein Beispiel mit mehreren Kommunikationspartnern wird in [KPR⁺11] vorgestellt, in dem ein Szenario mit mehr als zwei Aktoren modelliert wird. Die zweite Eigenschaft ist, dass die Applikation eine Prozessbasierte Anwendung ist. *Prozess-basiert* umfasst auch *Frage- und Antwort-Szenarien*. Mit Hilfe von semantisch reichhaltigen Relationen, wie z.B. der *IfUntil*- oder der *Elaborate*-Relation, können Prozesse auf einer sehr abstrakten Ebene mit starkem semantischen Inhalt modelliert werden.

Basierend auf dem bereits Erwähnten, kann die Schlussfolgerung gezogen werden, dass MARIA und UCP beide derzeit die gleiche Art an GUIs unterstützen. Hierbei geht es um einfache Prozessbasierte Anwendungen, die im Falle von MARIAE in Form von Webservices vorliegen müssen, während in UCP die Anwendungen, sowohl in Form einer Standard-Anwendung in Java implementiert werden können als auch einen Webservice-Aufruf enthalten können. Diese Tatsache resultiert in einem wesentlichen Unterschied zwischen diesen beiden Ansätzen: Dieser bezieht sich auf die Eigenschaft der „Statefulness“. Hierunter versteht man die Eigenschaft, dass eine Anwendung unabhängig davon, wie oft sie aufgerufen wird, keine Informationen über die Session speichert. Diese Stateless-Eigenschaft kann in dieser Form für Webservices gegeben sein, während dies für Standard-Geschäftsanwendungen nicht gilt.

Ein Vorteil, der sich daraus ergibt, dass UCP seine eigene Laufzeitumgebung mitliefert, über die das GUI mit dem Back-End kommuniziert, ist, dass sie hiermit nicht an einem fixen Protokoll gebunden ist. Diese Eigenschaft ist so zu verstehen, dass die Back-End in MARIA, nachdem sie in Form von Webservices vorliegen müssen, unbedingt auch den Richtlinien folgen müssen, die durch entsprechende Protokolle vorgegeben sind. Diese sind z.B. bei SOAP-Services die XML-Technologie und das HTTP-Protokoll. In dieser Hinsicht ist bei UCP mehr Freiheit in der Entscheidung gegeben, welche Technologien in der Kommunikation eingesetzt werden. Das Back-End kann sowohl eine einfache lokale Anwendung ohne Netzwerkzugriff sein als auch Webservices aufrufen.

6.1.3.2 Modelle

Die basierend auf modellgetriebenen Ansätzen generierten GUIs sind als Produkte der Transformationen zu verstehen. Damit sind die GUIs gleichzeitig das Abbild der Modelle eines Ansatzes und eine hohe Qualität dieser Modelle ist eine Voraussetzung für die Generierung von guten GUIs. Das Eingangsmodell ist eine abstrakte Darstellung der Interaktion, während die Zwischenmodelle dieses Modell verfeinern. Sowohl das Eingangsmodell als auch die Zwischenmodelle basieren auf eigene Metamodellen. In diesem Abschnitt wird untersucht welche Aspekte der MDUID von den unterschiedlichen Ansätzen durch entsprechende Modelle abgedeckt werden.

Wie schon am Anfang dieses Kapitels besprochen, untersuchen wir beim Vergleich der Modelle das Vorhandensein von Modellen, die unterschiedliche Aspekte der MDUID decken. Diese Modelle sind laut unterschiedlichen Studien [VJ99] notwendig, um die Mensch-Maschine Interaktion mit einem hohen Maß an Vollständigkeit zu modellieren. Tabelle 6.2 stellt einen Ausschnitt der Ergebnisse dieser Untersuchung dar.

Tabelle 6.2: Modellierung unterschiedlicher Aspekte in HCI

Kriterien	MARIA	UsiXML	UCP
Task- / Diskurs-Modell	✓ / -	✓ / -	- / ✓
Domain-Modell	-	✓	✓
Struktur-Modell	✓	✓	✓
Verhaltens-Modell	-	-	✓
Plattform-Modell	-	✓	✓
Umgebungs-Modell	-	✓	-
User-Modell	-	✓	-

In der Menge der untersuchten Modelle stellen die Task- und Dialog-Modelle die Gruppe der Eingangsmodelle dar. Diese Modelle definieren gleichzeitig die unterschiedlichen Philosophien in der Modellierung der HCI. Während ein Task-Modell darauf basiert, dass der Designer einen abstrakten Task in einer Reihe von konkreten Tasks zerlegt, modelliert ein Diskurs-Modell den Informationsaustausch zwischen zwei Akteuren.

Die hierarchische Task-Analyse und die auf ihr basierende CTT, zerlegen einen abstrakten Task in kleinere Teilaufgaben und ergeben als Resultat eine Baum-Struktur. Einer der wesentlichen Vorteile hierbei ist die systematische Herangehensweise an eine Problemstellung und die intuitive Transformation der Tasks in Unteraufgaben. Sollten am Ende eines CTT-Baumes konkrete Tasks stehen, die nicht weiter zerlegt werden können, ist es dem Designer bzw. Entwickler frei überlassen, wie tief die Verschachtelung erfolgt.

Eine Baum-Struktur, wie sie nach der Hierarchical Task Analysis vorliegt, berücksichtigt jedoch längst nicht alle Zusammenhänge zwischen den unterschiedlichen Unteraufgaben. Ein Beispiel für so einen Zusammenhang sind die Auswirkungen bzw. Einflüsse, die die Ausführung einer Aufgabe auf die nächste, folgende Aufgabe hat. Ein wesentlicher Vorteil, den UCP mit seinem Diskurs-basierten Eingangsmodell mit sich bringt, ist, dass auf der semantischen Ebene stärkere bzw. abstraktere Relationen zur Modellierung von Maschinen-basierten Entscheidungen vorliegen. Während in MARIA bzw. CTT nur binäre Operatoren vorliegen, die damit jeweils nur zwei Tasks verbinden können, definiert UCP zusätzlich Relationen die mehrere Zweige miteinander in Beziehung setzen. Diese Relationen werden als *Multi-NUCLEUS Relations* bezeichnet.

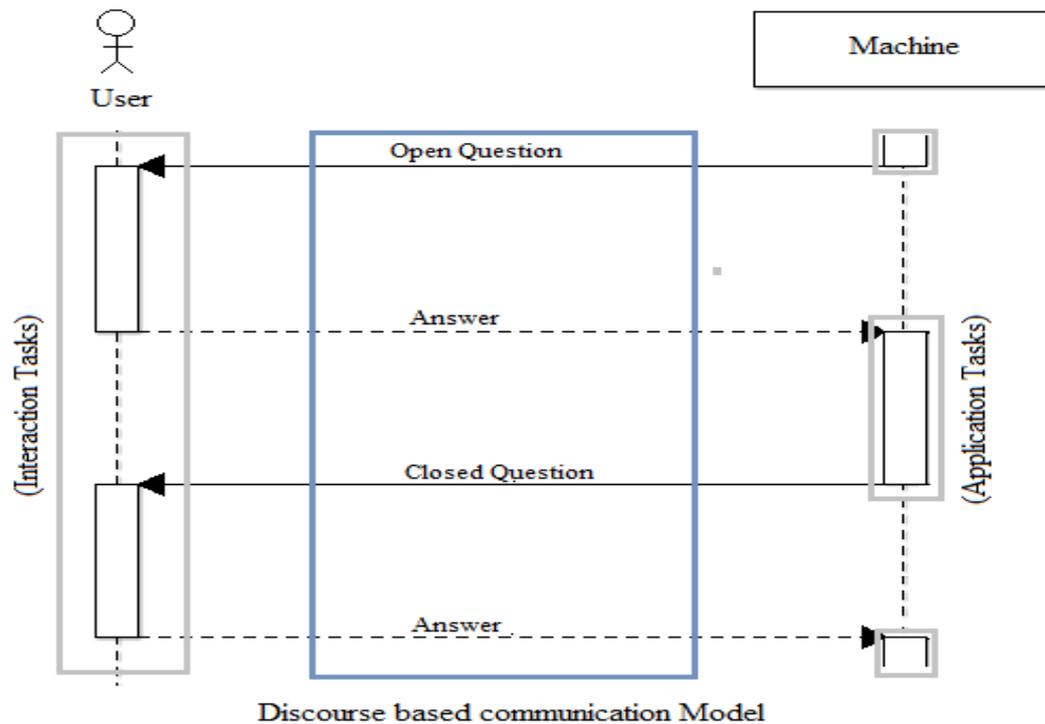


Abbildung 6.4: Gegenüberstellung der Diskurs-basierten Kommunikation und der Task-Modellierung

In Abbildung 6.4 sind UCP:CM und Task-Modell visuell und mit Hilfe einer anderen Notation verglichen. Hier stellt ein UML-Sequenzdiagramm deren Unterschied eindeutig dar. Während UCP:CM die Interaktion und den Informationsaustausch zwischen zwei Partnern modelliert, modelliert ein CTT-Modell bzw. allgemein ein Task-Modell die Aufgaben, die die Kommunikationspartner durchführen müssen, um ihr Ziel zu erreichen.

Es soll hier jedoch erwähnt werden, dass beide Ansätze bezüglich Skalierbarkeit bei der Modellierung von größeren Projekten an ihre Grenzen stoßen und die Überschaubarkeit der Modelle nicht mehr gewährleistet werden kann. Eine Möglichkeit, um die von CTT Task-Modellen bekannte Notation zu erweitern und damit die Skalierbarkeit dieser Modelle zu verbessern, wird in [MPW11] präsentiert. Dabei werden zwei Erweiterungen für das CTT Task-Modell vorgeschlagen: die *Composition* und die *Refinement/Abstraction* Notationen. Refinement/Abstraction erlauben es größere CTT-Modelle in kleinere zu zerteilen und diese miteinander zu verbinden. Die Compositions ermöglichen eine Modellierung von Informationsaustausch zwischen einzelnen CTT-Modellen.

Das Domain Model in MARIA und UsiXML repräsentiert, wie schon in den vorherigen Kapiteln erwähnt, die Objekte des UI, die in einer Interaktion mit der Benutzungsschnittstelle manipuliert werden. In UCP gibt es ein Domain-of-Discourse Model. Es unterscheidet sich hiermit vom Domain Model in den beiden anderen Ansätzen. Während UCP und UsiXML diesen Aspekt in expliziten Modellen behandeln, stellt MARIA nur ab der AUI-Ebene ein Datenmodell dar, welches im Zuge der Transformation des Task-Modelles generiert wird. Dieses Modell orientiert sich hauptsächlich an den WSDL- und WADL-Dateien jener Webservices, für die ein GUI generiert werden soll.

Das Struktur-Modell hat die Aufgabe, den strukturellen Aufbau einer Präsentation im GUI zu

definieren. Dies erfolgt oft mit Hilfe einer Template-Datei, die die Definition einer generisch anwendbarer Struktur erlaubt. Ein strukturelles Modell ist in allen untersuchten Ansätzen explizit definiert. Während die Generierung dieses Modelles in UCP und MARIA auf vordefinierten Transformationsregeln basiert, kann in UCP ein Optimierungsalgorithmus eingesetzt werden. Hierbei wird es dem Anwender ermöglicht, das strukturelle Modell basierend auf gewissen Kriterien zu optimieren. MARIAE definiert zwar keine Optimierungsverfahren, setzt jedoch Heuristiken ein, um die generierten GUI-Strukturen, falls notwendig anzupassen.

Ein Verhaltens-Modell definiert, wie die Interaktion zwischen Maschine und Mensch stattfinden soll. MARIA und UsiXML definieren keine expliziten Verhaltensmodelle, sondern leiten das Verhalten des Systems nur aus der Ebene der Concepts and Tasks ab und bauen dieses Verhalten in ihre Endanwendung ein. UCP hingegen definiert für jeden Diskurs explizit ein Verhaltens-Modell (WIMP UI Behavior Model). Dieses Verhaltens-Modell wird durch eine Zustandsmaschine repräsentiert, welche alle möglichen Zustände und Zustandsübergänge eines GUIs umfasst (Partitioning State Machine). Für die Modellierung des Verhaltens wurde ein eigenes Metamodell für die Partitioning State Machine definiert.

Ein Plattform-Modell spezifiziert die Eingabe- und Ausgabemöglichkeiten, die bei einem Endgerät vorhanden sind. Diese Eigenschaften können dann beim Transformationsprozess mitberücksichtigt werden, um eine passendere Wahl der Interaktionsobjekte zu treffen und ein an die Plattform angepasstes GUI zu generieren. In UsiXML ist das Plattform-Modell ein Teil des Context-Modells. UCP definiert die Application Tailored Device Specification, der die gleiche Rolle zukommt wie einem Plattform-Modell in UsiXML.

Das Umgebungs-Modell soll es ermöglichen, dem Anwender eines Systems abhängig von der Umgebung, in der das System zum Einsatz kommt, unterschiedliche GUIs zur Verfügung zu stellen. UsiXML ist der einzige Ansatz, der so ein Modell definiert. Dieses Modell ist in UsiXML Teil des Context-Models.

