Einführung in das Kalkül und die modale Logik von Mobile Ambients

bearbeitet von Ralf Gitzel
im Rahmen des Seminars "Ubiquitous Computing" am
Lehrstuhl PI4 (WS 2001-2002)

gitzel@wifo3.uni-mannheim.de

1. Einleitung

Die IT-Branche hat in den letzten Jahren eine enorme Expansion durchgemacht, die zu einer Verbreitung von Computern geführt hat, die man sich in den Anfangszeiten der Geschichte der Informatik nie in solcher Form vorgestellt hätte.

Leider bringt dieser 'Boom' ein Problem mit sich - Funktionalität steht im Vordergrund, Eigenschaften, die nicht sofort ins Auge fallen werden vernachlässigt. Das beste Beispiel hierfür ist sicherlich Microsoft, deren bisher laxe Einstellung zu Sicherheit in ihren Produkten allseits bekannt ist [1].

Mobile Ambients[1] sind ein Werkzeug, das zur Verbesserung der Sicherheit von verteilten Anwendungen, insbesondere solcher mit mobilem Code, verwendet werden kann. Es handelt sich um ein mathematisches Modell, bestehend aus einem Kalkül und einer Logik, mit dem Systeme modelliert und bezüglich ihrer Eigenschaften untersucht werden können.

Im Folgenden wollen wir uns näher mit den Mobile Ambients befassen. Der zweite Abschnitt beschreibt das Kalkül und die Modellierungsmöglichkeiten, welche sich aus ihm ergeben. Im Anschluss wird die darauf aufsetzende modale Logik erläutert und mit einem Beispiel illustriert.

Bislang gibt er nur relativ wenig Fälle eines Einsatzes von Mobile Ambients, z.B. [2]. Woran das liegen könnte wird am Ende der Arbeit erläutert.

2. Grundkonzept

Wie bereits angedeutet kann man mit dem Kalkül von Mobile Ambients verteilte Systeme mit mobilen Komponenten modellieren. In den Worten von Cardelli und Gordon, den Entwicklern der Mobile Ambients, dient das Kalkül dazu, "Mobilität zu untersuchen, die als Veränderung der räumlichen Konfiguration im Zeitablauf gesehen werden kann." ([3], Seite 1). Hierbei unterscheiden sie zwei Formen von Mobilität in verteilten Anwendungen:

- Mobile Computing: *Laptops, Mobiltelefone etc.*
- Mobile Computation: *Software-Agenten*

Somit bezieht sich Mobilität entweder auf die Hardware oder auf die Software.

Eine Kernanwendung für diese Modellierung ist sicherlich die Überprüfung von System-Sicherheit. Cardelli und Gordon geben ein Beispiel für die Anwendung des Kalküls in Zusammenhang mit Sicherheit (nämlich für Firewalls auf Seite 8 in [3]). Wenn man sich näher mit den Elementen des Kalküls befasst gewinnt man den Eindruck, das man besser als Sicherheit im Allgemeinen, sondern Sicherheits*konzepte* überprüfen kann. Für den Test der technischen Realisierung erscheinen Mobile Ambients eher ungeeignet - dies soll in Abschnitt 6 nochmals genauer im Detail erläutert werden.

3. Das Mobile Ambients Kalkül

Das Mobile Ambients Kalkül kann zur Modellierung einer Vielzahl von Technologien verwendet werden. Cardelli und Gordon nennen als Beispiele Java, Obliq, Telescript und Linda.

3.1. Grundelemente

Abb. 1 zeigt die Grundelemente des Kalküls in graphischer Form. Da Mobile Ambients eine rein abstrakte Darstellungsweise von verteilten Systemen darstellt, werden im Laufe dieser Arbeit viele Konzepte durch Schaubilder illustriert, um das Verständnis zu fördern.

