Technische Universität Berlin
April 2001

Use Case Maps
und Software Engineering

Jan Wloka
<jwloka@cs.tu-berlin.de>

Ausarbeitung zum gleichnamigen Referat im Rahmen der Lehrveranstaltung
"Komponentenkonzepte für das Continuous Engineering"
an der Technischen Universität Berlin im Wintersemester 2000/01*

Veranstalter: Prof. Dr. Herbert Weber und Felix Bübl

Kurzfassung

Die Verwendung von Use Case Maps (UCMs), besonders als Werkzeug zur Unterstützung der UML, erfordert keine großen Lehrübungen oder viel Literatur. Im Folgenden wird beispielhaft erläutert, wie wenige Elemente der UCM-Notation ausreichen, um erhebliche Schwierigkeiten mit UML-Diagrammen zu meistern. Mit diesem Ziel wird die UCM-Notation sowie die Vorgehensweise erläutert und beispielhaft eine UCM erstellt. Neben der Identifikation von Komponenten, Verantwortlichkeiten und Szenarien werden nützliche Konzepte zur Strukturierung, Erweiterung und Generalisierung von UCMs kurz beschrieben und anschaulich verwendet. Zusätzlich und eher überblickartig werden Verwendungsmöglichkeiten vorgestellt und die Verwendung von UCMs in einen Softwareentwicklungsprozeß eingeordnet.

Das "High level design magic" - Problem

Die derzeit am häufigsten verwendete Modellierungssprache ist die *Unified Modeling Language (UML)* [OMG00]. Sie beinhaltet viele Konzepte und Notationen. Besonders zur Spezifikation, Visualisierung und Dokumentation von objektorientierten bzw. komponentenbasierten Softwaresystemen. Unter Verwendung der verschiedenen Diagrammarten wird ein System dabei aus vielen unterschiedlichen Sichten beschrieben.

DIE KONZEPTIONELLE LÜCKE

Am Beginn von modernen Softwareentwicklungsprozessen werden UML-Anwendungsfalldiagramme (*use case diagrams*) verwendet, um alle Aktionen eines Systems zu definieren, die einem bestimmten Nutzer gewünschte Ergebnisse liefern. Das System wird dabei aus der Sicht dieses Nutzers als "*black box*" in Form von

*) http://cis.cs.tu-berlin.de/Lehre/WS-0001/Sonstiges/konteng_sem/Terminuebersicht.html

Szenarien beschrieben (szenariobasierter Ansatz). Aufbauend auf dieser sehr abstrakten Sicht wird meist versucht, die kausalen Zusammenhänge zwischen den einzelnen Anwendungsfällen in Entwurfszenarien festzuhalten, oft als Fließtext oder unter Verwendung der UML-Sequenzdiagramme (*sequence diagrams*). Diese werden manchmal auch durch Zustandsübergangsdiagramme (*statecharts*) und/oder Aktivitätsdiagramme (*activity diagrams*) unterstützt. Beim Beschreiben der kausalen Zusammenhänge ist es aber aufgrund der verwendeten Notationen schon nötig, die Struktur des Systems (implizit) zu definieren. Das entstandene Modell enthält dadurch genügend Informationen, um das System direkt zu implementieren.

Sollen komplexe Systeme entwickelt werden, ist Einfachheit und Übersichtlichkeit in den Diagrammen noch wichtiger. Durch die Verwendung von Sequenzdiagrammen zur Realisierung der Anwendungsfälle entstehen jedoch Strukturen, die sich gegenseitig ausschließen oder einfach unüberschaubar sind. Eine wirkliche Unterstützung bei der Implementierung ist also oft nicht gegeben, so daß sich die Frage stellt, warum man das System überhaupt modelliert.
Der Hauptgrund für die subjektiv empfundene Nutzlosigkeit der Modelle ist der Versuch, fast strukturlose Funktionalität (Anwendungsfalldiagramme) in einem einzigen Schritt auf eine noch nicht vorhandene Systemstruktur (Sequenzdiagramme) abzubilden[1]. Dieser Übergang zwischen der anwendungsorientierten *black-box* und der systeminternen *white-box* Sicht eines Systems wird auch **konzeptionelle Lücke** genannt [AmMu00]. Besonders bei größeren Systemen mit komplexen Anwendungsfällen ist es ohne viel Erfahrung, Abstraktionsvermögen und der entsprechenden “Vision” im Hinterkopf fast unmöglich, zu einem geeigneten Modell zu gelangen.

EIN EINFACHES BEISPIEL

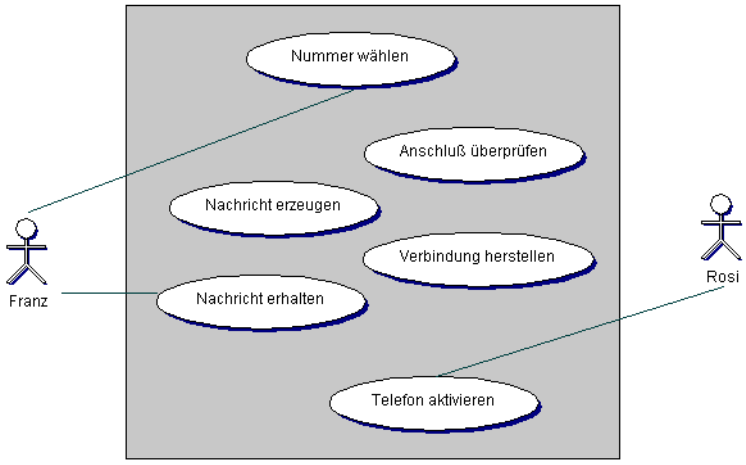
Es soll ein System entwickelt werden, daß einen sehr einfachen Telefondienst zur Verfügung stellt. Die Anforderungen an das System sind aus Sicht des Benutzers Franz wie folgt definiert:
"Der Benutzer Franz versucht, seine Sachbearbeiterin Rosi in der Vertragsverwaltung anzurufen. Er kennt ihre Telefonnummer genau und wählt diese. Der Status des Zielanschlusses wird überprüft. Ist er frei, wird eine Verbindung hergestellt und das Telefon von Rosi aktiviert. Ansonsten wird eine Besetztnachricht erzeugt und an Franz gesendet."

Die entscheidenden Elemente im beschriebenen Szenario sind leicht zu identifizieren:
„Der Benutzer Franz versucht, seine Sachbearbeiterin Rosi in der Vertragsverwaltung anzurufen. Er kennt ihre Telefonnummer genau und wählt diese. Der Status des Zielanschlusses wird überprüft. Ist er frei, wird eine Verbindung hergestellt und das Telefon von Rosi aktiviert. Ansonsten wird eine Besetztnachricht erzeugt und an Franz gesendet.“

Es lassen sich dabei zwei Akteure identifizieren, Franz und Rosi. ***Franz*** versucht mit ***Rosi*** zu kommunizieren und verwendet dabei Funktionen wie `Nummer wählen`, `Anschluß überprüfen`, `Verbindung herstellen`, etc., die vom System zur Verfügung gestellt werden. Die Anwendungsfälle mit ihren Beziehungen untereinander - also auch mit den Akteuren - lassen sich sehr übersichtlich in einem Anwendungsfalldiagramm darstellen (Abb. 1).