Das User-Modell hat bis jetzt nicht wirklich das Interesse der MDUID-Community wecken können. Dies hat unterschiedliche Gründe. Ein Grund dafür, dass das User-Modell nicht viel Beachtung findet, ist die Komplexität der Modellierung dieses Aspektes. Ein Beispiel für solch einen komplexen Fall wäre z.B. die Frage, ob man vom Alter eines Anwenders auf dessen Geschicklichkeit bzw. seine Technologiekenntnisse rückschließen kann. Diese Frage ist nicht selbstverständlich zu beantworten, da mit der Zeit immer mehr Menschen in jüngeren Jahren mit der digitalen Technologie in Berührung kommen und dadurch auch immer mehr ältere Leute ein breites technologisches Geschick vorweisen können. Das Thema der User-Modellierung wird in einer eigenen Community erforscht.

6.1.3.3 Werkzeuge

Für jeden Ansatz werden Werkzeuge und Entwicklungsumgebungen angeboten, die von den jeweiligen Forschungseinrichtungen weiterentwickelt werden. In dieser Arbeit werden die Ansätze hauptsächlich darauf geprüft, ob sie die wichtigsten Werkzeuge für das Design Time Process in CRF anbieten und welche Eigenschaften diese Werkzeuge vorweisen. Die Tabelle 6.3 zeigt eine Zusammenfassung der hier zu besprechenden Kriterien.

Tabelle 6.3: Unterstützende Werkzeuge

Werkzeuge	MARIA	UsiXML	UCP
Development Path	Forward: ✓	Forward: -	Forward: ✓
	Reverse: ✓	Reverse: -	Reverse: -
	Iterative: -	Iterative: -	Iterative: ✓
Editoren für Modelle	✓	✓	✓
Transformation-Unterstützung	✓	-	✓
Verifikation / Validierung	✓ / ✓	- / -	✓ / -
Anpassung	✓	-	~
Integration	✓	-	✓

Development Path

Obwohl aus den Abbildungen 6.1 bis 6.3 die Development Paths zum Teil abgelesen werden können, gilt dies nicht für die benötigte Tool-Unterstützung. Untersucht wurde die Tool-Unterstützung für die drei Development Paths des Forward Engineering, Reverse Engineering und die iterative Herangehensweise.

MARIA bietet mit *MARIAE* eine Unterstützung für das Forward Engineering, wobei der Entwicklungsprozess nicht unbedingt ab der Ebene der Concepts und Tasks gestartet werden muss. Die Entwicklung kann ebenfalls auf den Ebenen des AUI und CUI gestartet werden. Gleichzeitig unterstützt *MARIA* das Reverse Engineering. Dabei kann mit Hilfe von Werkzeugen, die im Zuge des Migration Frameworks [PZ10] implementiert wurden, aus bereits bestehenden GUI-Applikationen das entsprechende AUI und CUI generiert werden. *CTTE* ermöglicht es auch, aus der WSDL-Datei eines Webservices einen CTT-Baum zu generieren. Eine iterative Vorgehensweise wird in *MARIA* nicht unterstützt. Dazu fehlen die entsprechenden Persistierungs-Mechanismen, die eine iterative Entwicklung ermöglichen. Diese Mechanismen würden es erlauben, nur bestimmte Teile der Modelle bzw. Artefakte zu ändern und den Rest unverändert zu regenerieren.

Da für *UsiXML 2.0* nur Editoren für die verschiedenen Modelle vorliegen, könnte hier für die Development Paths keine Tool-Unterstützung untersucht werden.

UCP unterstützt neben Forward Engineering für die Generierung von GUIs, in dem das UCP:UI eine zentrale Rolle spielt, auch die iterative Entwicklung von GUIs. Hiermit ermöglicht *UCP*, bei erneuter Generierung von GUIs, die Persistenz von manuellen Änderungen in den Eingangsmodellen, Transformationsregeln und CSS Style Sheets. Die entsprechenden Technologien dazu basieren auf der Eclipse-Umgebung.

Unterstützung der Modelle und Transformationen

MARIA bietet für alle Ebenen des CRFs ein Tool an. Für die Concepts and Tasks-Ebene liegt ein expliziter Editor namens *CTTE* vor. Dieser unterscheidet sich von *MARIAE*, obwohl sich die in diesen Umgebungen implementierten Funktionalitäten teilweise überschneiden. In *MARIAE* kann das Task-Modell nur dargestellt, jedoch nicht bearbeitet werden. Die AUI und CUI können in *MARIAE* in einem integrierten Editor dargestellt und modifiziert werden. Zu diesem Zweck liegen sowohl ein textbasierter als auch ein graphischer Editor vor.

MARIAE unterstützt zur Generierung von FUIs folgende Modelle:

- HTML, JSP für GUIs
- VoiceXML für Vokale Benutzungsschnittstellen
- XHTML und Voice für Multimodale Anwendungen

All diese Technologien werden von MARIAE durch entsprechende Reifications unterstützt. Im Zuge dieser Arbeit wurden die in MARIAE erzeugten JSP GUIs untersucht. Diese enthalten im generierten Code Fehler, wobei die Wurzel dieser Fehler im AUI liegen und auf falsche Transformationsregeln zurückzuführen sind.

Eines der interessantesten Werkzeuge, welches mit MARIAE mitgeliefert wird, ist ein Werkzeug zur Unterstützung selbstdefinierter Transformationen, in dem der Anwender in einem grafischen Editor die Transformation von Objekten aus einem Source Model auf Objekte aus einem Target Model spezifizieren kann. MARIA ermöglicht es dem Entwickler, selbstdefinierte XSL-Transformationen anzuwenden und bietet dazu einen integrierten Editor.

Das „Migration Framework“ [PSS09] ermöglicht es, verteilte Szenarien zu realisieren. Dieses Framework erlaubt es den Anwendern über unterschiedliche End-Geräte ein bestimmtes Ziel zu erreichen, indem sie die notwendigen Tasks ausführen. In diesem Framework müssen sich im ersten Schritt all jene Geräte, die an der Zusammenarbeit interessiert sind, bei einem zentralen Server registrieren. Der Anwender, der über ein bestimmtes Endgerät eine Interaktion initiiert, kann dann zu einem späteren Zeitpunkt einen Antrag stellen, dass er das weitere Vorgehen über ein anderes Endgerät abhandeln will. Basierend auf dieser Anfrage erstellt das Framework das entsprechende UI für das neue Endgerät und überträgt dieses, inklusive des aktuellen Zustandes des Systems auf dem ersten Endgerät, an dem neuen Interaktionsgerät. Hier kann der Anwender dort weiterarbeiten (in jenem Zustand), wo er die Interaktion auf dem letzten Gerät abgebrochen hat.

Zur *UsiXML 2.0* und den benötigten Werkzeugen liegen nicht viele Informationen vor. Die einzigen Werkzeuge, die für diese Version bekannt sind, sind die Editoren, die in Form von Eclipse Plugins vorliegen. Mit Hilfe dieser Editoren können *Task-*, *Domain-*, *Context-* und *AUI-*Modelle erstellt werden. Diese Modelle sind im Abschnitt 4.1 ausführlich vorgestellt. UsiXML 2.0 bietet bezüglich Endtechnologien keine explizite Unterstützung an. Für die Transformationen und die Generierung des FUIs werden Werkzeuge benötigt, die nicht verfügbar sind. Dass jedoch aus einem CUI in *UsiXML 2.0*, ein vollständiges FUI generiert werden kann, wurde im Zuge dieser Arbeit untersucht und als möglich angesehen. Das Metamodell bietet dafür alle benötigten Informationen. Für eine Auflistung der in Publikationen vorgestellten Werkzeuge für *UsiXML 1.0* sei hier auf [Van08] hingewiesen.

UCP bietet für alle vorhandenen Modelle Werkzeuge zur Unterstützung des Entwicklers. Diese basieren zum Großteil auf dem EMF und auf der Eclipse Rich-Client Plattform. Diese Werkzeuge umfassen Editoren für DoD Model, ANM und Discoure Model. Zur Ausführung der Transformation von UCP:CM zum AUI und CUI wird auch ein Eclipse Plugin eingesetzt. Die Gesamtheit der vorhandenen Werkzeuge für Transformationen wird Generierungs-Framework (UCP:UI) genannt und ist in 5.2 vorgestellt. Das WIMP UI Behavioral Model und das Structural Model, die durch UCP:UI generiert werden, können ebenfalls in Eclipse-Editoren gelesen und bearbeitet werden. Die Modelle können sowohl mit einem graphischen Editor als auch in einem Text-Editor bearbeitet werden. Die Werkzeuge bieten die entsprechende Umgebung an [PRK13].

UCP unterstützt die Generierung von HTML-Seiten, die das Verhalten und den Aufbau des CUIs, bestehend aus dem WIMP UI Behavioural Model und dem Structural UI Model, implementieren.

UCP bietet zu diesem Zweck ein eigenes Generierungs-Framework an. In dieser Umgebung kann der Anwender alle relevanten Konfigurationen eingeben. Zu diesen Konfigurationen gehören die Adressen der zu erzeugten Modelle, die Quell-Modelle, die Art und Weise, wie das generierte GUI an eine Plattform angepasst werden soll, und welchen Randbedingungen diese Plattform unterliegt. Diese Konfigurationen und die entsprechenden GUIs sind in 5.2 vorgestellt.

Werkzeuge zur Validierung und Verifikation

Werkzeuge zur Validierung unterstützen den Anwender bei der Überprüfung, ob man das Richtige modelliert hat. In Zusammenhang mit den verglichenen Ansätzen gibt es ein Simulation Framework. Dieses ermöglicht es dem Anwender, die Anwendung auf der Task-Ebene zu simulieren und zu überprüfen, ob die richtigen Aufgaben sowohl angeboten als auch ausgeführt werden.

Im Gegensatz zur Validierung prüft die Verifikation, ob ein Modell bezüglich eines anderen Modells richtig ist. Ein Beispiel für so eine Überprüfung ist, ob alle notwendigen Felder für einen Knoten spezifiziert sind.

CTTE stellt sowohl zur Validierung als auch zur Verifikation Werkzeuge zur Verfügung. In der Simulationsumgebung kann der Anwender verschiedene Szenarien simulieren. Hierbei können auch die definierten Vorbedingungen mitberücksichtigt werden [Pat02]. Mit Hilfe der Verifikations-Werkzeugen können die Task-Modelle auf ihre Übereinstimmung mit dem Metamodell geprüft werden [MPS02].

UsiXML und UCP bieten keine Validierungs-Werkzeuge an. In UCP werden Modellintegritätsbedingungen definiert, welchen die Modelle in UCP:UI genügen müssen, damit aus diesen Modellen GUIs generiert werden können [Sch10]. Die Integritäts-prüfung in UCP kann zu der Kategorie der „Metamodell Compliance“-Werkzeuge gezählt werden. Das bedeutet, es untersucht die Übereinstimmung eines Modelles mit den Definitionen in einem Metamodell. Die zur Realisierung dieses Werkzeugs eingesetzte Technologie ist das „Eclipse Validation Framework“⁴.

Anpassungsmöglichkeiten

MARIA bietet viele Möglichkeiten zur Customization des GUIs. Sowohl auf der AUI-Ebene als auch auf der CUI-Ebene können mithilfe des entsprechenden Editors Änderungen an dem GUI vorgenommen werden. Am abstrakten GUI können außerdem die generierten abstrakten Scripts auf ihre Parameter verifiziert werden und es kann die Gruppierung der abstrakten Elemente überprüft und angepasst werden. Auf der CUI-Ebene bietet MARIAE, zur Unterstützung des Entwicklers einen integrierten Browser, der die konkreten Presentation Tasks darstellt. Ein Nachteil von MARIAE ist, dass diese Anpassungen nicht für etwaige Iterationen persistiert werden können.

UsiXML 2.0 bietet auch hier vielversprechende Möglichkeiten in seinem Metamodell. Da jedoch die entsprechenden Werkzeuge nicht vorhanden sind, gestaltet es sich als schwierig, diese Möglichkeiten zu evaluieren.

UCP ermöglicht es, die zu generierende Benutzungsschnittstelle, welche unter anderem als ein Structural Model vorliegt, anzupassen. Außerdem kann der Entwickler viele der Komponenten frei modifizieren.

⁴Obwohl die Technologie in Eclipse als eine Validierungs-Plattform bezeichnet wird, handelt es sich um eine Verifikation.

Integration des UI und des Back-End

In den meisten Arbeiten zum Thema MDUID wird der Aspekt der Anbindung der generierten UIs zur Geschäftslogik vernachlässigt. Falls keine Maßnahmen zur Anbindung des UIs an die Geschäftslogik gegeben sind, bedeutet das eine sehr zeitaufwendige Entwicklungsphase, in der diesem Aspekt manuell nachgekommen werden muss.

Da in MARIA die Geschäftsanwendung ein integrierter Bestandteil des Datenmodells ist, müssen die Application Tasks mit den entsprechenden Webservices verbunden werden. Basierend auf dieser Verknüpfung generiert MARIAE automatisch die den Webservices entsprechenden Clients. Diese können im Weiteren auf das Webservice zugreifen und die Kommunikation abwickeln.