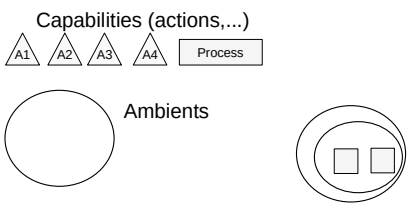

Abb. 1: Elements of Distributed Computing

Das Kalkül besteht aus den folgenden Elementen:

- Ambients
- Prozesse
- Capabilities

Aus diesen Grundelementen lassen sich die unterschiedlichsten Systeme modellieren.

Ein **Ambient** ist "a bounded place where computation happens" [3]. *Bounded* bedeutet in diesem Zusammenhang, dass das Ambient klar von anderen abgegrenzt ist und diese Trennung nicht automatisch überwunden werden kann. Beispiele für Ambients wären File-Systeme, ein Virtueller Adressraum, Laptops usw.

Ambients, in Abb. 1 als Kreise dargestellt, haben außerdem noch die folgenden Eigenschaften:

- Sie können verschachtelt werden.
- Sie können (in der Verschachtelungs-Hierarchie) bewegt werden
- Sie haben immer einen Namen

[1] Ironischerweise wurden *Mobile Ambients* von Mitgliedern von Microsoft Research entwickelt.

Ein Ambient wird im Kalkül wie folgt dargestellt:

$$n[P]$$

ist ein Ambient, das einen Prozess P enthält.

Ein **Prozess** wird von Cardelli und Gordon nicht verbal definiert sondern lediglich im Bezug auf seine Eigenschaften im Rahmen des Kalküls. Aus dem Kontext der Arbeit lässt sich jedoch erschließen, dass der Begriff Prozess für eine Abfolge von Befehlen steht, die evtl. in einem Ambient gekapselt sind. Prozesse werden im Kalkül als Großbuchstaben dargestellt, in Abb. 1 sind sie durch Rechtecke repräsentiert. Sie haben die folgenden Eigenschaften:

- Man kann einem Prozess, *capabilites* "vorschalten"
- Ein Prozess kann aus mehreren parallelen Prozessen bestehen
- Ein Prozess kann aus einem Ambient bestehen

Capabilities (in Abb. 1 als Dreiecke dargestellt) sind unter anderem Operationen, welche die "hierarchische Struktur" [3] des Systems verändern. Auch Kommunikation zwischen Prozessen wird durch *capabilities* modelliert (s. Abschnitt 3.3).

Sie haben die folgenden Eigenschaften:

- Die häufigsten *capabilities* sind: *in n*, *out n*, *open n*, wobei *n* ein Name ist.
- *in n*: kann *n* betreten, wobei das den Prozess umgebende Ambient mit in *n* gezogen wird (s. Abb. 2)
- *out n*: kann n verlassen, wobei wieder das umgebende Ambient mitbewegt wird
- *open n*: kann n von außen öffnen**,** d.h. *n* existiert nach dieser Operation nicht mehr, sein Inhalt bleibt aber unversehrt.

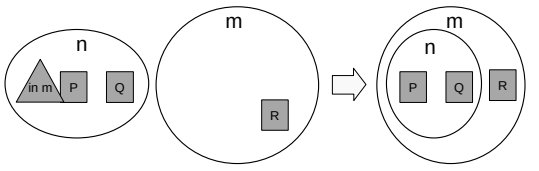

Abb. 2: Using the *in*-Capability

In diesem Zusammenhang ist eine kleine Anmerkung zu den Namen in diesem Kontext angebracht. Es ist nämlich ausschließlich das Wissen über den Namen eines Ambients das seine Manipulation (über *capabilities*) möglich macht. Besonders im Zusammenhang mit Sicherheitsfragen sind folgende Definitionen interessant:

- *Restrictions* ($(vn)P$) beschränken einen Namen auf einen Prozess, d.h. alle Prozesse, die nicht Teil von P sind, können n nicht verwenden. [4]
- $fn(P)$**:** *freie Namen* in P, d.h. solche, die weder belegt sind (d.h. entsprechendes Ambient existiert), noch *restricted* sind.