1. Auf UML-Aktivitätsdiagramme als Hilfestellung in einem szenariobasierten Prozeß geht Anhang A näher ein.

ABBILDUNG 1. Aus den Anforderungen identifizierte Anwendungsfälle



Nachdem festgelegt wurde, was das System leisten soll, muß es irgendwie realisiert werden. Aus den Anforderungen können die folgenden zwei Szenarien entnommen werden:

1. Besetzt-Szenario: Franz wählt – Anschluß nicht frei – Besetztnachricht erzeugen – Besetztzeichen bei Franz
2. Erfolgsszenario: Franz wählt – Anschluß frei – Verbindung herstellen – Rosis Telefon klingelt

Dem szenariobasierten Ansatz folgend, wird ein zugehöriges Sequenzdiagramm entwickelt, um die Szenarien auf Systemfunktionalitäten abzubilden. Die erste Aufgabe ist das Verteilen der Szenarioelemente auf einzelne Komponenten des Systems. Dabei müssen Verantwortlichkeiten im System verteilt werden, obwohl die Systemstruktur noch nicht definiert ist. Insbesondere erzwingt ein Sequenzdiagramm das Festlegen der Struktur auf der untersten Abstraktionsebene (Klassen mit Nachrichtenaustausch), so daß eine grobe Systemstruktur schon vorhanden sein muß. Dieser Schritt stellt für Softwareentwickler im allgemeinen ein großes Problem dar. R.J.A Buhr[1] bezeichnet ihn daher treffend als “*high level design magic*”.

GETRENNTE DARSTELLUNG VON STRUKTUR UND VERHALTEN

In der UML werden Struktur und Verhalten in getrennten Diagrammen dargestellt. So werden Funktionalitäten (Verhalten) z.B. in Anwendungsfalldiagrammen beschrieben, während die Struktur eines Systems in Klassendiagrammen enthalten ist. Eine explizite Verbindung dieser beiden Dimensionen ist in der UML nicht möglich. Einzig Interaktionsdiagramme ermöglichen die Zuordnung von Funktionalitäten zur Systemstruktur. Diese ist jedoch so implementationsnahe, daß ein übergeordnetes Verständnis und eine geeignete Systemstruktur Voraussetzung sind. Für das Verstehen des Systems ist aber gerade eine gemeinsame Darstellung von Struktur und Verhalten auf Systemebene von großer Bedeutung. Diese verknüpft die verschiedenen Diagrammarten der UML so miteinander, daß die Verteilung der Funktionalitäten auf Komponenten explizit in nur einem Diagramm dargestellt wird.

Zusammengefaßt fehlt beim Entwurf eines Softwaresystems ein Zwischenschritt, der die Möglichkeit bietet, auf Systemebene Komponenten festzulegen und ihnen

1. R.J.A. Buhr - Einer der Autoren von [BuCa96] und Erfinder von UCMs.

explizit Funktionalitäten zuzuweisen, bevor die Beziehungen zwischen den Klassen auf der Ebene von Nachrichtenaustausch und Protokolldefinitionen festgelegt sind. Unter anderem für diesen Zweck wurde das Konzept der *Use Case Maps* entwickelt. Es beruht im wesentlichen auf zwei Grundgedanken [Reiser99]:

- Übersichtlichkeit und Einfachheit der Modelle durch Verzicht auf eine Detailansicht,
- Verbindung der Struktur des Systems mit seinem Verhalten in einem einzigen Modell

Das Konzept der Use Case Maps (UCMs)

Wie der Name vermuten läßt, basieren UCMs auf der Idee von Anwendungsfällen (*use cases)* und Entwurfszenarien. Ein Anwendungsfall ist hierbei ein möglicher Fall der Anwendung eines Systems. Unter einem Szenario ist hingegen eine Abfolge von Systemeingaben/-ausgaben und internen Aktionen zu verstehen, die das System ausführt, um die Aufgaben eines entsprechenden Anwendungsfalls zu bewältigen. Eine UCM als Integration der beiden Konzepte beschreibt nun die kausalen Beziehungen zwischen den Aktionen der Anwendungsfälle. Hierbei ist es möglich, die Aktionen als Verantwortlichkeiten direkt auf Komponenten zu verteilen.

WAS SIND USE CASE MAPS?

UCMs als Diagrammart. Mit dem Begriff “Use Case Maps” ist in der Regel eine Diagrammart gemeint. UCM-Diagramme ermöglichen die Darstellung eines Systems auf einem sehr hohen Abstraktionsniveau, ein sogenanntes “*big picture*”. Zusätzlich erlauben sie, Systemstruktur in Verbindung mit seinem Verhalten integriert darzustellen. Dies ist ohne Darstellung von Nachrichtenaustausch möglich, und erleichtert somit das Erkennen von Abhängigkeiten. UCM-Diagramme sollen die Grundlage von detaillierteren Modellen bilden, um nicht zuletzt auch die Kommunikation zwischen allen Beteiligten zu verbessern.

UCMs als Notation. Es kann mit UCMs auch die Notation gemeint sein. Diese UCM-Notation wird einerseits graphisch in den Diagrammen verwendet, andererseits bietet sie auch für jedes Element eine formal textuelle Repräsentation in XML. Damit ist es einfacher, UCM-Diagramme z.B. auf Vollständigkeit oder Konsistenz zu überprüfen oder auch Spezifikationen automatisiert zu erzeugen [AmBuGr99].

UCMs als Modellierungstechnik. Zusätzlich bezeichnet der Begriff UCMs eine Modellierungstechnik. Sie ist besonders geeignet zur Modellierung von sehr komplexen Systemen auf einem sehr hohen Abstraktionsniveau. Im wesentlichen beschreibt sie, wie UCM-Diagramme verwendet werden können, um sie als Grundlage detaillierterer Modelle zu verwenden.

Wie werden Use Case Maps erstellt?

Mit dem Ziel einfach und übersichtlich zu sein, beschränkt sich die UCM-Notation im wesentlichen auf Komponenten und Pfadelemente (z.B. kausal verknüpfte Verantwortlichkeiten). Diese Grundnotation läßt sich um eine Vielzahl von Ausdrucksmitteln erweitern, um auch deutlich komplexeren Anforderungen gerecht zu werden.

Dies spiegelt den ersten Grundgedanken von UCMs wieder - Übersichtlichkeit und Einfachheit - und ermöglicht einen schnelleren Zugang zur Notation.

WELCHE INFORMATIONEN WERDEN BENÖTIGT?

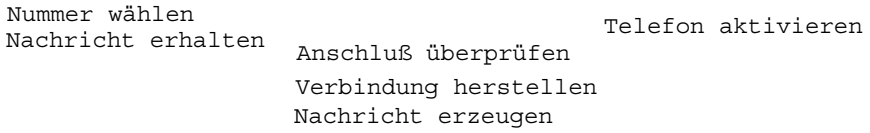
UCMs lassen sich im allgemeinen direkt aus den (informal definierten) Anforderungen erstellen. Dabei ist lediglich darauf zu achten, daß die Aufgaben bzw. Verantwortlichkeiten identifizierbar sind. Natürlich können aber auch schon vorhandene Anwendungsfalldiagramme als Grundlage verwendet werden, was u.a. das Festlegen der Schnittstelle zwischen System und Umgebung vereinfacht, wodurch das Positionieren von Start-/Endpunkten erleichtert wird. Eine zusätzliche Hilfe ist die Beschreibung von Komponenten in Art, Charakter und ihren Beziehungen untereinander (beispielsweise kann eine Komponente Netzwerk Subkomponenten wie Zugriffssteuerung, Benutzerverwaltung und Nachrichtenerzeugung enthalten).
Es können nun zuerst separate UCMs für einzelne Systemfunktionalitäten oder Szenarien erstellt werden, die Stärke von UCMs liegt jedoch darin, viele Szenarien integriert in einer UCM darzustellen.