MARIAE ermöglicht es, mit der Hilfe eines eingebauten Editors die unter einer URL zu erreichende WSDL-Datei in der Entwicklungsumgebung zu laden. Diese Dateien beschreiben in einer XML-Sprache den Aufbau des Back-Ends. Damit die Application Tasks diese aufrufen, reicht es bei der Entwurfsphase die entsprechenden Tasks mit diesen Webservices zu verbinden. Es ist keine weitere Entwicklungsarbeit notwendig. Der in MARIAE generierte Code besteht aus zwei Code-Paketen bzw. Komponenten. Das erste Paket enthält die Kern-Module, die unter anderem auch das Datenmodell umfassen. Das zweite Paket enthält die Datenstrukturen, die zur Kommunikation mit den Webservices benötigt werden. Es wird keine explizite Schnittstelle zwischen dem GUI und der Geschäftslogik definiert, jedoch kann der Entwickler in das Datenmodell eingreifen und falls notwendig dieses ändern.

In *MARIAE* kann ausgehend von jeder Ebene des CRF ein Prototyp erstellt werden. Sobald das Webservice implementiert ist, kann wie schon bereits erwähnt das entsprechende Task-Modell zur Ein-/Ausgabe der entsprechenden Daten generiert werden. Alle weitere Transformationen werden mitgeliefert und sind sofort einsetzbar. Dementsprechend ist das Erstellen eines ersten Prototyps mit einem relativ geringen Aufwand verbunden.

UsiXML 2.0 bietet keinen expliziten Ansatz für die Integration der Geschäftsanwendung. Dadurch ergibt sich für diesen Ansatz ein Nachteil, da genau die Unterstützung dieser Eigenschaft einen wesentlichen Beitrag dazu leistet, die MDUID in dem Software-Entwicklungsprozess einzubauen.

UCP stellt keine expliziten Werkzeuge zur Integration der Geschäftsanwendung bereit. Die Kommunikation mit dieser Geschäftsanwendung bildet jedoch einen wesentlichen Bestandteil des Ansatzes. Die Architektur der generierten Anwendung entspricht dem MVC Pattern, wobei UPC:RT die Rolle des Controllers übernimmt und die Kommunikation zwischen GUI und Applikationslogik regelt. [Kai13] Der Inhalt der Anwendung wird teilweise schon bei der Modellierung des DoD und des ANMs spezifiziert, um danach in der Applikationslogik implementiert zu werden. Die Verbindung der GUI zum Back-End erfolgt über den *Application Adapter*. Das Gerüst zu dieser Klasse wird durch die Tool-Unterstützung automatisch generiert, muss jedoch durch den Entwickler vervollständigt werden [Š14]. Die Applikationslogik, auf der von dem Application Adapter aus zugegriffen wird, muss manuell implementiert werden.

6.1.3.4 Das generierte GUI

In diesem Abschnitt sollen Eigenschaften der generierten GUIs verglichen werden, die in der Tabelle 6.4 dargestellt sind.

Tabelle 6.4: Das generierte GUI

Das generierte GUI	MARIA	UsiXML	UCP
Fehlerfreiheit der Transformationen	~	-	~
Hilfe-Funktion	✓	✓	-

Fehlerfreiheit der Transformationen

Unter „Fehlerfreiheit der Transformationen“ soll hier die Fehlerfreiheit der Transformations-Logik verstanden werden, die zur Generierung des Quellcodes bzw. der GUI-Quellcode beiträgt. In diesem Abschnitt werden die vorgefundenen Fehler, soweit welche vorhanden sind, aufgelistet.

Im Zuge dieser Arbeit wurden die Code-Generatoren der MARIAE-Releases 1.5.4 bis 1.5.6 angewendet. Während in den ersten Versionen einige Fehler im generierten Code zu finden waren, sind diese in Version 1.5.6 behoben.

Das im *UCP* generierte GUI enthält im generierten Code keinen Fehler. In der ersten Version des UCP-Toolings, mit welches im Zuge dieser Arbeit experimentiert wurde, war ein Fehler in den Transformationsregeln des DoD Models enthalten. Dieser Fehler führte für Attribute des Typs Boolean zu Inkonsistenzen zwischen der Spezifikation und der GUI. Dieser Fehler wurde in der aktuellen Version des UCP-Werkzeugs behoben.

Hilfe-Funktionen des generierten GUIs

Hilfe-Funktionen spielen im Bereich der GUIs aus dem Grund eine wichtige Rolle, da die Anwender eines interaktiven Systems unterschiedliches Hintergrundwissen haben. Es trägt einen wesentlichen Beitrag zu einer benutzerfreundlichen Interaktion bei, wenn in Fällen, wo die Anwender Tipps und Hilfsanweisungen brauchen, ihnen diese auch angeboten werden.

MARIAE kommt diesem Aspekt mit Hilfe der sogenannten Annotation Files entgegen. Wie schon in dieser Arbeit erwähnt, können die Entwickler der Webservices mit Hilfe der Annotations zu den unterschiedlichen Parametern eines Webservices zusätzliche Informationen anbieten. Im Zuge des ServFace Projekts⁵ wurde zu diesem Zweck ein eigener Workflow definiert. Hierbei wird dem Entwickler zur Bearbeitung der WSDL Files, welche die unterschiedlichen Webservices beschreiben, eine eigene Umgebung zur Verfügung gestellt. In dieser Entwicklungsumgebung kann der Entwickler mit Hilfe der URL einer WSDL-Datei diese laden und über eine graphische Oberfläche den unterschiedlichen Feldern des Webservices zusätzliche Informationen zuordnen. Diese modifizierte WSDL-Datei wird danach in einem eigens dafür vorgesehenen Repository abgelegt und kann von *MARIAE* abgefragt und geladen werden. Aus den Rendering-Informationen können auch Hilfe-Funktionen generiert werden. Diese können z.B. in Form einer Warnmeldung auftreten. Es folgt ein Beispiel für die Definition einer solchen Hilfe-Funktion in *MARIAE*. Wie im folgenden XML-Ausschnitt zu sehen ist, wird hier für das Objekt „login“ (*Zeile 1*) in der *Zeile 2* ein Feedback-Text mit dem Inhalt „Feedback“ generiert. In der *Zeile 4* wird eine Fehlermeldung definiert, die angezeigt wird, wenn für das Objekt Login keine Eingaben vorliegen.

⁵<http://www.servface.eu/>

```

1 <referenceObjects hierarchicalName="login" refType="DataTypeElement">
2   <annotations xsi:type="SAMA:TextFeedback" type="Help" languages="//@languages
   .0" platforms="//@platforms.0" text="Feedback" />
3   <annotations xsi:type="SAMA:MandatoryField" >
4     <errorMessage xsi:type="SAMA:TextFeedback" text="Hilfe-Text" type="Error"
       languages="//@languages.0" platforms="//@platforms.2" />
5   </annotations>
6 </referenceObjects>

```

UsiXML spricht dieses Thema in der Publikation [GF10] an. Es werden basierend auf Traceability-Mechanismen, die den aktuellen Zustand des System auf den entsprechenden Knoten im Task-Modell zurückführen, Hilfe-Funktionen generiert, die vom Anwender abgerufen werden können. Diese Informationen werden in Form von Fragen und Antworten präsentiert. Bei einer solchen Vorgehensweise werden die Informationen, die in der Entwurfsphase durch ein Task-Modell ausgedrückt worden sind, in der Laufzeitumgebung miteinbezogen und betrachtet. Dadurch können mit Hilfe der Informationen, die in den generierten GUIs enthalten sind und jenen aus dem Task-Modell, Rückschlüsse auf den Informationsfluss gemacht werden, so dass dem Anwender gezeigt werden kann, mit welchem Task er sich gerade beschäftigt. Aufbauend auf dieser Information können Hilfsinformationen zu den Tasks sowohl im vorherigen Zustand als auch im nächsten Zustand des GUIs angezeigt werden. Ein entsprechendes Screencast ist unter <http://www.youtube.com/watch?v=Ww9yXT4e050> zu finden.

UCP bietet derzeit noch keine Konzepte zur Bereitstellung von Hilfe-Funktionen. Deren Generierung wäre jedoch, basierend auf zusätzlichen Informationen, die man im Discourse Model z.B. mit Hilfe von *Informing* explizit modellieren kann, möglich.

6.2 Praktischer Vergleich

Der Vergleich der erwähnten Ansätze basiert neben dem konzeptionellen Vergleich auf Erfahrungen, die bei der Implementierung von Beispielanwendungen gemacht wurden. Die erste Beispielanwendung ist das Running Example, welches aus einem einfachen Login Use Case besteht. Hierbei soll sich der Anwender in einem „Login“-Szenario bei einem System anmelden und im Falle eines erfolgreichen Logins für ihn bestimmte Daten angezeigt bekommen. Schlägt der Login-Versuch fehl, soll die Anzeige in einem anderen Zustand wechseln. In diesem Zustand soll dem Anwender eine Fehlermeldung angezeigt werden, in der der Anwender aufgefordert wird, sich noch einmal anzumelden.

Bei dem zweiten Beispiel handelt es sich um eine Anwendung zur Reservierung von Mietwagen. Das entsprechende Szenario ist wie folgt definiert:

1. Der Anwender sucht sich den Ort des Verleihs und der Abgabe des Mietwagens aus.
2. Der Anwender sucht sich die Zeitspanne aus, für die er das Auto mieten möchte.
3. Der Anwender entscheidet sich für einen Wagen aus einer Liste verfügbarer Autos.
4. Der Anwender kann sich für eine zusätzliche Ausstattung entscheiden.
5. Der Anwender gibt eine Reihe an persönlichen Informationen an.

6. Der Anwender bekommt detaillierte Informationen über die Vermietung eines Wagens angezeigt, bevor er sich endgültig entscheiden kann.

6.2.1 Login

Wie schon erwähnt, wurde das Login-Beispiel als Running Example verwendet. Es hat sich gezeigt, dass dieser Anwendungsfall, obwohl er ein simples und kompaktes Beispiel darstellt, einige wichtige Eigenschaften der Ansätze zum Vorschein gebracht hat. Die meisten Erfahrungen, die mit diesen Eigenschaften zusammenhängen, wurden bereits im Zuge des konzeptionellen Vergleichs wiedergegeben, da das Login-Beispiel neben den praktischen Aspekten auch die Präsentation des theoretischen Aufbaus der Ansätze untermauerte. Genau aus diesem Grund soll ein eigener Abschnitt nur dem praktischen Vergleich der Ansätze gewidmet werden. Die zusätzlichen Erkenntnisse aus einem praktischen Vergleich werden bei der Generierung eines GUIs für die Car Rental-Anwendung erarbeitet. Nachdem für UsiXML die entsprechenden Werkzeuge nicht vorhanden sind, sind bei dem Vergleich im nächsten Abschnitt, nur die zwei anderen Ansätze in Betracht gezogen.

6.2.2 Car Rental

Ein praktischer Vergleich zwischen MARIA und UCP anhand von Car Rental ist Inhalt dieses Abschnittes. Die Car Rental-Anwendung wird im W3C-Standard als ein Use Case eingeführt, um die Möglichkeiten von MDUID-Ansätzen zu evaluieren und zu präsentieren. Das genaue Ziel dieses

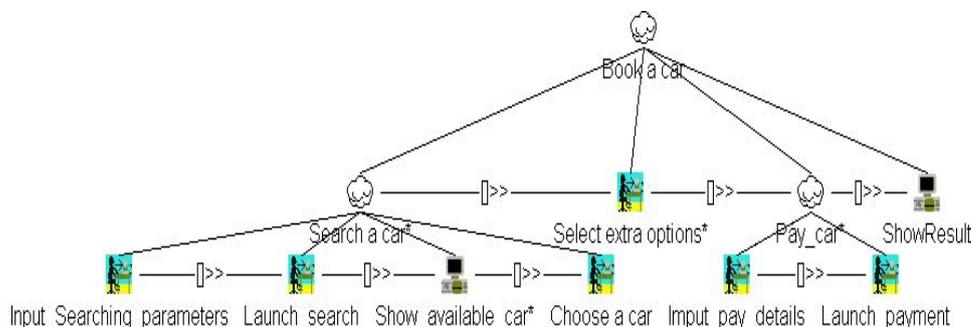


Abbildung 6.5: Ausschnitt des CTT der Car Rental Anwendung (kopiert von der Homepage der uclouvain⁶)

Szenarios und dessen Unterteilung in Unteraufgaben, ist schon im Abschnitt 6.2 präsentiert. Die Abbildung 6.5 stellt einen Ausschnitt aus dem CTT Task-Baum dieser Aufgabenstellung dar. Das vollständige CTT-Modell ist im Anhang zu finden. In diesem Abschnitt ist, ausgehend von einem Reverse Engineering des Task-Baums im Anhang, das entsprechende Kommunikations-Modell in UCP, bestehend aus DoD Model, ANM und Diskurs Model, spezifiziert. Aus diesen Modellen wurde mit UCP:UI ein GUI generiert und die benötigte Applikationslogik manuell in Java implementiert. Das generierte GUI und dessen Verhalten wurde mit der Referenz-Implementierung des Beispiels verglichen. Diese Referenz ist unter <http://sites.uclouvain.be/mbui/index.php?t=carRenting/v3> für verschiedene Plattformen zu finden.

⁶http://sites.uclouvain.be/mbui/trees/carRenting/v3/task_model.png

Mit Hilfe des Task-Baumes in Abbildung 6.5 ist zu erkennen, dass das Modell, keinen iterativen Task darstellt. Der vollständige CTT-Baum kann dem Anhang A dieser Arbeit entnommen werden.