Mögliche Prozesse:

$(vn)P$	Name n ist auf P beschränkt, d.h. *capabilities* dürfen ihn nur verwenden, wenn sie innerhalb von bzw. 'vor' P sind, s. auch [4]
0	Inaktivität
$P\|Q$	Komposition (zweier paralleler Prozesse)
$!P$	Replikation (beliebig viele 'Kopien' von P laufen parallel) Bem.: Namen (auch von Ambients) sind nicht eindeutig
$n[P]$	Ambient n, enthält Prozess P
$M.P$	Aktion (*Capability* M wird ausgeführt!)

3.2. Umformungsregeln

Wie bereits angedeutet, können sich mit Mobile Ambients modellierte Systeme im Zeitablauf verändern, daher ist es sinnvoll Umformungsregeln zu definieren. Es gibt hierbei zwei zentrale Regeln:

1. $P \equiv Q$: *congruence*, d.h. Umformungsregeln, die nicht das Ausführen einer Aktion beinhalten, z.B. $P \equiv P|0$. Anders gesagt [3]: die Prozesse sind äquivalent (equivalence), bis auf triviale Syntax-Umformungen.
2. $P \rightarrow Q$: *reduction relation*s beschreiben, welcher Zustandsübergang durch Aktionen ausgelöst wird. Aktionen sind in diesem Zusammenhang z.B. Kommunikation oder Mobilitäts-Operationen.

In der Regel dienen Kongruenzumformungen dazu eine Verkettung von Reduktionen zu ermöglichen, wie von [5] auf Seite 4 beschreiben.

Neben diesen beiden Regeln gibt es noch weitere Umformungen, die beim Einsatz von Mobile Ambients helfen.

- Definition: $x \urcorner y$ definiert ein neues Element für das Kalkül.
- Gleichheit (*identity*):
  $(vn)P = (vm)P\{n \diamond m\}, m \diamond fn(P)$
  bedeutet, dass die Ausdrücke bis auf die für sie vergebenen Namen identisch sind.

- *Contextual equivalence*:
  $P \diamond n \neg P \equiv (\nu m_1 .... m_i)(n[P^{\diamond}] | P^{\diamond})$, wobei $n \diamond \{m_1 ..... m_i\}$, d.h. wir haben eine Situation, in der ein *n* parallel zu einem anderen Prozess existiert, wobei *n* nicht zu den restriktierten Namen gehört (und somit von P manipuliert werden kann).
- *Transitive closure:*
  $P \diamond n \neg P \rightarrow^{\neg} Q \neg (Q \diamond n)$, d.h. die oben beschriebene Situation ist von P aus über eine *transitive closure* erreichbar.
- *Contextual equivalence*:
  $P \neg Q \neg \diamond n, C[] : C[P] \diamond n \diamond C[Q] \diamond n$, d.h. für einen 'unvollständigen' Prozess C (mit Löchern), haben beide als Füller die gleiche Wirkung, nämlich es taucht irgendwann ein nicht-geschütztes *n* auf.

3.2.1. Reduction Rules

Die Wirkung der verschiedenen auszuführenden Aktionen eines Prozesses (die durch *capabilities* ausgedrückt werden) wird durch *reduction rules* beschrieben. Die grundlegenden Regeln werden in Abbildung 3 definiert.

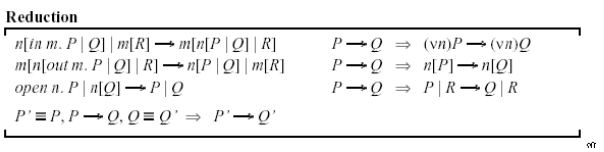

Abb. 3: Reduktionen [X1]