VERSCHIEDENE VORGEHENSWEISEN SIND MÖGLICH

UCMs lassen sich auf verschiedene Weisen erstellen. Es können zuerst die Verantwortlichkeiten identifiziert, diese entsprechend gruppiert und dann einer bestimmten Komponente zugewiesen werden. Eine andere Möglichkeit wäre, zuerst die Verantwortlichkeiten einem bestimmten Szenario zuzuweisen, wobei man versucht, die Komponenten “unterwegs” zu identifizieren. Vielleicht ist es aber auch einfacher, Struktur und Funktionalität getrennt zu identifizieren und erst anschließend beide Sichten zusammenzufügen.

Zurück zum Beispiel. Ausgehend vom Anwendungsfalldiagramm (Abb. 1) sind die Verantwortlichkeiten leicht zu identifizieren. In diesem Beispiel handelt es sich um sehr einfache Szenarien, deshalb lassen sich die Verantwortlichkeiten direkt aus den Anwendungsfällen übertragen und wie folgt inhaltlich gruppieren:

ABBILDUNG 2. Semantisch gruppierte Verantwortlichkeiten



Für jede Gruppe von Verantwortlichkeiten wird nun im folgenden eine Komponente erstellt.

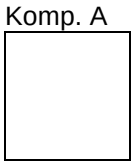
IDENTIFIZIEREN DER UCM-KOMPONENTEN

Die Komponenten der UCM-Notation dienen zur Beschreibung der Struktur eines Systems. Sie werden dabei als unabhängige, miteinander interagierende Einheiten betrachtet. Sie können dabei verschiedenste Dinge repräsentieren [Reiser99]:

- Funktionen, Objekte, Prozesse, Programme (Software),
- Peripheriegeräte, Rechnerteile, Steuerelektronik (Hardware),
- Personen, Objekte aus der realen Welt (Systemumgebung).

Insbesondere können sie auch Elemente aus der Umgebung des Systems enthalten, d.h. die Grenze zwischen System und Umgebung ist in einer UCM nicht ausdrückbar und sollte deshalb explizit in der Dokumentation beschrieben sein[1]. Die Funktio-

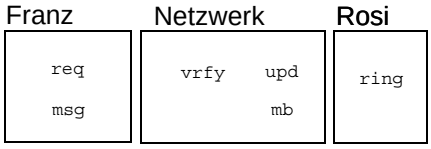
nalität des Systems wird durch Verantwortlichkeiten ausgedrückt, die an einzelne Komponenten gebunden werden können.



UCM-Komponenten werden als Vierecke dargestellt. Am häufigsten wird die als Rechteck dargestellte generische Komponente verwendet. Sie kann in weitere Unterkomponenten aufgeteilt werden, da Komposition von Komponenten nicht nur erlaubt sondern als Abstraktionsmittel sogar erwünscht ist. Der Komponentenbezeichner an der linken oberen Ecke ist die einzige Möglichkeit, eine Komponente zu beschreiben. Dabei ist natürlich eine ausdruckstarke Benennung von Vorteil. Alle zusätzlichen Informationen über eine Komponente sollten sich in der Dokumentation wiederfinden.
Eine Komponente muß nicht zwingend eine Entsprechung im implementierten System haben, womit sich Entwurfsentscheidungen verzögern lassen (d.h. mit welchen Klassen eine Komponente realisiert wird).

Die oben gruppierten Verantwortlichkeiten aus dem Beispiel lassen sich leicht auf drei Komponenten verteilen. Eine Komponente ***Franz*** kann eine `Nummer wählen (req)` und eine `Nachricht erhalten (msg)`. Ein ***Netzwerk*** ist für `Anschluß überprüfen (vrfy)`, `Verbindung herstellen (upd)` und `Nachricht erzeugen (mb)` verantwortlich. Die Komponente ***Rosi*** bemerkt, daß ein `Telefon klingelt (ring)`. Dies ist natürlich nur eine mögliche Systemstruktur.

ABBILDUNG 3. Identifizierte Komponenten mit zugeordneten Verantwortlichkeiten



Diese Form eines UCM-Diagramms nennt sich *"component substrate"*. Sie stellt nur die Systemstruktur mit den zugeordneten Funktionalitäten dar.

UCM-PFADE

Mit den Pfaden der UCM-Notation wird das Verhalten eines Systems modelliert. Sie werden als *"wigly lines"* dargestellt, die einen Startpunkt, verschiedene Verantwortlichkeiten und einen Endpunkt miteinander verbinden. Ein *Startpunkt* wird durch einen gefüllten Kreis dargestellt und repräsentiert eine Vorbedingung bzw. eine auslösende Ursache. Ein *Endpunkt* ist als senkrechter Balken dargestellt und repräsentiert eine Nachbedingung oder resultierende Effekte. Die Bezeichner für Start- und Endpunkte sind wiederum das einzige Mittel der Basisnotation zur Angabe von Vor- bzw. Nachbedingungen. Alle zusätzlichen Angaben sind Aufgabe der Dokumentation.

Pfade bezeichnen nicht den Weg der Information, obwohl dieser meist entlang der Pfade verläuft. Sie sind auch keine Kontrollpfade, sondern stellen eine kausale Abfolge der Verantwortlichkeiten (s.u.) dar. Zusammen mit diesen Verantwortlichkeiten repräsentieren sie einen möglichen Ablauf eines Szenarios. Das bedeutet,

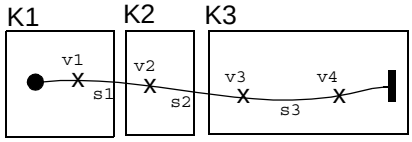
1. Siehe hierzu „Welche Rolle spiel die Dokumentation?“ auf Seite 15

beginnend am Startpunkt wird jede Verantwortlichkeit entlang des Pfades nacheinander abgearbeitet, bis am Endpunkt ein Ergebnis entstanden ist.

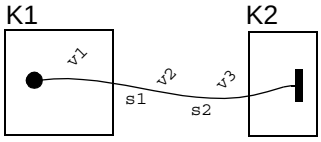
DARSTELLUNG DER UCM-VERANTWORTLICHKEITEN

Einzelne Aktionen, Funktionen oder Teilaufgaben werden in UCMs durch Verantwortlichkeiten repräsentiert. Sind sie einem Pfadsegment zugewiesen, müssen sie zur Erfüllung des entsprechenden Szenarios durchgeführt werden. Eine Verantwortlichkeit bezeichnet dabei nur, daß etwas in diesem bestimmten Szenarioabschnitt von einer Komponenten durchgeführt wird. Was genau eine Verantwortlichkeit beinhaltet, kann nur durch ihren Bezeichner ausgedrückt werden. Damit sei wiederum auf die Dokumentation verwiesen. Auf keinen Fall sollte der Bezeichner einer Verantwortlichkeit beschreiben, wie eine Aufgabe durchgeführt wird.
Pfade werden durch Verantwortlichkeiten in einzelne Pfadsegmente unterteilt, die in den folgenden Abbildungen mit **s1**, **s2**, etc. bezeichnet werden.

Explizite Darstellung. Verantwortlichkeiten lassen sich explizit durch ein Kreuz auf einem Pfad darstellen, wobei sich das Kreuz innerhalb einer Komponente befinden muß. Die Verantwortlichkeit wird damit der Komponente zugeordnet (**v1**, **v2**, ...). Diese Darstellung ist einer der großen Vorzüge von UCMs: die Verbindung von Struktur und Verhalten in einem Diagramm.