Abbildung 6.6 stellt ein Discourse Model für Car Rental dar, das basierend auf dem Task-Baum in Anhang A modelliert wurde. In diesem Discourse Model wurde als Wurzel-Knote das Procedural Construct *Sequence* (*S1*) eingesetzt. Dieses erzwingt eine sequenzielle Ausführung der einzelnen Zweige und entspricht hiermit dem Informationsfluss und der Prozessdefinition aus dem CTT-Baum am besten. Die Reihenfolge der Ausführung dieser Zweige wird durch die Eigenschaft *Condition* der Zweige festgelegt. Die einzelnen Unterzweige umfassen entweder gleich einen Communicative Act oder verbinden mehrere Communicative Acts mit Hilfe von RST-Relationen.

Im vierten Zweig erscheint die erste RST-Relation, welche die untergeordneten Adjacency Pairs in geordneter Reihenfolge in einer Präsentation darstellt. Diese Relation ist der *OrderedJoint* (*O1*). Sie sorgt hier, wie auch im Zweig fünf dafür, dass die hierarchisch darunterliegenden Communicative Acts in der angegebenen Reihenfolge im GUI abgebildet werden. Die Reihenfolge wird ebenfalls basierend auf der *Condition* Property festgelegt. Hiermit kommt der OrderedJoint Relation eine ähnliche Bedeutung zu wie dem Interleaving-Operator in CTT. Genau betrachtet ist dieser Operator in UCP generischer definiert, da man in UCP nicht nur Elemente gleichzeitig darstellen kann, sondern noch explizit die genaue Reihenfolge im Modell spezifizieren kann. Dies geschieht in CTT implizit, in dem die Reihenfolge der parallelen Tasks dem graphischen Modell entnommen wird. Der Grund hierfür liegt in dem Aufbau der Operatoren in CTT. Während CTT nur binäre Operatoren definiert, mit welchen jeweils nur zwei Tasks verbunden werden können, definiert UCP Multi-NUCLEUS-Relationen, die mehrere NUCLEUS-Zweige miteinander verbinden können. Diese NUCLEUS-Zweige können entweder RST-Relationen oder Procedural Constructs beinhalten.

Im fünften Zweig wird mit Hilfe der *Title* (*T1*) Relation ein NUCLEUS-Zweig dargestellt, welches ein Ordered Joint umfasst und darauf folgend wird der SATELLITE- Zweig dargestellt. Wie man dem Discourse Model entnehmen kann, enthält die Title-Relation in dem SATELLITE-Zweig eine Fragestellung. Die Zustandsmaschine, die hinter einer Title-Relation steckt, sorgt dafür, dass die aktuelle Präsentation basierend auf der Antwort dieser Frage ihren Endzustand erreicht. Konkret handelt es sich hier um die Antwort darauf, ob ein Auto gebucht werden soll oder nicht. Basierend auf der Antwort des Anwenders wird in der Applikationslogik (Abbildung 6.10) eine Variable gesetzt, die im nächsten Schritt abgefragt wird.

Dementsprechend ist im nächsten Schritt die *Condition*-Relation (*C1*) eingesetzt. Wie auch im graphischen Modell dargestellt, prüft diese Condition, ob die Bedingung $OK == true$ erfüllt ist. Die Überprüfung erfolgt durch die Abfrage der oben erwähnte Variable in der Applikationslogik. Der Inhalt dieser Variable entscheidet im nächsten Schritt, ob der *THEN*- oder der *ELSE*-Zweig ausgeführt wird. Wenn die Bedingung erfüllt ist, also der Anwender einer Buchung zustimmt, wird zum THEN-Zweig gewechselt, in dem der Anwender eine Bestätigung seiner Buchung und die Möglichkeit eines Neu-Startes bekommt. Falls der Buchung nicht zugestimmt wird, wird zum ELSE-Zweig gewechselt, wo dem Anwender entsprechendes Feedback angezeigt wird. Zusätzlich bekommt der Anwender auch hier die Möglichkeit eines Neustart der Prozesskette.

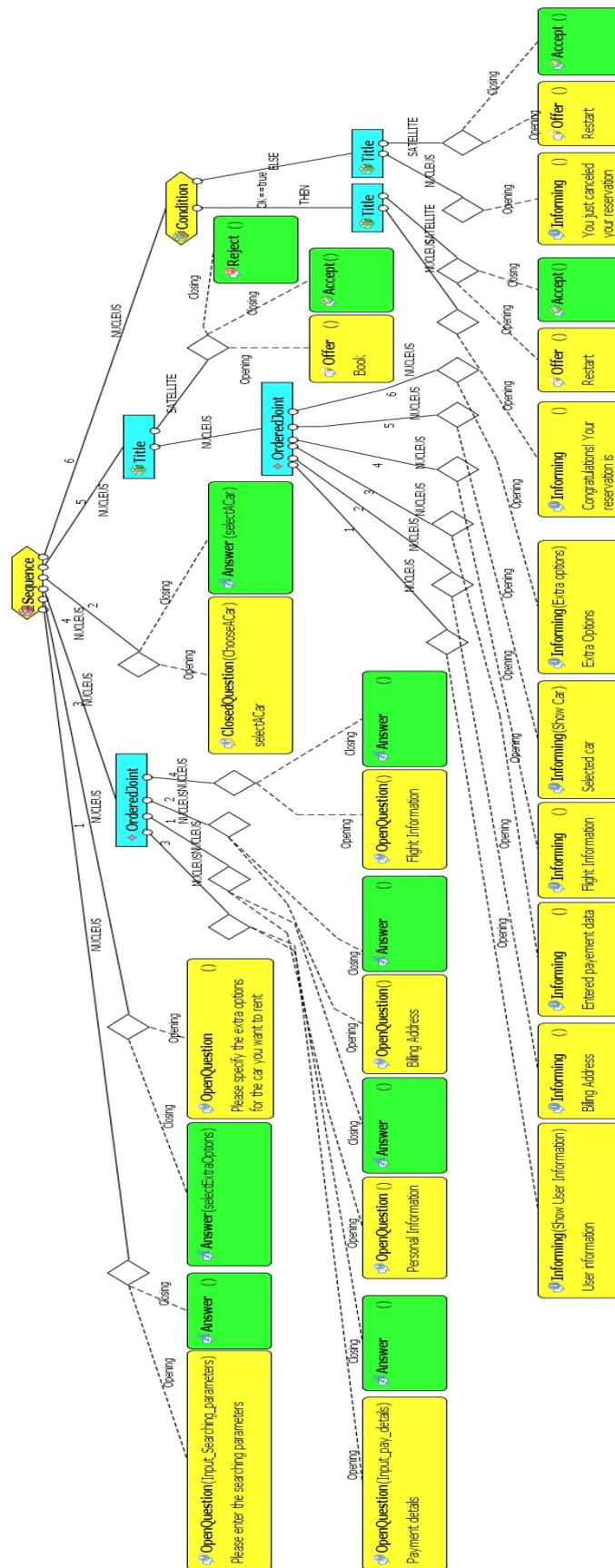


Abbildung 6.6: Discourse Model der Car Rental-Anwendung

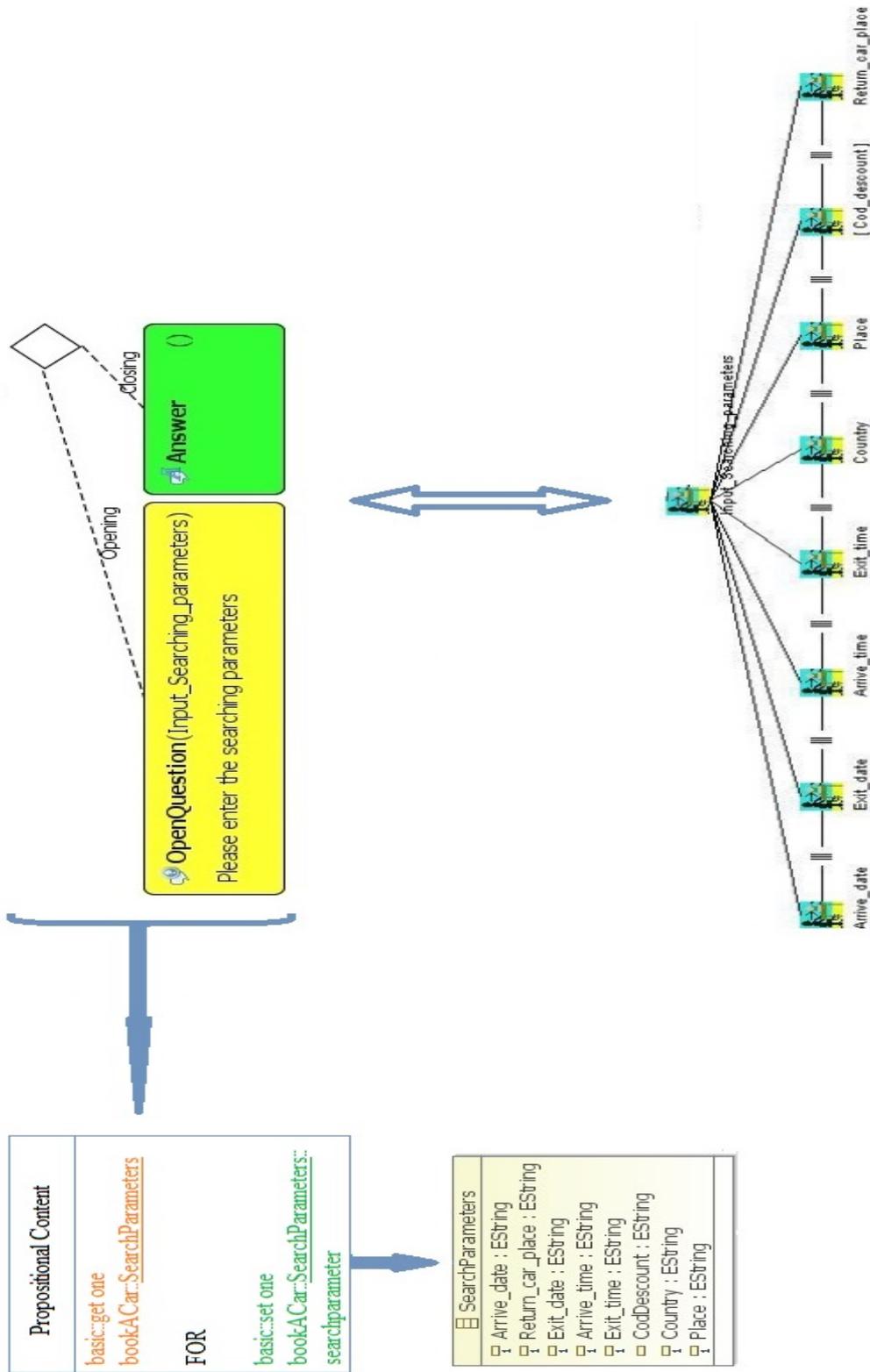


Abbildung 6.7: Vergleich der Granularitäten in Discourse Model und CTT

Was beim Vergleich des Discourse Models mit dem CTT-Modell ersichtlich wird, ist, dass beim CTT-Modell die Anfragen an den Anwender von der Granularität her viel detaillierter sein müssen als im Discourse Model. Dies ist auf die Tatsache zurückzuführen, dass UCP in seinem DoD Model explizit die Objekte definiert, die während einer Interaktion ausgetauscht werden. Dadurch ist es in UCP, im Gegensatz zu CTT-Modellen möglich, diese Objekte abzufragen. In CTT jedoch ist der Entwickler in der Entscheidung wie feingranular das Modell sein soll dadurch eingeschränkt, dass er für jeden Application Task den er mit einem Webservice verbindet, die entsprechenden Methodenparameter vom Anwender abfragen muss. Dieser Nachteil ergibt sich primär nicht aus der CTT-Philosophie, sondern aus dem CTT-Tooling. Um dieses Verhalten genauer zu erläutern stellen Sie sich in einem CTT-Modell ein Application Task vor, welches mit einem Webservice verbunden ist und die Aufgabe hat ein Produkt in einem Warenkorb zu legen. Nachdem im CTT keine Möglichkeit vorliegt ein *Produkt* explizit zu definieren, kann kein interaktiver-Task modelliert werden, der in der Lage ist ein Objekt vom Typ *Produkt* abzufragen. Das bedeutet das hierzu alle Parameter einzeln abgefragt werden müssen. Diese Abfragen finden sich im CTT Task-Modell in Form von einzelnen Interaction Tasks, die die Größe des Modells schnell wachsen lassen. Zusammengefasst bedeutet dies, dass man bei der Modellierung einer Interaktion im Discourse Model bezüglich Granularität des Modells einen größeren Freiraum genießt, während es in einem CTT-Modell diesbezüglich nur eine Möglichkeit gibt. Abbildung 6.7 stellt ein Beispiel für die unterschiedliche Modellierung des gleichen Sachverhaltes in den beiden Ansätzen dar.

Ein zweiter, wesentlicher Unterschied zwischen den beiden Modellen sind die Operatoren oder Relationen, die den Informationsfluss modellieren. MARIA definiert mit den temporalen Operatoren sowohl den Informationsfluss als auch die Struktur der Präsentationen, die in den GUIs eingebaut werden. UCP hingegen trennt die Relationen, die sie in zwei Gruppen unterteilt, nämlich einerseits die Relationen, die die Struktur der GUI Präsentationen modellieren und andererseits jene die den Informationsfluss definieren. So wie schon im Kapitel über UCP erwähnt, bezeichnet UCP die erste Gruppe an Operatoren als RST Relations und die zweite als Procedural Constructs.