Einige der wichtigsten Regeln sollen hier kurz erläutert werden. Man kann zwei Arten von Regeln unterscheiden. Zum einen gibt es Regeln, welche im Grunde genommen die verschiedenen *capabilities* definieren. Z.B. weiß man aufgrund der Regel

$$n[in\ m.P|Q]|m[R] \rightarrow m[n[P|Q]|R]$$

dass ein *in* sein umgebendes Ambient in ein anderes hineinzieht, inklusive aller enthaltener Prozesse (s. Abb. 2). Anders gesagt beschreibt jede Reduktion einen Abarbeitungsschritt eines Prozesses. Aus diesen Definitionen kann man auch ein interessantes Verhalten bei Mobile Ambients ableiten - ist nämlich eine Aktion nicht ausführbar, so wartet der Prozess so lange, bis sie es ist, da keine Reduktion möglich ist. Dies wird besonders im Zusammenhang mit Kommunikation interessant, da hierdurch Prozesse beim Warten auf Input blocken.

Die zweite Gruppe von Regeln erlaubt eine Schachtelung von Prozessen, indem sie Aussagen darüber trifft, wann bestehende Reduktionsregeln auch in zusammengesetzten Konstrukten zum Einsatz kommen.

Eine wichtige Regel lautet z.B.

$$P \rightarrow Q \diamond n[P] \rightarrow n[Q]$$, d.h.

kann P zu Q reduziert werden, so gilt dies auch, wenn P in ein Ambient eingeschlossen ist.

Einige der *reduction rules* sollen noch kurz näher erläutert werden, um ein Verständnis der weiterführenden Konzepte zu erleichtern.

- $m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]$, ist die Umkehrung zu *in* und erlaubt ein Verlassen von Ambients. Auch hier wird das umgebende Ambient mitbewegt. Diese Art von Bewegung wird von Cardelli und Gordon als subjektiv bezeichnet [3]. Eine alternative Form der Bewegung wäre objektiv, d.h. ohne Mitnahme des umgebenden Ambients. Beide Bewegungsarten werden einander in Abb. 4 gegenübergestellt. Eine objektive Bewegung müsste wie folgt definiert sein:
  - *Move in:* $mv\ in\ m.P|m[R] \rightarrow m[P|R]$
  - *Move out*: analog

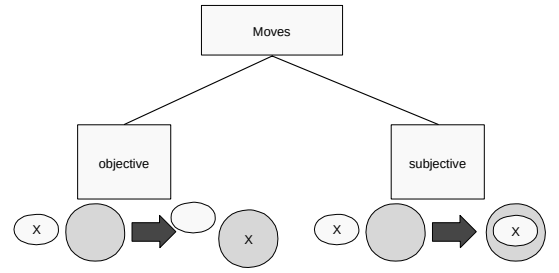

Abb. 4: Objektive und Subjektive Bewegung

- $open\ n.P|n[Q] \rightarrow P|Q$ öffnet ein paralleles Ambient, indem es dieses auflöst, um seinen Inhalt freizusetzen. Eine Alternative wäre eine *acid*-Capability wie bei S. 9 [3], die ein Ambient von innen heraus auflöst. *Open* wird hauptsächlich zu Kommunikationszwecken eingesetzt (s. Abschnitt 3.4), ist aber nicht ganz unproblematisch, wie in [5] ausgeführt. Außerdem kann es dazu verwendet werden, objektive Bewegungen mittels subjektiver zu kodieren, was durch die folgenden Definitionen ausgedrückt wird:
  - $allow\ n \neg\ !open\ n.0$
  - Betreten eines Ambients wird wie folgt umgesetzt: Wir haben ein Ambient um den Prozess, das mittels *in n* nach n gebracht wird, dort wird es verlassen und 'beseitigt' (in Kombination mit einem *allow*!), herausgehen analog. $mv\ in\ n.P \neg$

$(\nu k)k[in\ n.in[out\ k.open\ k.P]]$ (in Verbindung mit: $n^{\diamond}[P] \mid n[P|allow\ in]$, bzw. $n^{\diamond}$("allow out") und $n^{\diamond}$ ("allow in-out")).

Die meisten Reduktionsregeln sind weitgehend intuitiv. Sie sind auch später in der Logik interessant, vor allem im Zusammenhang mit *sometime*-Aussagen (s. Abschnitt 4.2.2).