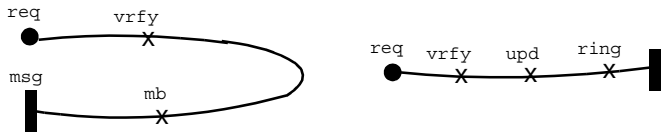


Implizite Darstellung. Häufig sind die Verantwortlichkeiten noch nicht vollständig identifiziert, so daß man sie noch nicht direkt als Kreuz einzeichnen kann. Stattdessen ist immer dann von einer oder mehreren Verantwortlichkeiten auszugehen, wenn ein Pfad eine Komponente scheidet. Möchte man nun eine Verantwortlichkeit genauer festlegen, kann diese durch einen Bezeichner (direkt am Pfad) beschrieben werden (**v1**, **v2**, ...). Um diese Bezeichner von Pfadsegmenten zu unterscheiden, werden sie um 45° gedreht. Ein großer Vorteil dieser Variante ist, daß nicht alle Verantwortlichkeiten aufgeführt werden müssen bzw. zusätzliche existieren können. Häufig werden Mischformen zwischen beiden Notationsvarianten verwendet, so daß der Dokumentation bzw. dem Zusammenhang zu entnehmen sein muß, wann und wo genau von Verantwortlichkeiten auszugehen ist.



Die identifizierten Szenarien aus dem Beispiel lassen sich mit Pfaden und expliziten Verantwortlichkeiten wie folgt darstellen:

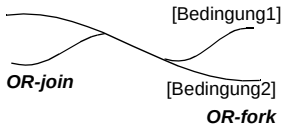
ABBILDUNG 4. In Pfade übertragene Szenarien



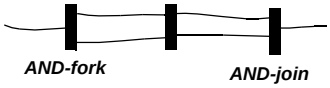
GABELUNGEN UND VERBINDUNGEN VON PFADEN

Häufig stimmen Szenarien in einigen Abschnitten überein. In der UML müssen alle Szenarien einzeln dargestellt werden, so daß auch die gemeinsamen Abschnitte getrennt dargestellt sind. In einer UCM hingegen werden Szenarien als überlap-

pende Pfade repräsentiert, d.h. es gibt gemeinsame und unterschiedliche Pfadsegmente. Durch die Überlappung ist nur ein Pfad zu erkennen, der sich teilt oder verbindet. Tatsächlich ist aber von mehreren Pfaden auszugehen, die gemeinsame Segmente haben.



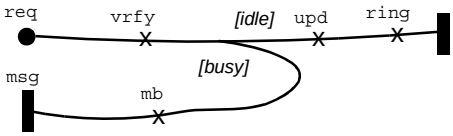
OR-forks und OR-joins. Sollen zwei oder mehrere überlappende Pfadsegmente miteinander verbunden werden, verwendet man einen sogenannten *"OR-join"*. Einen *"OR-fork"* hingegen teilt zwei oder mehrere sich überlappende Pfadsegmente in alternative Segmente. Hierbei kann die Auswahl durch Bedingungen für die Alternativen eingeschränkt werden. Diese Bedingungen werden als Labels in eckigen Klammern repräsentiert.



AND-forks und AND-joins. Zur Darstellung eines Pfades der gleichzeitig auszuführende Segmente besitzt, verwendet man in einer UCM sogenannte *"AND-forks"* und *"AND-joins"*. Beide werden durch vertikale Balken dargestellt. Ein *AND-fork* kennzeichnet den Start der gleichzeitigen Ausführung, ein *AND-join* das Ende. Soll die Ausführung der Segmente synchronisiert werden, so können sie durch einen vertikalen Balken verbunden werden.

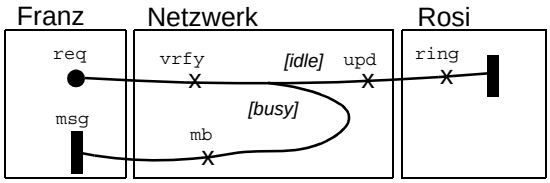
Im obigen Beispiel wurden zwei Pfade identifiziert, die in den ersten beiden Verantwortlichkeiten übereinstimmen (Abb. 4). In einer UCM werden die beiden Pfade überlappend dargestellt, wobei dieser Verzweigung aus den Szenarien abgeleitete Bedingungen (Anschluß frei/besetzt) zugeordnet werden können:

ABBILDUNG 5. Identifizierte Pfade mit bedingter Pfadgabelung



Wird die oben festgelegte Komponentenstruktur mit den Pfaden verknüpft, erhält man eine integrierte Darstellung von Struktur und Verhalten für das Telefonsystem, auch *"bound map"* genannt:

ABBILDUNG 6. An die Komponentenstruktur gebundene Pfade



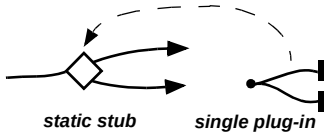
Interessante Erweiterungen der Basisnotation

Die oben eingeführte Basisnotation führt für komplexere Anwendungen häufig zu eher unübersichtlichen Diagrammen. Daher bieten UCMs zusätzlich Notationsmittel zur Komposition und Dekomposition von Informationen.

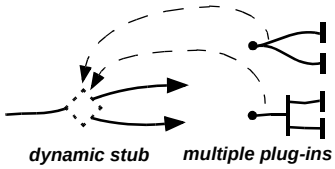
(DE)KOMPOSITION VON UCMS

Zur Verbesserung der Übersichtlichkeit einer UCM lassen sich einzelne Teile in Unterdiagramme - sogenannte *"submaps"* - auslagern. Der Mechanismus zum Definieren und Strukturieren von Unterdiagrammen ist das wichtigste Strukturierungsmittel, um UCMs auch bei sehr komplexen Systemen einfach und übersichtlich zu halten.
Dazu wird die Basisnotation um zwei Elemente erweitert: Platzhalter im Diagramm (genannt *stubs*) repräsentieren die Teildiagramme (genannt *plug-ins*). Dabei kann ein *plug-in* im selben Diagramm oder in einem anderen Diagramm definiert sein. *Stubs* lassen sich in zwei Arten unterteilen: Statische und Dynamische.

Statische Stubs. Ein sogenannter *"static stub"* wird durch eine durchgezogene Raute dargestellt. Er repräsentiert maximal ein *plug-in* und dient hauptsächlich zur hierarchischen Dekomposition von komplexen Diagrammen, indem Teile einer UCM in ein Unterdiagramm ausgelagert werden.



Dynamische Stubs. Soll ein *stub* verschiedene *plug-ins* enthalten können, z.B. um zwischen unterschiedlichen Funktionalitäten kontextabhängig auszuwählen, wird ein *"dynamic stub"* verwendet. Dieser ist als gestrichelte Raute dargestellt. Die Auswahl kann dabei zur Laufzeit durch eine beliebige Auswahlstrategie (oft als Vorbedingung beschrieben) bestimmt werden. Es können auch mehrere *plug-ins* gleichzeitig ausgewählt werden, die parallel oder sequentiell ausgeführt werden können.



Die Verbindung eines ausgewählten *plug-ins* mit einem (statischen oder dynamischen) *stub* wird durch ein sogenanntes *"binding"* beschrieben.
Ein *stub* besitzt eingehende und ausgehende Pfadsegmente. Ein *binding* definiert genau, welcher Ein- bzw. Ausgang eines *stubs* mit welchem Start- bzw. Endpunkt eines *plug-ins* verbunden werden soll. Dies kann wie folgt notiert werden:

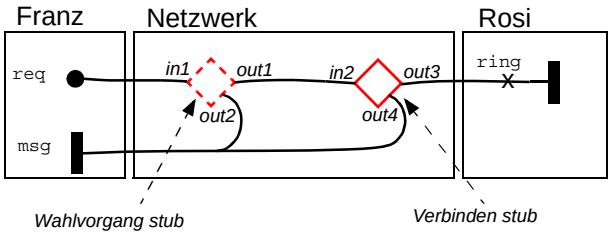
```
<stub_incoming_segment, plug-in_start_point>
<stub_outgoing_segment, plug-in_end_point>
```

Für einen *dynamic stub* ist für jedes *plug-in* ein *binding* zu definieren.
Bindings können sowohl in der Dokumentation als auch im Diagramm enthalten sein.