Zur Generierung der Applikation und des GUIs werden noch das ANM und das DoD Model benötigt. Diese sind in den Abbildungen 6.8 und 6.9 dargestellt. Im Action-Notification Model sind abgesehen von den in der Basic.anm Datei [Pop12] vordefinierten Actions und Notifications zusätzliche Klassen definiert. Diese umfassen *bookACar*, *exit* und *restart*. Diese werden in der angegebene Reihenfolge zur Buchung eines Autos, zum Abschließen eines Buchungs-Vorganges oder zur Wiederholung einer Buchung verwendet. Die zusätzlich definierten Notifications *Fail* und *Success* werden eingesetzt, um nach einer Buchung bzw. einem Versuch ein Auto zu buchen, Feedback zu geben. Die drei Punkte, die unter den Bezeichnungen eingezeichnet sind (<...>) sind Platzhalter für den Beschreibungstext, der für jede Klasse angegeben werden kann. Auf dieser Beschreibungsmöglichkeit in den Abbildungen wurde hier verzichtet.



Abbildung 6.8: Action-Notification Model der Car Rental-Anwendung

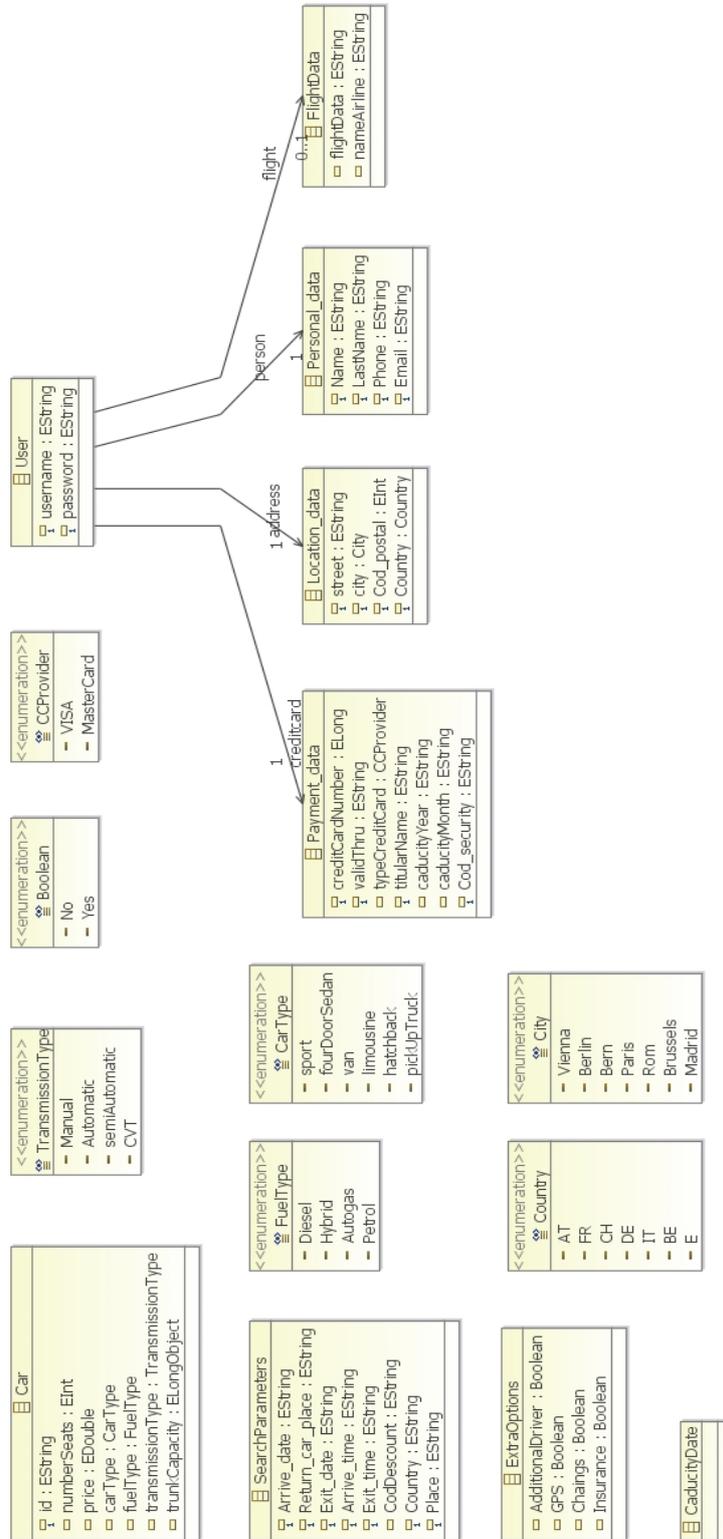


Abbildung 6.9: Domain of Discourse Model der Car Rental-Anwendung

Das ANM ist wegen der Architektur, in der UCP ihre GUIs generiert, unbedingt notwendig, da die im ANM definierten Elemente zur Definition der Propositional Content beitragen. Dieses Modell muss dementsprechend bei jeder Anwendung gemäß der Kommunikation zwischen GUI und Back-End angelegt werden. Ein ähnliches Konzept ist für CTT und MARIA nicht vorgesehen.

Das DoD Model stellt all jene Informations-Objekte dar, die im Kontext der Buchung eines Autos zwischen den Kommunikationspartnern ausgetauscht werden. Die Klasse „Boolean“ wurde in diesem Beispiel explizit modelliert. Der Grund hierfür war, dass versucht wurde, für die Auswahl von extra Ausstattung ähnliche Interaktions-Objekte in dem GUI zur Verfügung zu stellen wie in den Referenzimplementierung.

Abbildung 6.10 stellt einen Ausschnitt aus der Implementierung der Applikationslogik für Car Rental dar. Die roten Punkte markieren die Variablen, während die grüne Markierung die Methoden kennzeichnet. Die Methoden in der Applikationslogik umfassen alle *getter*- und *setter*-Methoden für die definierten Variablen. In der Abbildung ist aus Platzgründen nur ein Ausschnitt der definierten Methoden wiedergegeben.

Abbildung 6.11 zeigt die erzeugten GUI-Zustände. Bei der Generierung des GUIs wurden die Einstellungen des Generierungs-Frameworks so vorgenommen, dass die unterschiedlichen Abfragen und Präsentationen in Form einer Liste angezeigt werden bzw. genug Platz in Anspruch genommen wird, damit so eine Anzeige möglich ist.

Es soll noch das generierte GUI kurz mit den GUI Mock-Ups im Internet⁷ verglichen und Unterschiede angesprochen werden. Die generierte Anwendung entspricht sowohl in ihrer Struktur als auch in ihrem Verhalten zum Großteil der im Internet vorgegebenen Implementierung. Der größte Unterschied in der Struktur bezieht sich auf die Anzeige der Zwischen-Informationen in der rechten Spalte der GUI. Dieser Unterschied hat seine Wurzeln jedoch in der Inkonsistenz zwischen dem CTT-Baum des Car Rental und der GUI-MockUps welche online dargestellt werden. Diese Eigenschaft kann man sich so vorstellen, dass z.B. auf der zweiten GUI-Seite, zusätzlich zu der Abfrage des konkreten Autos die bereits angegebene Such-Parameter zur Buchung eines Autos auch angezeigt werden. Dieses Verhalten ist zwar in den Referenz-Implementierungen (Mock Ups) eingebaut, im CTT-Baum ist es jedoch nicht modelliert. Bei der Implementierung des Car Rental in UCP hat man sich in dieser Arbeit vollkommen an die CTT-Spezifikation gehalten. Die Modellierung dieses Verhaltens in UCP, könnte z.B. mit Hilfe der Title Relation erfolgen.

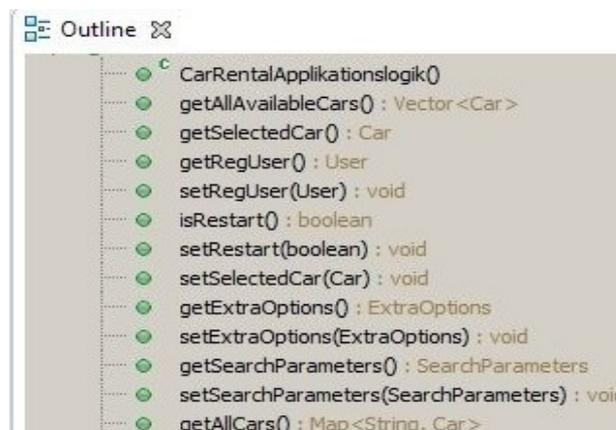
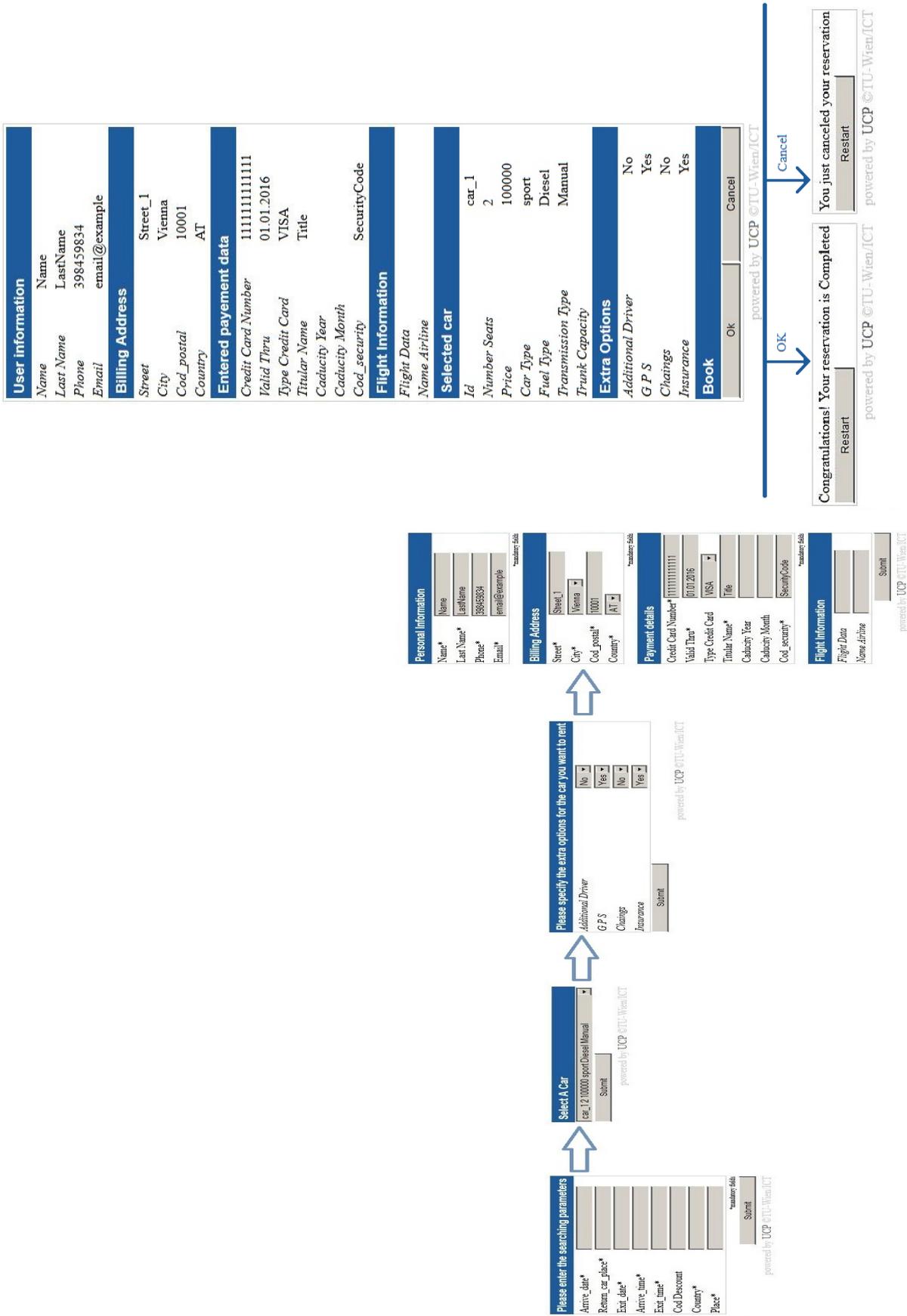


Abbildung 6.10: Ausschnitt aus der Applikationslogik von Car Rental

⁷<http://sites.uclouvain.be/mbui/index.php?t=carRenting/v3>



(a) Car Rental Prozess

User information	
Name	Name
Last Name	LastName
Phone	398459834
Email	email@example
Billing Address	
Street	Street_1
City	Vienna
Cod_postal	10001
Country	AT
Entered payment data	
Credit Card Number	1111111111111111
Valid Thru	01.01.2016
Type Credit Card	VISA
Titular Name	Title
Caducity Year	
Caducity Month	
Cod_security	SecurityCode
Flight Information	
Name Airline	
Selected car	
Id	car_1
Number Seats	2
Price	100000
Car Type	sport
Fuel Type	Diesel
Transmission Type	Manual
Trunk Capacity	
Extra Options	
Additional Driver	No
GPS	Yes
Changes	No
Insurance	Yes
Book	
Ok	Cancel

The flow diagram shows the sequence of these steps, with arrows indicating the progression from search to completion. The final step is a confirmation message: "Congratulations! Your reservation is Completed" with a `Restart` button.