3.3. Kommunikationsprimitive

Das Kalkül stellt *capabilities* für Kommunikation zwischen Ambients zur Verfügung. Kommunikation wurde von Cardelli und Gordon so modelliert, dass sie es nicht erlaubt, die Einschränkungen auf *capabilities*, wie sie bisher definiert wurden, einfach zu umgehen. Deshalb ist Kommunikation nur asynchron und innerhalb eines Ambients möglich[2]. Somit muss ein anderes Ambient zur Kommunikation betreten werden, was z.B. durch Namensbeschränkungen unmöglich gemacht werden kann. Eine typische Kommunikation wird in Abb. 5 dargestellt.

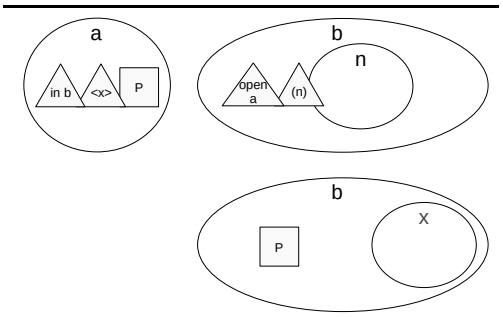

Abb. 5:Example of Inter-Ambient Communication

Die Abbildung zeigt die beiden für Kommunikation zuständigen *capabilities*.

$\neg M \neg$, ist eine asynchrone *output action*, d.h. M wird auf den 'Äther' gestellt und kann von *input actions* eingefangen werden. Die Äther-Methapher ist so zu interpretieren, dass der Wert M von der nächsten *input action*, welche im gleichen Ambient ausgeführt wird, aufgenommen wird und so lange unsichtbar präsent ist.

Wenn eine *input action* $(x).P$ einen Wert in *x* aufnimmt, so bewirkt dies, dass alle x, die im gleichen Prozess *P* nach (x) kommen, den entsprechenden Wert zugewiesen bekommen.

Die zu diesem Vorgang gehörige und in Abb. 5 illustrierte Reduktion lautet wie folgt:

$$(x).P|\neg M \neg \diamond\ P\{x \diamond M\},$$

d.h. alle in P vorkommenden x werden durch M ersetzt.

Kommunikationsprimitive können zu verschiedenen Zwecken genutzt werden, z.B.:

- Befehle können weitergeben werden, d.h. man fügt eine ganze *capability* in den Prozess ein: $(x).x.P$
- Man kann Ambients benennen: $(x).x[P]$
- Man kann Ambient-Namen übergeben, damit ein Prozess diese in einer *capability* (z.B. *open*) einsetzen kann.

3.4. Kommunikation zwischen Ambients

Die Kommunikationsprimitive aus dem vorherigen Abschnitt dienen lediglich der Kommunikation innerhalb eines Ambients. Es ist allerdings unter Verwendung dieser Primitive möglich, eine Inter-Ambient Kommunikation zu modellieren, wie in [3] beschrieben.

Es sollte keine Überraschung sein, dass zur Kommunikation zwischen Ambient ein 'Hilfsambient' als Transportmedium zum Einsatz kommt. Man kann dieses Ambient als einen mobilen Agenten betrachten.

Um einen allgemeineren Ausgangspunkt als in Abb. 5 zu haben, betrachte man eine Situation wie in Abb. 6 dargestellt. Eine Information *x* aus dem Ambient *a* soll nach Ambient *b* transportiert werden und dort von Prozess *Q* in Empfang genommen werden.

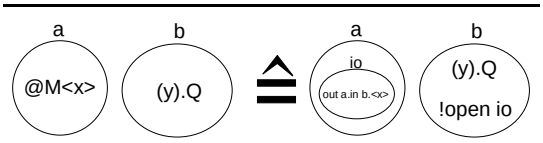

Abb. 6: Remote Output schematisch betrachtet

Das hierzu neu eingeführte Element, so wie es in der Abbildung eingesetzt wird, ist wie folgt definiert:

Remote Output bei M: $@M\neg a \neg \mid io[M.\neg a \neg]$, d.h. wir bewegen uns auf dem Pfad M und werden im Ziel geöffnet, um unseren Wert auf den Äther zu stellen. *M* besteht aus einer Abfolge von *in* und *out capabilities*.