HINZUFÜGEN VON NEUEN ANFORDERUNGEN

Ausgehend vom "obersten" Diagramm, welches auch als *"root map"* bezeichnet wird, ist an allen Stellen auf einem Pfad ein *stub* einzufügen, an denen zusätzliche Funktionalität benötigt wird.
Das Telefonsystem aus dem obigen Beispiel soll beispielsweise um eine Rufzeichenanzeige und eine Nummernsperre erweitert werden. Dazu wird in die *root map* ein *Verbinden stub* und ein *Wahlvorgang stub* eingefügt:

ABBILDUNG 7. Um *stubs* erweiterte *root map*



Der *stub* für den veränderten Wahlvorgang ist dynamisch gewählt, um die Aktivierung der Nummernsperre durch zwei verschiedene *plug-ins* zu modellieren:

Überprüfung auf Wahlerlaubnis. Das Telefonsystem bietet die Möglichkeit zu überprüfen, ob eine Nummer gewählt werden darf. Zum Ein- bzw. Ausschalten der Überprüfung werden also zwei *plug-ins* benötigt: ein *Standardwahl plug-in* und ein *Originating Call Screening (OCS) plug-in*. Das *Standard plug-in* verbindet lediglich den Eingang direkt mit dem Ausgang und führt also keine Überprüfung durch. Das *OCS plug-in* führt hingegen eine neue Komponente ein. Die als *OCSList* bezeichnete Datenstruktur enthält alle verbotenen Nummern. Solche Komponenten, die keine Funktionalität enthalten, werden als "*passive Objekte*" bezeichnet (siehe Tabelle 1). Ist die Überprüfung aktiviert (das *OCS plug-in* ausgewählt), wird die Nummer vor dem Wählen zuerst überprüft (`chk`). Ist die Nummer verboten (in *OCSList* enthalten), wird eine entsprechende Nachricht (`msg`) erzeugt und an den Benutzer gesendet. Andernfalls wird der Wahlvorgang fortgesetzt.

ABBILDUNG 8. Standard plug-in

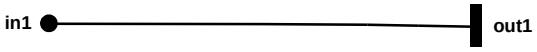
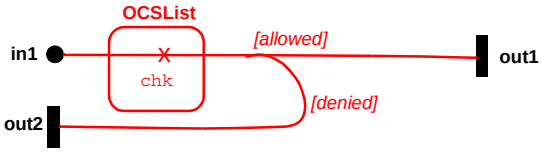
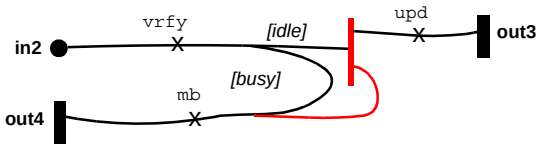


ABBILDUNG 9. OCS plug-in



Rufzeichen bei erfolgreicher Verbindung. Das Erfolgsszenario ist noch unvollständig, da ***Franz*** als Anrufer nicht bemerkt, wenn eine Verbindung zustandegekommen ist. Die Durchführung des Wahlvorganges wird in das *Verbinden plug-in* ausgelagert. Es überprüft, ob der gewählte Anschluß frei oder besetzt ist. Ist er frei, wird eine Verbindung hergestellt (`upd`) und außerdem ein Rufzeichen beim Anrufer erzeugt (`mb`). Ist der Anschluß hingegen besetzt, wird beim Anrufer ein Besetztzeichen erzeugt (`mb`).

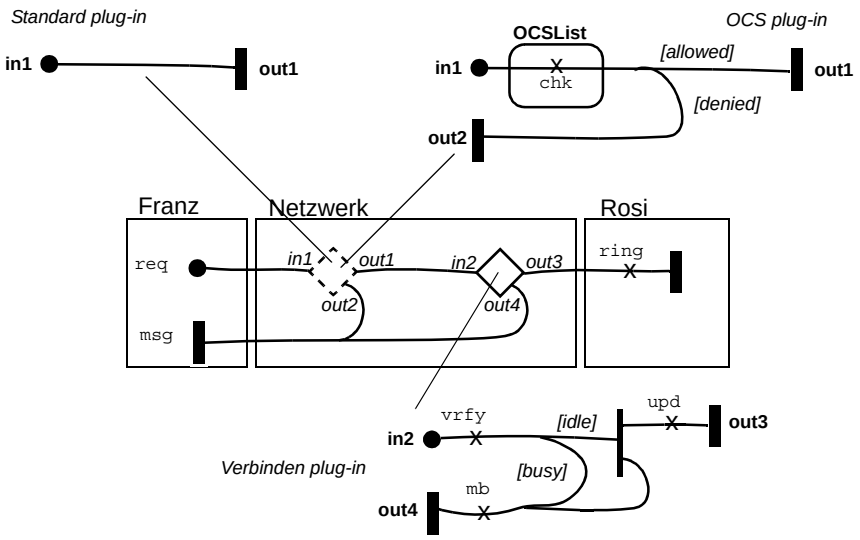
ABBILDUNG 10. Verbinden plug-in



Geht man nun ein Szenario durch, werden die *bindings* ausgewertet, indem die Ein- bzw. Ausgänge der *stubs* mit den Start- bzw. Endpunkten der zugehörigen *plug-ins* identifiziert werden. Für das *Standard plug-in* aus dem Beispiel hieße das, daß der Startpunkt ***in1*** des *plug-ins* dem hineinkommenden Pfadsegment ***in1*** des *stubs* zugeordnet wird. Analog dazu wird der Endpunkt ***out1*** an das Ausgangssegment ***out1*** gebunden. Wie hier gut zu sehen ist, muß ein *plug-in* nicht zwingend für alle Ein- und Ausgänge eines *stubs* auch Start- und Endpunkte haben. Wichtig ist nur, daß zumindest eine Verbindung zustande kommt.

Durch das Einführen von *stubs* lassen sich UCM-Diagramme stark vereinfachen, obwohl sie immer noch alle möglichen Szenarien enthalten. Sind *stubs* an geeigneten Stellen definiert, ist es wie im Beispiel sehr einfach, neue Funktionalitäten durch neue *plug-ins* hinzuzufügen. Existierende Diagramme können so mit neuen Pfaden erweitert und neuen statischen bzw. dynamischen *stubs* zerlegt werden.

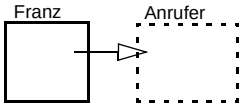
ABBILDUNG 11. Die erweiterte *root map* mit allen Unterdiagrammen



DARSTELLUNG VON BESONDEREN INTERAKTIONEN

Zusätzlich gibt es für fast jede besondere Interaktion zwischen Pfaden eine einfache Erweiterung der Basisnotation: u.a. für synchrone oder asynchrone Interaktionen, Timer, Abbruch, Fehlerpunkte oder sogar für gemeinsame Verantwortlichkeiten. Eine sehr kompakte Übersicht über so ziemlich alle Elemente der UCM-Notation ist im [Tutorial] vorhanden.

GENERALISIERUNG VON UCMS



Ebenso wichtig wie das Hinzufügen von Funktionalität ist auch das Verallgemeinern von Szenarien oder Komponenten. Die konkreten Komponenteninstanzen ***Franz*** und ***Rosi*** aus dem Beispiel sollen z.B. durch die allgemeineren Komponenten ***Anrufer*** und ***Angerufener*** ersetzt werden. Ein solcher Komponententyp, der mit unterschiedlichen Komponenteninstanzen assoziiert werden kann, wird in einer UCM durch *"slots"* repräsentiert.