(b) Ergebnis des Car Rental

Abbildung 6.11: FUI der Car Rental Anwendung

7 DISKUSSION UND ZUSAMMENFASSUNG

Es folgt eine kurze Diskussion und eine Zusammenfassung dieser Arbeit.

7.1 Diskussion

Die Untersuchungen haben gezeigt dass, obwohl das Thema der modellgetriebenen GUI-Entwicklung seit Jahren in der Forschung behandelt wird, dieses noch von einer Industriereife entfernt ist.

Ein wichtiger Punkt, welcher im Zusammenhang mit MDUID behandelt werden sollte und bis jetzt kaum Aufmerksamkeit gefunden hat, ist die der Messung von *quantitativen Merkmalen* der Eingangsmodellen. In dieser Arbeit wurden diese Merkmale zwar teilweise angesprochen, jedoch wurde auf einen quantitativen Vergleich der Eingangsmodelle nicht eingegangen, da dies den Rahmen der Arbeit sprengen würde. Hier sollten geeignete *Metriken* definiert werden, um die *Qualität* der Metamodelle messbar und vergleichbar zu machen.

Die mit den Ansätzen gelieferte Werkzeuge bieten viele weitere Möglichkeiten, die jedoch in dieser Arbeit nicht überprüft werden könnten. Dies hatte unterschiedliche Gründe. Die Werkzeuge, die im Zusammenhang mit UsiXML 1.0 versprochen werden, waren nicht alle für UsiXML 2.0 verfügbar und daher lag auch keine Möglichkeit vor, diese zu untersuchen. Die in CTTE und MARIAE implementierte Werkzeuge versprechen viele zusätzliche Vorteile, deren Untersuchung jedoch den Rahmen dieser Arbeit sprengen würde.

Ein wesentliches Hindernis für einen kommerziellen Einsatz der MDUID-Ansätze ist eine Skalierbarkeit. Alle untersuchten Ansätze werden vor allem auf der Ebene der Rechenkapazität, im Falle einer umfangreicheren Anwendung, an ihre Grenzen stoßen. Hier sind Implementierungen gefragt, die eine Transformation der Modelle in Benutzungsschnittstellen durchführen können und gleichzeitig für die Skalierbarkeit dieser Modelle sorgen.

Ein Aspekt, welcher in dem Vergleich nur indirekt berücksichtigt wurde, da er nicht zum Kern dieser Arbeit zählt, ist der der *Prozesse* in MDUID. Den Prozessen, die beim MDUID eingesetzt werden, kommt eine große Bedeutung zu, da sie über die Steigerung der Produktivität eines Entwicklers entscheiden [RKP⁺14].

Die modellgetriebene Entwicklung von GUIs wird höchstwahrscheinlich immer ein semi-automatisches Prozess bleiben. Der Aspekt der Usability ist der Hauptgrund, der einer voll automatischen Generierung entgegenwirkt, da es schwer ist, Aspekte der Usability im generierten Source Code zu

berücksichtigen. Trotzdem bietet MDUID vielversprechende Möglichkeiten, die durch eine stärkere Standardisierung der Ansätze und der hierdurch wachsender *Wiederverwendbarkeit* der Modelle stärker zur Erhöhung der Produktivität der Entwickler beitragen können.

7.2 Zusammenfassung

Der Zweck eines modellgetriebenen Ansatzes zur Generierung von Benutzungsschnittstellen ist es, die bekannten Vorteile aus modellgetriebenen Ansätzen in der Softwaretechnik zur Generierung von UIs einzusetzen. Dabei definiert man, ausgehend von einem Eingangsmodell, mit Hilfe von Transformationsregeln, Development Paths, die zur Generierung von verschiedenen Modellen führen. Als letztes Modell liegt das GUI, implementiert in einer gegebenen Technologie vor. Die sich hieraus ergebenden Vorteile haben in den letzten Jahren vermehrt das Interesse von Forschungseinrichtungen geweckt. Als Ergebnis dieses Interesses gibt es eine Menge an unterschiedlichen Ansätzen, für die es allerdings wenige konkrete Vergleichsstudien gibt.

Ziel dieser Diplomarbeit war ein Vergleich etablierter Ansätze für modellgetriebene Entwicklung von GUIs. Dieser Vergleich erfolgte in zwei Schritten. Im ersten Schritt wurden die Ansätze in einem konzeptionellen Vergleich, nach der Klassifizierung ihres Aufbaus im CRF-Schema mit eigens dafür definierten Kriterien verglichen. Im zweiten Teil wurde ein praktischer Vergleich durchgeführt, der auf einem W3C Use Case aufbaute.

Es wurden für die unterschiedlichen Ansätze die definierten Modelle und die angebotenen Werkzeuge untersucht. Abbildung 7.1 stellt eine Zusammenfassung jener Eigenschaften dar, die im Zusammenhang mit den Ansätzen erwähnt worden sind. Bei den Interaktionsmodellen standen die Task-basierte Modellierung und die Diskurs-basierte Modellierung im Mittelpunkt. Es hat sich gezeigt, dass beide Ansätze, obwohl sie grundlegend unterschiedliche Lösungsstrategien verfolgen, zum Großteil das gleiche Spektrum an Problemstellungen abdecken. Sowohl die Unterschiede, als auch die Gemeinsamkeiten dieser Ansätze wurden im Zuge des Vergleichs aufgezeigt. Bezüglich Skalierbarkeit der Modelle wies UCP durch ihr DoD Model einen wesentlichen Vorteil gegenüber MARIA auf. MARIA stellt dafür als einziger Ansatz die Möglichkeit zur Modellierung von Collaborative Tasks zur Verfügung.

Die Werkzeug-Unterstützung in MARIAE und UCP erwies sich in einem Stadium, in dem erste Prototypen in absehbarer Zeit modelliert und die entsprechende GUIs generiert werden können. UsiXML 2.0 erwies sich, trotz seiner vollkommene Übereinstimmung mit CRF, als ein eher theoretischer Ansatz, für den keine Werkzeug-Unterstützung vorhanden ist. Es wurden jedoch die Modell-Editoren für UsiXML 2.0 evaluiert. Aus den Informationen im UsiXML CUI wurde manuell eine GUI implementiert. Daraus folgte, dass das CUI in UsiXML alle zur Generierung eines UIs benötigten Informationen beinhaltet.

Das Thema der Dokumentation ist ein Punkt, welcher im Zuge des Vergleichs als nur genügend erarbeitet angesehen wurde. Es war bei der Implementierung praktischer Beispiele in unterschiedlichen Ansätzen zu beobachten, dass eine bessere Dokumentation vor allem für die anzuwendenden Methoden bzw. Prozessen einen erheblichen Vorteil bezüglich Entwicklungszeit und Erstellung eines Prototyps geschaffen hätte.

Die Möglichkeit Hilfe-Funktionen zu generieren, um den Anwender durch einen Prozess zu begleiten, welcher er über die GUI abarbeiten möchte, wurde zwar von allen Ansätzen angesprochen, praktisch implementiert wurde sie jedoch nur von MARIA.

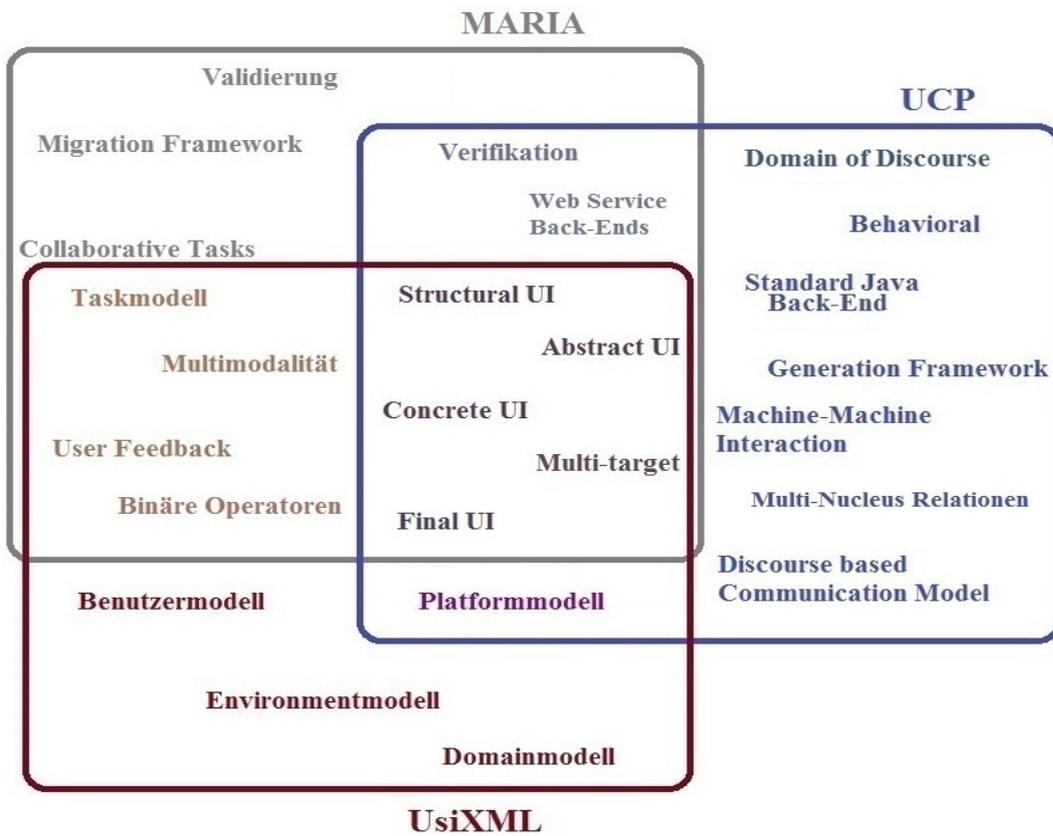


Abbildung 7.1: Die Eigenschaften der Ansätze im Überblick

Es zeigte sich weiterhin, dass für einen Erfolg der MDUID-Ansätze die Themen der Usability und Standardisierung einer höheren Aufmerksamkeit und mehr Forschung bedürfen. Die Möglichkeit zur Anpassung von Modellen, welche auch im konzeptionellen Vergleich angesprochen wurde, ist ein guter Ansatz, die Usability der generierten GUIs zu verbessern. Dieser Ansatz bedarf aber weiterer Überlegungen, um die Möglichkeiten einer MDUID besser auszuschöpfen.

A CAR RENTAL TASK-BAUM

Die folgenden Abbildungen stellen den vollständigen Task-Baum für das Car Rental Use Case dar und wurden von <http://sites.uclouvain.be/mbui/trees/carRenting/v3/task\hskip.5em\relaxmodel.png> kopiert.