In diesem Zusammenhang wird angenommen. dass im Zielambient *b* mehrere Prozesse zum Öffnen von *io* parallel laufen, also $!open\ io$.

Analog zu entferntem Schreiben, kann man auch entferntes Lesen definieren. Ist M^{-1} der umgekehrte Pfad zu *M*, dann lässt sich *remote input* beschreiben als:$@M(x)M^{-1}.P \neg$ $(\nu n)(io[M.(x).n[M^{-1}.P]]|open\ n)$, d.h. *io* läuft

[2] Wie in Abb. 5 und Abschnitt 3.4 zu sehen, kann eine Kommunikation zwischen Ambients nur mittels *in* bzw. *out* stattfinden - damit sind alle Einschränkungen auf diese *capabilities* auch für die Kommunikation relevant!

nach *M* und empfängt dort einen Wert, der in *x* gespeichert wird. *n* dient als Ambient für die Rückkehr, daher ist dieser Name auch geschützt (damit es nicht am Ziel aufgelöst wird). Zurück in *a* wird es dann von *open n* aufgelöst. Problematisch hierbei ist, dass P hin- und zurückübertragen wird, obwohl es ja erst am Ausgangsort mit *x* weiterarbeiten soll. Eine bessere *remote input* Variante würde wie folgt aussehen. P ist lokal und wartet auf einen Input. Der Transportagent ist wie oben, nur dass er den gelesenen Wert am Ursprungsort für P freisetzt.

Ein Remote Procedure Call sähe in diesem Rahmen wie folgt aus:

$$@M\ \mathrm{arg}\neg a\neg\, res(x)M^{-1}.P \neg (\nu n)$$
$$(io[M.(\neg a\neg|open\ res.(x).n[M^{-1}.\neg x\neg])|open\ n])$$
$$|(x).P,$$

3.5. Kommunikation über Kanäle

Kanäle können auch über Ambients simuliert werden. Kommunikation erfolgt auf dem Kanal, d.h. alle Lese- und Schreibwünsche werden dorthin gesetzt und agieren dann miteinander. Die Wünsche werden in *io* Ambients transportiert.

Ein Kanal wird definiert als

$$ch\ n \neg\ n[!open\ io],$$

d.h. alle io-Operationen werden im Kanal freigesetzt.

Ein neuer Kanal wird geöffnet mit dem Ausdruck:

$$(\nu n)(ch\ n|P),$$

d.h. parallel zu P läuft ein Kanal *ch*.

Schreiben auf einen Kanal (asynchron) wird mit folgendem Kostrukt durchgeführt:

$$n\neg M\neg\neg\ io[in\ n.\neg M\neg],$$

d.h. *io* geht in ein Ambient *n* (den Kanal) und wird dort freigesetzt, durch das *open*.

Das Lesen von einem Kanal ist entsprechend wie folgt definiert:

$$n(x).P \neg (\nu p)(io[in\ n.(x).p[out\ n.P]]|open\ p)$$

3.6. Modellierungsbeispiele

Cardelli und Gordon liefern in ihrer Arbeit viele Beispiele dafür, wie man bestehende Systeme mit Mobile Ambients modellieren kann.

Einige davon sind unter dem Gesichtspunkt interessant, als dass sie veranschaulichen wie gut oder schlecht gewisse Problembereiche durch das Kalkül abgedeckt werden.

Die Modellierung von *locks* ([3], Seite 8) beispielsweise erlaubt es, darauf aufbauend Konzepte der parallelen Programmierung zu erstellen. Monitore und ähnliche Elemente sind Basis für eine Analyse von parallelen Programmen bezüglich ihrer Qualität, eine Eigenschaft von Mobile Ambients, die vielleicht auf den ersten Blick nicht möglich erscheinen würde.