KOMPONENTENNOTATION

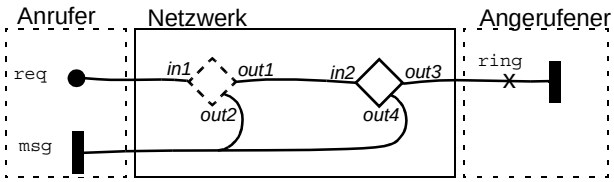
In UCMs sind beliebige Komponententypen benutzbar und anhand der Notation auch erkennbar. Hierfür empfiehlt Buhr einen eigenen Katalog von Komponententypen mit zugehöriger Notation [BuCa96]. Die wichtigsten Typen sind in Tabelle 1 aufgeführt. Prinzipiell können in UCMs jedoch beliebige Komponententypen und -notationen wie z.B. UML-Components verwendet werden.

TABELLE 1. Komponententypen

Typ	Beschreibung	Notation
team	Dargestellt als Vierecke, repräsentieren sie Komponenten von jedem Typ. Sie werden als generische Standardkomponente am häufigsten verwendet.	
slot	Dargestellt als gestrichelte Vierecke, können sie verschiedene Instanzen aufnehmen.	
process	Dargestellt als Parallelogramme, repräsentieren sie aktive Komponenten.	
object	Dargestellt als abgerundete Vierecke, repräsentieren sie passive Komponenten (z.B. Datenstrukturen)	

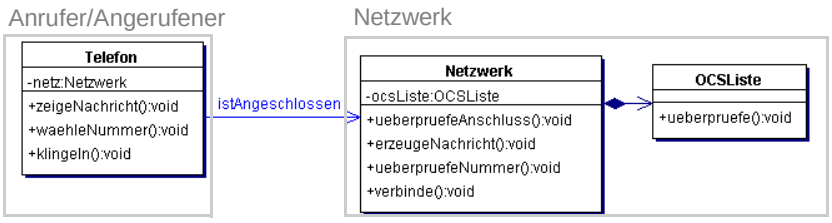
Die verallgemeinerte *root map* für das Telefonsystem, würde wie in der folgenden Abbildung aussehen:

ABBILDUNG 12. Die verallgemeinerte *root map*



Um die dargestellte UCM zu implementieren, werden zu Diagrammbestandteilen UML-Diagramme erstellt. Eine UCM-Komponente wird auf eine Menge von Klassen (in UML-Klassendiagrammen) abgebildet. Dabei werden die Verantwortlichkeiten der Komponente auf die Klassen verteilt und mittels Interaktionsdiagramme (UML Sequenz-/Objektinteraktionsdiagramme) ausmodelliert. Die Pfade werden genauso mit Interaktionsdiagrammen modelliert, wobei hier aber der Fokus auf den Interaktionen zwischen den Komponenten liegt. Die dadurch entstandenen implementationsnahen UML-Diagramme werden implementiert. Die Umsetzung von UCMs in verschiedene UML-Diagramme wird ausführlich in [Amyot99] beschrieben.

ABBILDUNG 13. Ein mögliches Klassendiagramm für das Beispiel



Wie werden UCM-Diagramme gelesen?

Aus einer UCM können viele Informationen über ein System abgeleitet werden. Im folgenden sind die wichtigsten Interpretationsmöglichkeiten kurz zusammengefaßt.

Die **Gesamtheit der Komponenten** ergibt die Struktur des Systems.
Die **Gesamtheit der Szenarien**, die durch die Pfade mit ihren Verantwortlichkeiten beschrieben sind, ergibt das Verhalten des Systems.
Die **Verantwortlichkeiten innerhalb einer Komponente** beschreiben die Funktionalität dieser Komponente. Folgen dabei unmittelbar mehrere Verantwortlichkeiten aufeinander, muß die Komponente selbst den korrekten Ablauf der sequentiellen Abarbeitung gewährleisten.
Kreuzt ein Pfad zwei Komponenten, ist eine wie auch immer geartete, nicht notwendigerweise direkte **Kommunikation zwischen den Komponenten** nötig. Hierbei löst die erste Komponente die Abarbeitung der folgenden Verantwortlichkeiten in der zweiten Komponente aus. Jedoch wird lediglich die Existenz einer Kommunikation festgelegt, nicht jedoch die Art und Weise der Kommunikation.

Durch die besonders hohe Abstraktion sollen UCMs bei der **Auswahl eines geeigneten Architekturstils** eine besondere Hilfestellung bieten. Einige Anhaltspunkte für diese Auswahl ergeben sich aus der Verbindung von Struktur und Verhalten. Es ist leicht zu sehen, welche Teilstrukturen welche Funktionalitäten übernehmen. Zusätzlich sind die UCM-Modelle sehr übersichtlich und erleichtern durch die vollständige Wiederverwendbarkeit der Szenarien viele Experimente mit unterschiedlichen Architekturen.

Wofür lassen sich Use Case Maps verwenden?

INTEGRATION VON SYSTEMANFORDERUNGEN

UCMs eignen sich besonders zur Integration vieler Anwendungsfälle. Bei der Konstruktion einer UCM, werden die zugehörigen Szenarien miteinander in Beziehung gesetzt, so daß Probleme wie Unvollständigkeit, Inkorrektheit, Inkonsistenz, unterschiedliche Abstraktionsniveaus oder Zweideutigkeiten leicht zu erkennen sind. UCMs haben bei diesem Integrationsprozeß gegenüber Entwurfsszenarien aus der UML außerdem den Vorteil, das die Szenarien ohne Definition von Nachrichtenaustausch zwischen den Komponenten vollständig modelliert werden können. So sind nicht nur bestimmte sondern auch allgemeine Szenarien modellierbar.

FESTLEGEN DER SYSTEMSTRUKTUR

Soll beim Systementwurf zuerst die Systemstruktur festgelegt werden, ermöglichen UCMs die Darstellung von Komponenten und ihren Verantwortlichkeiten auf oberster Entwurfsebene ("*component substrate*"). Anschließend können einzelne Szenarien (unter Verwendung der internen Verantwortlichkeiten) durch einzeichnen von Pfaden auf der Systemstruktur „ausprobiert" werden. So lassen sich unterschiedliche Systemstrukturen und Pfadkonstellationen auf eine einfache Art und Weise experimentell überprüfen. Durch die vollständige Wiederverwendbarkeit der Pfade ist der Aufwand bei solchen Experimenten deutlich geringer als bei Entwurfsszenarien.
Die Schnittstellen einer Komponente sind durch die Unterscheidung von komponenteninternen bzw. nach außen sichtbaren ("*observable*") Verantwortlichkeiten inhaltlich leicht zu identifizieren.

KONTINUIERLICHE INTEGRATION VON FUNKTIONALITÄTEN

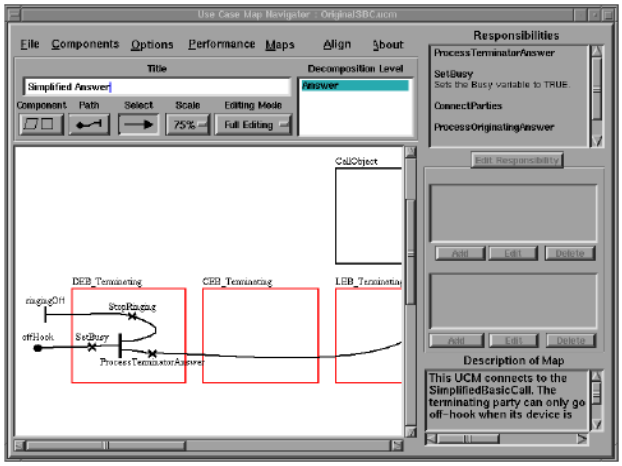
Eine überlegte Platzierung von *stubs* auf den Pfaden, ermöglicht insbesondere unter Verwendung von *dynamischen stubs* eine leichte Integration von weiteren Funktionalitäten mittels neuer *plug-ins*.