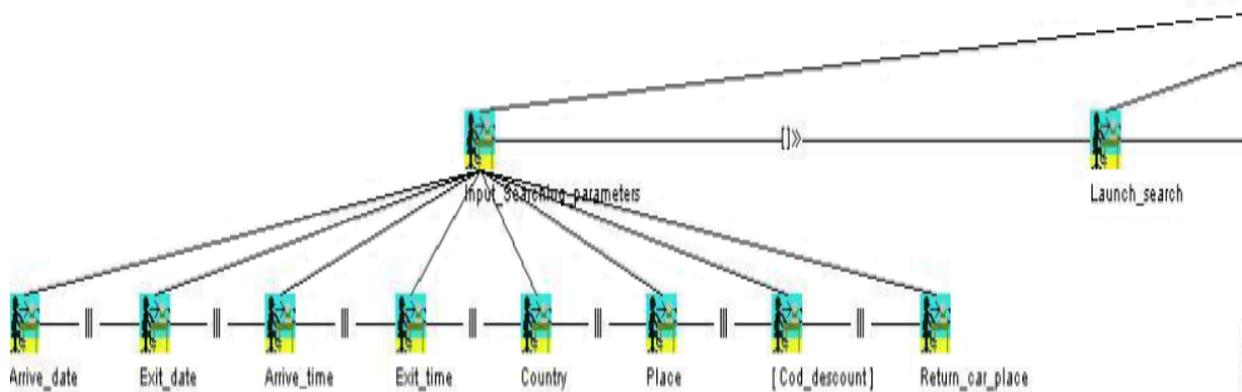


Abbildung A.1: Task-Baum der Car Rental-Anwendung, Teil 1

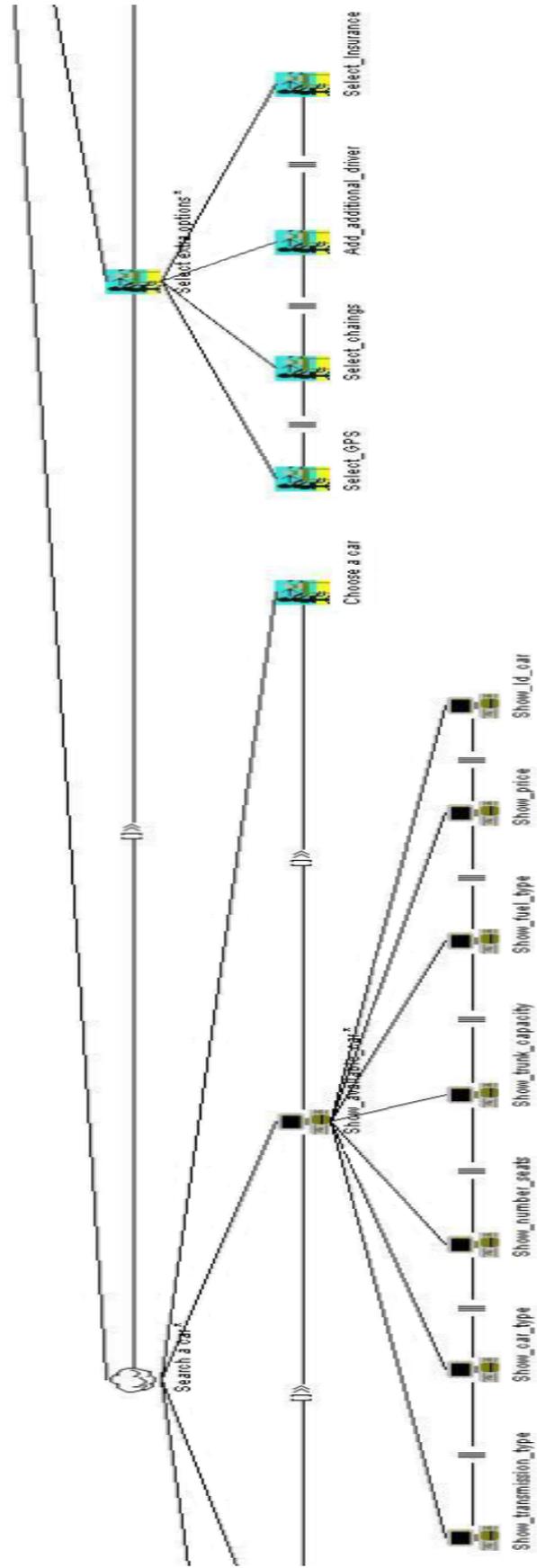


Abbildung A.2: Task-Baum der Car Rental-Anwendung, Teil 2

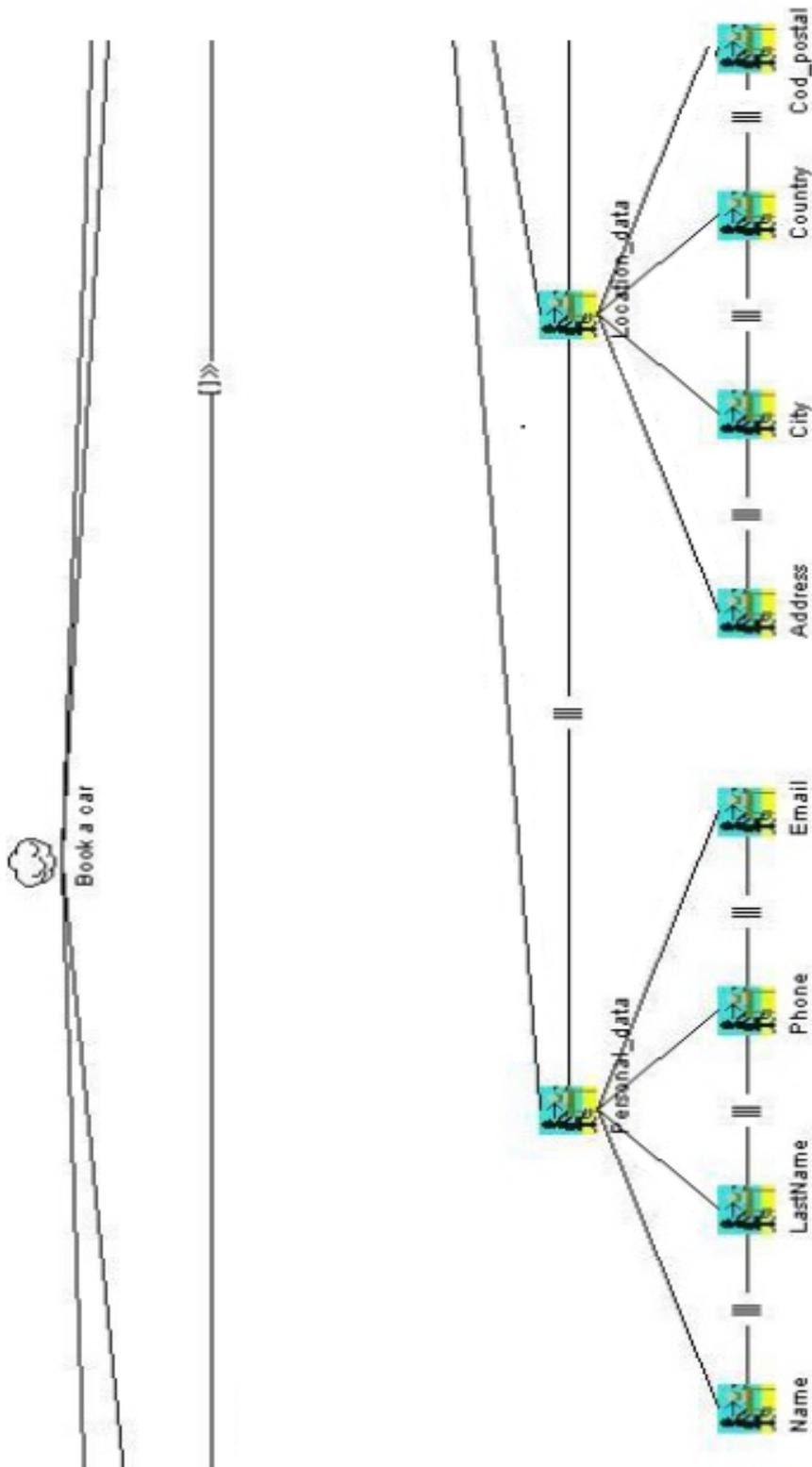


Abbildung A.3: Task-Baum der Car Rental-Anwendung, Teil 3

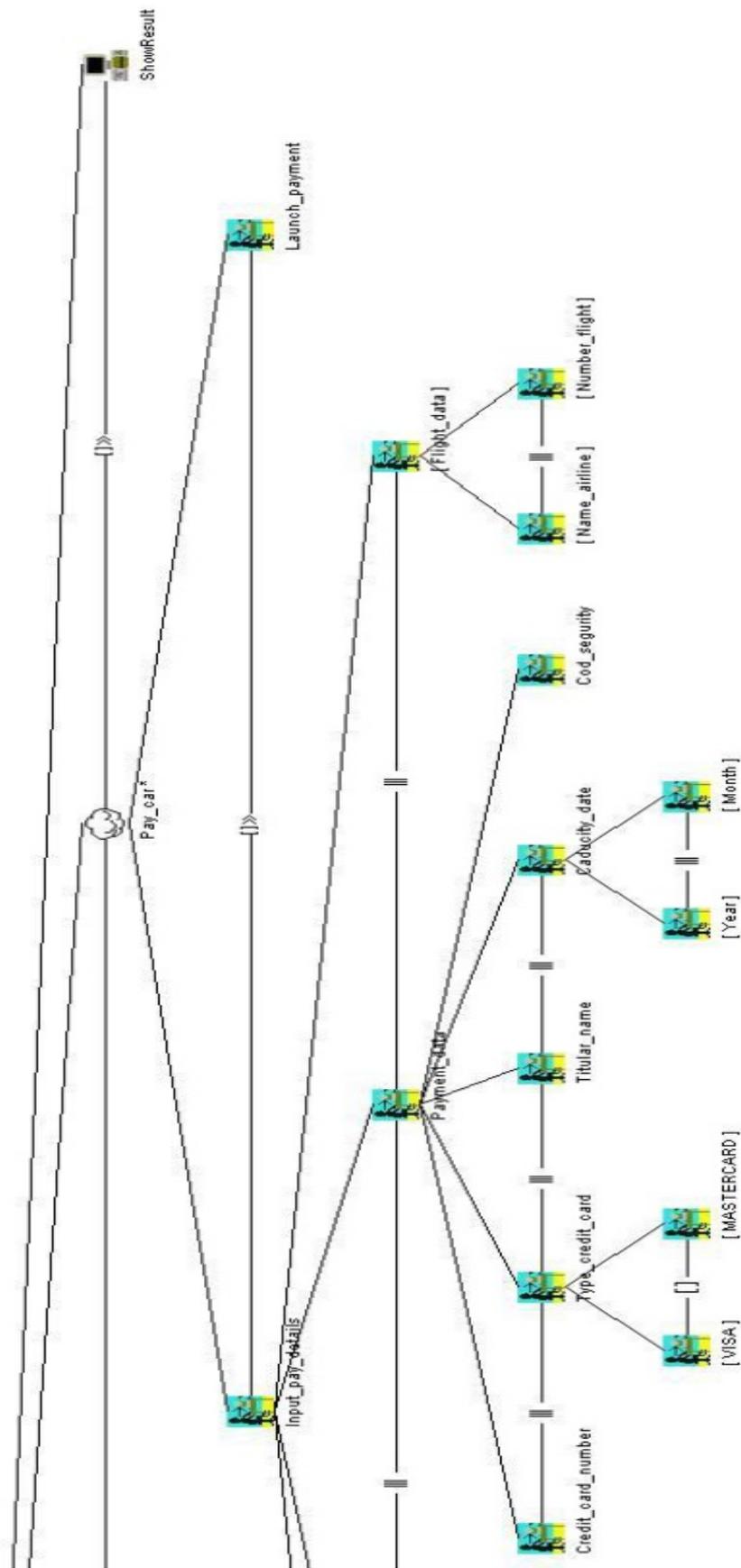


Abbildung A.4: Task-Baum der Car Rental-Anwendung, Teil 4

ABBILDUNGSVERZEICHNIS

2.1	Modelle im CAMELEON Reference Framework (kopiert von [CClvoV+02])	7
2.2	CAMELEON Reference Framework für einen Design Time Process	8
3.1	Task-Kategorien in CTTE	11
3.2	Temporale (High Priority) Operatoren in CTTE	12
3.3	CTT-Modellierung der Login-Anwendung	13
3.4	Beispiel für eine Binding in MARIAE	14
3.5	Datenmodell der Login-Anwendung	16
3.6	Event-Modell für das Login-Beispiel	17
3.7	Ein Presentation Task aus der Login-Anwendung in MARIAE	18
3.8	Beispiel einer Vorbedingung in CTTE	20
3.9	Der CTTE Simulator	21
3.10	Consistency Check in CTTE	22
3.11	CTT browser und service binding in MARIAE	23
3.12	CUI Editor in MARIAE	25
3.13	FUI der Login-Anwendung in MARIAE	26
4.1	Taskmodell	29
4.2	Ein Domain-Modell für die Login-Anwendung gemäß UsiXML	30
4.3	Context-Modell	31
4.4	Ausschnitt aus dem AUI-Metamodell in UsiXML (kopiert von [VBMT12])	33
4.5	AUI-Modell	34
4.6	Ausschnitt des CUI-Metamodells in UsiXML (kopiert von [CMP ⁺])	35
4.7	FUI für die Login-Anwendung	36

5.1	UCP	38
5.2	DoD Model der Login-Anwendung	39
5.3	Action-Notification Model in UCP	40
5.4	Discourse-Diagramm der Login-Anwendung	41
5.5	Beispiel für Propositional Content in UCP	43
5.6	Alternatives Discourse Model für die Login-Anwendung	43
5.7	UCP:UI im Überblick (kopiert von [Pop12])	44
5.8	PSTM-Metamodell	45
5.9	UCP	45
5.10	FUI der Login-Anwendung	46
5.11	Discourse Model Editor	48
5.12	Tool-Unterstützung zur Generierung eines FUIs in UCP	50
5.13	Weitere Tool-Unterstützung zur Generierung eines FUIs in UCP	51
6.1	CRF-Darstellung für MARIA	53
6.2	CRF-Darstellung für UsiXML	54
6.3	CRF-Darstellung für UCP	55
6.4	Gegenüberstellung der Diskurs-basierten Kommunikation und der Task-Modellierung	62
6.5	CTT-Ausschnitt des Car Rental (kopiert von der Homepage der uclouvain)	70
6.6	Discourse Model der Car Rental-Anwendung	72
6.7	Vergleich der Granularitäten in Discourse Model und CTT	73
6.8	Action-Notification Model der Car Rental-Anwendung	74
6.9	Domain of Discourse Model der Car Rental-Anwendung	75
6.10	Ausschnitt aus der Applikationslogik von Car Rental	76
6.11	FUI der Car Rental Anwendung	77
7.1	Die Eigenschaften der Ansätze im Überblick	80
A.1	Task-Baum der Car Rental-Anwendung, Teil 1	81
A.2	Task-Baum der Car Rental-Anwendung, Teil 2	82
A.3	Task-Baum der Car Rental-Anwendung, Teil 3	83
A.4	Task-Baum der Car Rental-Anwendung, Teil 4	84