FORMALE VALIDIERUNG UND VERIFIKATION

Neben der graphischen Notation besitzen UCMs eine semi-formale Semantik. Diese kann verwendet werden, um formalere Modelle und Spezifikationen, wie z.B. LOTOS, SDL, etc., aus UCMs zu generieren. Eine UCM kann somit von Hand oder automatisiert (bei werkzeugunterstütztem Einsatz) in Modelle übertragen werden, die ein schnelles Protoyping bzw. eine Validierung erlauben [AmBuGr99]. Zusätzlich existieren Abbildungen nach hierarchisch strukturierten Zustandsautomaten (wie z.B. in UML-RT verwendet). Mittels der semi-formalen Semantik ist es auch möglich, Testfälle zur Validierung eines Systementwurfs zu generieren, z.B. im Sinne von Vollständigkeit gegen funktionale Tests.

Werkzeugunterstützung. Die textuelle Repräsentation von UCMs ist in XML definiert. Dies wird für unterschiedliche Werkzeuge u.a. als Eingabeformat und zur Generierung der Dokumentation verwendet. Das Werkzeug mit der wohl vollständigsten Unterstützung von UCMs ist der *UCM-Navigator*. Mit ihm ist es möglich, UCMs zu erzeugen, zu editieren und auf Korrektheit zu überprüfen [Tutorial]. Er unterstützt die Pfadnotation, Buhr's Komponentennotation, XML als Dateiformat, eingebettete *stubs* und *plug-ins*, sehr leichtes Zuordnen von Verantwortlichkeiten zu Komponenten, Notationserweiterungen für Agentensysteme und Performancemodellierung und automatische Dokumentationserzeugung in PDF und Postscript. Verfügbar ist der *UCM-Navigator* für Linux, Solaris, HP-UX und Windows NT.

ABBILDUNG 14. Screenshot des UCM-Navigators



Welche Rolle spielt die Dokumentation?

Die Ausdrucksstärke der UCM-Notation ist zugunsten der Einfachheit und Übersichtlichkeit stark beschränkt. Ein großer Teil der Systemeigenschaften, insbesondere viele Details, können nicht in den UCM-Diagrammen wiedergegeben werden. Diese zusätzlichen Informationen müssen in der Dokumentation enthalten sein. Obwohl sie nicht Bestandteil der UCM-Notation ist, hat die Dokumentation des Systems für UCMs daher eine größere Bedeutung als bei anderen Modellierungstechniken.

Was sollte enthalten sein? Besonders wesentlich sind Systemeigenschaften, die mit der UCM-Notation nicht darstellbar sind. So muß z.B. aus der Dokumentation eindeutig die Systemgrenze hervorgehen, ob Parallelverarbeitung möglich ist oder nicht und welche Variante zur Darstellung von Verantwortlichkeiten (explizit/implizit) gewählt wurde. Eine gute Dokumentation beschreibt darüber hinaus die Komponenten, Verantwortlichkeiten und Szenarien. Hierbei sollte besonders die Intention und die Beziehungen der einzelnen Elemente erläutert werden, wie z.B. die Komponente Netzwerk kann folgende Subkomponenten enthalten.

Wann sollten Use Case Maps im Software-Entwicklungsprozeß eingesetzt werden?

Im allgemeinen werden Modellierungstechniken überwiegend in der Entwurfsphase eines Softwareentwicklungsprozesses eingesetzt, mit dem Ziel, ein System so zu beschreiben, daß es anschließend direkt implementiert werden kann. Bei UCMs wurde zugunsten der Übersichtlichkeit ein Abstraktionslevel gewählt, welches eine direkte Implementierung nicht möglich macht. Die sich daraus ergebene unkomplizierte Notation und Übersichtlichkeit der Modelle eignen sich dafür hervorragend als Kommunikationsmittel auf Papier oder Whiteboards und zur “first step”-Modellierung.

UCMS ALS REINES KOMMUNIKATIONSMITTEL

Während die mündliche bzw. schriftliche Kommunikation weiter im Vordergrund steht, sollen UCM-Diagramme das Gesagte bzw. Geschriebene veranschaulichen. Sie sind dabei lediglich für den Augenblick bestimmt, auch wenn z.B. ein Entwickler sie aufhebt, um sich später an ein Gespräch zu erinnern. Sie sollten jedoch nicht verwendet werden, um sich oder andere an das Dargestellte zu binden, wie z.B. bei Spezifikationen üblich. Im Gegenteil, UCMs bieten durch ihre abstrakte Darstellung von Informationen eher eine Art Experimentiergrundlage.
Durch die folgenden Eigenschaften eignen sich UCMs besonders als Kommunikationsmittel:

- UCM-Diagramme bieten einen Überblick über das gesamte System, ein sogenanntes *“big picture”*
- Durch die hohe Abstraktion und Flexibilität ist es möglich, uninteressante Entwurfsentscheidungen zurückzustellen.
- Kein Zwang zum Detail.

UCMS ALS “FIRST STEP” - MODELLIERUNG

Möchte man noch einen Schritt weiter gehen, können UCM-Diagramme auch als Spezifikation eingesetzt werden. Jedoch nicht als implementierungsfähige Beschreibung, sondern eher als sehr abstrakte Spezifikation. Klassische Modellierungstechniken werden dadurch nicht überflüssig, sondern durch UCMs unterstützt, so daß sich Modellverfeinerungen leichter ableiten lassen.

UCMs in der Designphase. In einem ersten Schritt wird das System mit UCMs modelliert. Auf der Grundlage dieses abstrakten und übersichtlichen Modells wird das System mit herkömmlichen Diagrammen detaillierter beschrieben. Diese können anschließend implementiert werden.

UCMs in anderen Entwicklungsphasen. In anderen Entwicklungsphasen lassen sich z.B. neue Anforderungen bei der Erstellung von UCMs identifizieren oder Testfälle aus den Szenarien ableiten. Auf jeden Fall können alle Phasen eines Softwareentwicklungsprozesses von der Übersichtlichkeit der UCMs profitieren.
UCMs lassen sich mit allen anderen Entwicklungsmethoden verbinden, wobei das Hauptaugenmerk immer auf der anderen Methode bleiben sollte. Sämtliche Mischformen sind hierbei nicht nur erlaubt, sondern sogar erwünscht.

Ein Fazit aus der Sicht des Software Engineering

UCM haben viele nützliche Eigenschaften und sind bei einer noch ungenauen Vorstellung von einem System deutlich einfacher anzuwenden als diverse UML Diagramme. Sie bieten selbst bei sehr großen Systemen eine einfache Darstellung auf hohem Abstraktionsniveau. Die nicht explizit darstellbaren Informationen der UML-Diagramme, wie z.B. die Verantwortlichkeiten einer Komponente, lassen sich im Kontext des Gesamtverhaltens explizit in einem Diagramm darstellen. Diese gemeinsame Darstellung von Struktur und Verhalten hat neben der besseren Verständlichkeit auch Vorteile bei der Modellverfeinerung, so ergeben sich z.B. Schnittstellen zwischen Komponenten direkt aus den Pfadübergängen.
UCMs haben ein sehr variables Abstraktionsniveau, d.h. fast beliebig abstrakte Darstellungen sind im Gegensatz zu UML-Diagrammen leicht möglich. Die Darstellung der Details wird dabei anderen Diagrammen bzw. der Dokumentation überlassen. Zusätzlich wird das Zurückstellen von Designentscheidungen, durch *stubs* und *plug-ins* besonders unterstützt.

Verwendet man UCMs in einem evolutionären Softwareentwicklungsprozeß als Spezifikation, sind sie ein weiteres Modell, das konsistent gehalten und aktualisiert werden muß. Dieser zusätzliche Aufwand lohnt sich insbesondere nur dann, wenn die Komplexität des zu entwickelnden Systems dies rechtfertigt. Im Gegensatz dazu sind UCMs als Kommunikationsmittel bzw. als Unterstützung auf dem Weg zum implementierbaren Modell durch vernachlässigbaren Aufwand in jedem Fall empfehlenswert.

Anhang A: *Use Case Maps und UML-Aktivitätsdiagramme*

Bisher wurde eine wesentliche Diagrammart der UML, das Aktivitätsdiagramm (*activity diagram*), noch nicht weiter betrachtet. UCMs und Aktivitätsdiagramme teilen viele Konzepte und sogar Notationselemente. Im folgenden sollen kurz UCMs von Aktivitätsdiagrammen abgegrenzt werden. Eine deutlich ausführlichere Betrachtung der Beziehungen zwischen UCMs und UML-Diagrammen ist in [AmMu00] und [Amyot99] zu finden.

Aktivitätsdiagramme sind im allgemeinen ein spezieller Fall eines Zustandsübergangsdiagramms. Im Gegensatz zu “normalen” Zustandsübergangsdiagrammen (*statecharts*) werden jedoch nicht die externen Ereignisse, sondern der Kontrollfluß durch interne Operationen betrachtet. Sie eigenen sich sehr gut, um einen Zustandsautomaten für Prozedurabläufe oder Geschäftsprozesse zu modellieren.
Ein Aktivitätsdiagramm enthält im wesentlichen Abfolgen von Aktionen und Bedingungen, unter denen die Aktionen ausgeführt werden. Eine Zuordnung von Aktionen zu Komponenten steht dabei nicht im Vordergrund.

GEMEINSAMKEITEN MIT DER UCM-BASISNOTATION

Beide Notationen unterstützen Abfolgen von Aktionen, sowie Alternativen und Gleichzeitigkeit. Auch Start- und Endpunkte haben den selben Zweck. Die Verantwortlichkeiten in den UCMs sind den Aktivitäten sehr ähnlich. Eine komplexe Aktivität kann in einem weiteren Aktivitätsdiagramm verfeinert werden, wie auch mit statischer Pfaddekomposition (mittels *static stubs*) bei UCMs möglich.

VORZÜGE DER UCMS NOTATION

Das Konzept der *stubs* erscheint jedoch etwas allgemeiner. So erlauben sie mehrere ein- und ausgehende Pfade. *Dynamische stubs* ermöglichen zusätzlich die Verwendung mehrerer *plug-ins* (als Verfeinerungen), welche durch eine Selektionsstrategie ausgewählt werden. *Stubs* ermöglichen somit nicht nur das Darstellen von dynamischen Verhalten, sondern können auch komplexe Systemstrukturen überschaubarer darstellen.

Identifikation von Komponenten. Die eigentliche Stärke von UCMs ist die Zuordnung von Verantwortlichkeiten auf Komponenten. Aktivitätsdiagramme werden in der Regel dafür nicht verwendet, obwohl dies auch begrenzt möglich wäre [Amyot99]. So kann ein Aktivitätsdiagramm visuell (durch vertikale Linien) in Schwimmbahnen unterteilt werden. Jede Schwimmbahn repräsentiert eine Verantwortlichkeit für einen Teil einer übergeordneten Aktivität und kann durch ein oder mehrere Objekte implementiert werden. Hierbei ist jede Aktion an eine Schwimmbahn gebunden und Transaktionen können Schwimmbahnen kreuzen. Es ist also möglich mittels Schwimmbahnen einfache Komponenten zu identifizieren, wenn auch nicht so konkret wie in einer UCM. Diese Komponenten sind jedoch nur sehr

einfach und eindimensional. Beziehungen zwischen Komponenten (z.B. durch ihre relative Position, ihren eigenen Charakter) sind nicht explizit darstellbar. UCMs hingegen stellen Komponenten mit ihren Verantwortlichkeiten integriert dar und zeigen dabei wie die Komponentenstruktur zur Laufzeit beeinflußt bzw. verändert wird.

Literaturempfehlungen

TABELLE 2. Referenzen und Empfehlungen

Abkürzung	Beschreibung
[Amyot99]	Daniel Amyot, *Use Case Maps and UML for Complex Software-Driven Systems*, UML'99. Besonders anschaulich wird dargestellt, wie die Konzepte der UML durch UCMs ergänzt werden können. Zu dem sind die Beziehungen zu den einzelnen Diagrammarten in der UML ausführlich beschrieben. http://www.usecasemaps.org/pub/uml99.pdf*
[AmMu00]	Daniel Amyot und Gunter Mussbacher, *On the Extension of UML with Use Case Maps Concepts*, University of Ottawa, Mitel Corporation, UML 2000. Die Erweiterung der UML Konzepte durch UCMs wird motiviert, beschrieben und in UML-Diagrammen veranschaulicht. Es werden insbesondere einzelne Teilelmente miteinander verglichen. http://www.usecasemaps.org/pub/uml2000.pdf*
[Tutorial]	Daniel Amyot, *Use Case Maps - Quick Tutorial*, University of Ottawa, 1999. Die Basiselemente und einige Erweiterungen von UCMs werden erläutert und anschaulich verwendet. Im Anhang ist eine sehr komprimierte Übersicht über die meisten Notationselemente enhalten. http://www.usecasemaps.org/pub/UCMtutorial/UCMtutorial.pdf*
[Reiser99]	Oliver Reiser, *Use Case Maps*, Technische Universität Berlin, 1999. Ein gelungener Überblick über die Konzepte und Notationselemente der UCMs. Durch Konzentration auf die Basiskonzepte sehr einfach gehalten, besonders zum Einstieg zu empfehlen.
[AmBuGr99]	D. Amyot, L. Logrippo, R.J.A. Buhr und T. Gray, *Use Case Maps for the Capture and Validation of Distributed Systems Requirements*, 1999. Ein Einstieg in die Aufnahme von Anforderungen mittels UCMs mit gleichzeitiger Erzeugung von formalen Spezifikationen in LOTOS zu ihrer Überprüfung. http://www.usecasemaps.org/pub/re99.pdf*
[BuCa96]	R.J.A Buhr und R.S. Casselman, *Use Case Maps for Object-Oriented Systems*, Prentice Hall, 1996. Im diesem Buch sind alle UCM-Konzepte ausführlich beschrieben, wenn z.T. auch sehr allgemein. Es ist gut als Nachschlagewerk verwendbar. http://www.usecasemaps.org/pub/UCM_book95.pdf*

TABELLE 2. Referenzen und Empfehlungen

Abkürzung	Beschreibung
[Buhr98]	R.J.A. Buhr, *Use Case Maps as Architectural Entities for Complex Systems*, Carleton University Ottawa, 1998. Zusätzlich zur Basisnotation und Verwendung von UCMs werden viele erweiterte Notationselemente der UCMs sehr plastisch erläutert. http://www.usecasemaps.org/pub/ucmupdate.pdf*
[OMG00]	Object Management Group, Unified Modeling Language Specification, Version 1.3 Die Spezifikation der UML in der Version 1.3. ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf*

* letzter Zugriff am 12.02.2001