WISSENSCHAFTLICHE LITERATUR

- [ABB⁺08] Roberto Acerbis, Aldo Bongio, Marco Brambilla, Stefano Butti, Stefano Ceri, and Piero Fraternali. Web applications design and development with webml and webratio 5.0. In RichardF. Paige and Bertrand Meyer, editors, *Objects, Components, Models and Patterns*, volume 11 of *Lecture Notes in Business Information Processing*, pages 392–411. Springer Berlin Heidelberg, 2008.
- [B.87] Myers B. Gaining general acceptance for uimss. *Computer Graphics*, 21(2):130–134, 1987.
- [BBF⁺87] B Betts, David Burlingame, Gerhard Fischer, Jim Foley, Mark Green, David Kasik, Stephen T Kerr, Dan Olsen, and James Thomas. Goals and objectives for user interface software. *SIGGRAPH Comput. Graph.*, 21(2):73–78, April 1987.
- [BDRS96] Thomas Browne, David Davila, Spencer Rugaber, and R. E. Kurt Stirewalt. The mastermind user interface generation project. Technical report, Georgia Institute of Technology, 1996.
- [BHL⁺95] Francois Bodart, Anne-Marie Hennebert, Jean-Marie Leheureux, Isabelle Provot, Benoit Sacre, and Jean Vanderdonckt. Towards a systematic building of software architecture: the trident methodological guide. In Philippe Palanque and Remi Bastide, editors, *Design, Specification and Verification of Interactive Systems'95*, Eurographics, pages 262–278. Springer Vienna, 1995.
- [BKFP08] Christian Bogdan, Hermann Kaindl, Jürgen Falb, and Roman Popp. Modeling of interaction design by end users through discourse modeling. In *Proceedings of the 2008 ACM International Conference on Intelligent User Interfaces (IUI 2008)*, Maspalomas, Gran Canaria, Spain, 2008. ACM Press: New York, NY.
- [CClvoV⁺02] G. Calvary, J. Coutaz, D. Thevenin (First Version last version of V1.1), L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt (2nd version of V1.1), L.Marucci, F.Paternò, and C.Santoro (3rd version of V1.1). The cameleon reference framework. <http://giove.isti.cnr.it/projects/cameleon/pdf/CAMELEOND1.1RefFramework.pdf>, 2002.
- [CCT⁺03a] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *INTERACTING WITH COMPUTERS*, 15:289–308, 2003.
- [CCT⁺03b] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.
- [CMP⁺] Juan M. González Calleros, Gerrit Meixner, Fabio Paternò, Jaroslav Pullmann, Dave Raggett, Daniel Schwabe, and Jean Vanderdonckt. Model-Based UI XG Final

- Report. <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/>.
- [CNM83] Stuart K. Card, Allen Newell, and Thomas P. Moran. *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.
- [E.96] Schulungbaum E. Model-based user interface software tools-current state of declarative models. Technical report, Graphics, Visualization nad Usability Center, Georgia Institute of Technology, 1996.
- [EK12] Dominik Ertl and Hermann Kaindl. Semi-automatic generation of multimodal user interfaces for dialogue-based interactive systems. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction (ICMI'12)*, New York, NY, USA, 2012. ACM.
- [Ert11] Domink Ertl. *Semi-Automatic Generation of Multimodal User Interfaces for Dialogue-based Interactive Systems*. PhD thesis, E384, 2011.
- [FKH⁺06] Jürgen Falb, Hermann Kaindl, Helmut Horacek, Cristian Bogdan, Roman Popp, and Edin Arnautovic. A discourse model for interaction design based on theories of human communication. In *Extended Abstracts on Human Factors in Computing Systems (CHI '06)*, pages 754–759. ACM Press: New York, NY, 2006.
- [GF10] Alfonso García Frey. Self-explanatory user interfaces by model-driven engineering. In *Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '10*, pages 341–344, New York, NY, USA, 2010. ACM.
- [GM13] Joëlle Coutaz Gerrit Meixner, Gaëlle Calvary. Introduction to model-based user interfaces. 2013.
- [GVGC08] J. Guerrero, J. Vanderdonckt, and J.M. Gonzalez Calleros. Flowixml: a step towards designing workflow management systems. In *Journal of Web Engineering, Vol. 4, No. 2*, pages 163–182, 2008.
- [HH93] Deborah Hix and H. Rex Hartson. *Developing User Interfaces: Ensuring Usability Through Product & Process*. John Wiley & Sons, Inc., New York, NY, USA, 1993.
- [JF89] S. Kovacevic K. Murray J. Foley, W. Kim. The user interface design environment - a computer aided software engineering tool for the user computer interface, 1989.
- [Kai13] Hermann Kaindl. Model-based transition from requirements to high-level software design. In *APSEC*, pages 81–82, 2013.
- [KPR⁺11] Hermann Kaindl, Roman Popp, David Raneburger, Dominik Ertl, Jurgen Falb, Alexander Szep, and Cristian Bogdan. Robot-supported cooperative work: A shared-shopping scenario. *Hawaii International Conference on System Sciences*, 0:1–10, 2011.
- [KRF⁺09] Sevan Kavaldjian, David Raneburger, Jürgen Falb, Hermann Kaindl, and Dominik Ertl. Semi-automatic user interface generation considering pointing granularity. In *Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics (SMC 2009)*, San Antonio, TX, USA, Oct. 2009.
- [Lar02] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (2nd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.
- [LFG90] Paul Luff, David Frohlich, and Nigel Gilbert. *Computers and Conversation*. Academic Press, London, UK, January 1990.
- [LV09] Quentin Limbourg and Jean Vanderdonckt. Multipath transformational development of user interfaces with graph transformations. In Ahmed Seffah, Jean Vanderdonckt, and Michel C. Desmarais, editors, *Human-Centered Software Engineering*, Human-Computer Interaction Series, pages 107–138. Springer London, 2009.

- 10.1007/978-1-84800-907-3_6.
- [LVM⁺04] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins, and Daniela Trevisan. Usixml: A user interface description language for context-sensitive user interfaces. In *IN PROCEEDINGS OF THE ACM AVI'2004 WORKSHOP & QUOT;DEVELOPING USER INTERFACES WITH XML: ADVANCES ON USER INTERFACE DESCRIPTION LANGUAGES*, pages 55–62. Press, 2004.
- [MCC14] Gerrit Meixner, Gaëlle Calvary, and Joëlle Coutaz. Introduction to model-based user interfaces. Technical report, W3C, 2014.
- [MLJ06] F. Montero and V. López-Jaquero. Idealxml: An interaction design tool—a task-based approach to user interfaces design. In *Proc. of 6th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006)*, pages 245–252. Springer-Verlag, Berlin, 2006.
- [MM03] Eds.: Joaquin Miller and Jishnu Mukerjij. MDA guide version 1.0.1. Technical report, Object Management Group (OMG), 2003.
- [MPS02] Giulio Mori, Fabio Paternò, and Carmen Santoro. Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Trans. Softw. Eng.*, 28(8):797–813, August 2002.
- [MPS04] G. Mori, F. Paternò, and C. Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *Software Engineering, IEEE Transactions on*, 30(8):507 – 520, aug. 2004.
- [MPV11] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past, present, and future of model-based user interface development. *i-com*, 10(3):2–10, November 2011.
- [MPW11] Célia Martinie, Philippe Palanque, and Marco Winckler. Structuring and composition mechanisms to address scalability issues in task models. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part III, INTERACT'11*, pages 589–609, Berlin, Heidelberg, 2011. Springer-Verlag.
- [MR92] Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '92*, pages 195–202, New York, NY, USA, 1992. ACM.
- [MT88] W. C. Mann and S.A. Thompson. Rhetorical Structure Theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.
- [MV08] Benjamin Michotte and Jean Vanderdonckt. GrafiXML, a multi-target user interface builder based on UsiXML. In *Proceedings of the Fourth International Conference on Autonomic and Autonomous Systems*, pages 15–22, Washington, DC, USA, 2008. IEEE Computer Society.
- [Pat02] Fabio Paternò. Tools for task modelling: Where we are, where we are headed. In *Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design, TAMODIA '02*, pages 10–17. INFOREC Publishing House Bucharest, 2002.
- [Pat03] Fabio Paternò. Concurtasktrees: An engineered approach to model-based design of interactive systems. *The Handbook of Task Analysis for Human-Computer Interaction*, pages 483–503, 2003.
- [PFA⁺09] Roman Popp, Jürgen Falb, Edin Arnautovic, Hermann Kaindl, Sevan Kavaldjian, Dominik Ertl, Helmut Horacek, and Cristian Bogdan. Automatic generation of the behavior of a user interface from a high-level discourse model. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences (HICSS-42)*,

- Piscataway, NJ, USA, 2009. IEEE Computer Society Press.
- [Pop12] Roman Popp. A unified solution for service-oriented architecture and user interface generation through discourse-based communication models. Doctoral dissertation, Vienna University of Technology, Vienna, Austria, 2012.
 - [PRK13] Roman Popp, David Raneburger, and Hermann Kaindl. Tool support for automated multi-device GUI generation from discourse-based communication models. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive computing systems*, EICS '13, New York, NY, USA, 2013. ACM.
 - [PS94] Angel R. Puerta and Pedro Szkeley. Model-based interface development. In *Conference Companion on Human Factors in Computing Systems*, CHI '94, pages 389–390, New York, NY, USA, 1994. ACM.
 - [PSS09] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4):19:1–19:30, November 2009.
 - [PSS10] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Exploiting web service annotations in model-based user interface development. In *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '10, pages 219–224, New York, NY, USA, 2010. ACM.
 - [PSS11] Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. Engineering the authoring of usable service front ends. *Journal of Systems and Software*, 84(10):1806 – 1822, 2011.
 - [PSSR] Fabio Paternò, Carmen Santoro, Lucio Davide Spano, and Dave Raggett. Mbui - task models. <http://www.w3.org/TR/task-models/>.
 - [PZ10] Fabio Paternò and Giuseppe Zichitella. End-user customization of multi-device ubiquitous user interfaces. In *Proceedings of the MDDAUI'10 Workshop on Model Driven Development of Advanced User Interfaces*, 2010.
 - [Ran08] David Raneburger. Automated graphical user interface generation based on an abstract user interface specification. Master's thesis, Vienna University of Technology, Vienna, Austria, 2008.
 - [RKP+14] David Raneburger, Hermann Kaindl, Roman Popp, Vedran Šajatović, and Alexander Armbruster. A process for facilitating interaction design through automated GUI generation. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14)*, 2014.
 - [RPK+11a] David Raneburger, Roman Popp, Hermann Kaindl, Jürgen Falb, and Dominik Ertl. Automated Generation of Device-Specific WIMP UIs: Weaving of Structural and Behavioral Models. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '11, pages 41–46, New York, NY, USA, 2011. ACM.
 - [RPK+11b] David Raneburger, Roman Popp, Sevan Kavaldjian, Hermann Kaindl, and Jürgen Falb. Optimized GUI generation for small screens. In Heinrich Hussmann, Gerrit Meixner, and Detlef Zuehlke, editors, *Model-Driven Development of Advanced User Interfaces*, volume 340 of *Studies in Computational Intelligence*, pages 107–122. Springer Berlin / Heidelberg, 2011.
 - [RPKF11] David Raneburger, Roman Popp, Hermann Kaindl, and Jürgen Falb. Automated WIMP-UI behavior generation: Parallelism and granularity of communication units. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 2816 –2821, Oct. 2011.

- [Sch10] Alexander Schörkhuber. Integritätsprüfung von diskursmodellen, transformationsregeln und strukturellen modellen von graphischen user interfaces. Master's thesis, Technische Universität Wien, Fakultät für Elektrotechnik und Informationstechnik, Institut für Computertechnik, E384, 2010.
- [Sea69] John R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England, 1969.
- [SLN92] Pedro Szekely, Ping Luo, and Robert Neches. Facilitating the exploration of interface design alternatives: The humanoid model of interface design, 1992.
- [Sta08] Adrian Stanciulescu. *Ph.D. thesis*. Université catholique de Louvain, Louvain-la-Neuve, Belgium, 25 June 2008, 2008.
- [Sze96] Pedro Szekely. Retrospective and challenges for model-based interface development. In *Design, Specification and Verification of Interactive Systems 96*, pages 1–27. Springer-Verlag, 1996.
- [Van05] Jean Vanderdonckt. A MDA-compliant environment for developing user interfaces of information systems. In *CAiSE*, pages 16–31, 2005.
- [Van08] Jean M. Vanderdonckt. Model-driven engineering of user interfaces: Promises, successes, and failures. In *Proceedings of 5th Annual Romanian Conf. on Human-Computer Interaction*, pages 1–10. Matrix ROM, Sept. 2008.
- [VBMT12] Jean Vanderdonckt, François Beuvs, Jérémie Melchior, and Riccardo Tesoriero. User Interface eXtensible Markup Language (UsiXML), W3C Working Group Submission 1 February 2012. http://www.w3.org/wiki/images/5/5d/UsiXML_submission_to_W3C.pdf, 2012.
- [VdVLB96] G. Van der Veer, B. Lenting, and B. Bergevoet. *Gta: Groupware task analysis - modeling complexity*, 1996.
- [VG06] Ralf Laue Volker Gruhn. *Komplexitätsmetriken für geschäftsprozessmodelle*, 2006.
- [VJ99] Puerta A.R. Vanderdonckt J.M. Introduction to computer-aided design of user interfaces, preface of CADUI 99, 1999.
- [Š14] Vedran Šajatović. Improved tool support for model-driven development of interactive applications in UCP. Master's thesis, Vienna University of Technology, 2014.