Die *routable packets* ([3], Seite 13) lassen ähnliche Schlüsse bezüglich der Netzwerkprogrammierung zu, auch wenn hier zu bemängeln ist, dass in einem realen System in der Regel in den Paketen keine Prozesse verschickt werden.

Die Turing Maschine ([3], Seite 10) ist ein sehr komplexes Beispiel das aber eine wichtige Eigenschaft von Mobile Ambients beweist - Turing Vollständigkeit! Salopp ausgedrückt bedeutet dies, das jedes Computerprogramm mit Mobile Ambients modelliert werden kann.

4. Die modale Logik (*Anytime Anywhere*)

Die modale Logik zu Mobile Ambients wurde ebenfalls von Cardelli und Gordon entwickelt [6]. Während das Kalkül Systeme modelliert, kann man mittels der Logik Aussagen über selbige beweisen.

4.1. Der Begriff der Modalen Logik

Modale Logik im Allgemeinen ist die "study of the deductive behavior of the expressions 'it is necessary that' and 'it is possible that'" [7]. In einem weiteren Sinne (so wie hier) enthält sie auch temporale Ausdrücke.

Man kann also mit einer solchen Logik beweisen, dass ein Zustand irgendwann erreicht wird, oder auch dass er nie erreicht wird. Dies ist besonders nützlich im Sicherheitsbereich.

Die modale Logik zu Mobile Ambients erlaubt Aussagen über Raum und Zeit.

- **Raum:** Ein System kann entweder als Venn-Diagramm oder als ungeordneter Baum mit beschrifteten Kanten dargestellt werden (vgl. Abb. 7). Die Kanten sind Ambients, die Prozesse sitzen in den Knoten.
- **Zeit:** Prozesse können Unterbäume von einer Position an eine andere bewegen.

Abb 7.: Darstellung eines Systems [6]

4.2. Grundlagen

Wir können Prozesse in räumlich (Parallelität, Ambients etc.) und temporal (Capabilities, Input, Output) einteilen (Abb. 8). Besteht ein Prozess nur aus räumlichen Operatoren, so handelt es sich um eine *spatial configuration.*

$P,Q,R ::=$	processes	
$\mathbf{0}$	void	spatial
$P \mid Q$	composition	spatial
$!P$	replication	spatial
$M[P]$	ambient	spatial
$M.P$	capability action	temporal
$(n).P$	input action	temporal
$\langle M \rangle$	output action	temporal

Abb. 8: Temporal vs. Spatial [6]

congruence-Beziehungen bedeuten im Grunde genommen, dass sich eigentlich nichts verändert hat, eine räumliche Anordnung wird nur anders beschrieben.

reduction-Beziehungen hingegen modellieren das dynamische Verhalten von Ambients (und somit Veränderung im Zeitablauf). Dies umfasst Bewegung und Input/Output.

Um ein räumliches Äquivalent zur Reduktion zu haben, führt man folgende Definition ein:

$P \diamond P^{\diamond} \diamond\!\!\!\!\diamond \quad n, P^{\diamond}.P \equiv n[P^{\diamond}] \mid P^{\diamond}$, also: P enthält ein P' genau eine Ebene tiefer. Analog ist $\diamond^{\neg}$ definiert!

4.2.1. Logik und Kalkül

Wie sich im Folgenden zeigen wird, besteht eine enge Beziehung zwischen Logik und Kalkül. Leider jedoch baut die Logik nur auf einer Untermenge des Kalküls auf - es ist in der Logik nicht möglich, Namensbeschränkungen zu berücksichtigen. Die Folgen dieser drastischen Einschränkung werden in Abschnitt 6 näher erläutert.

4.2.2. Was sagt eine Logik-Formel aus?

Die Aussage einer Formel bezieht sich immer auf das hier und jetzt, kann aber ausgehend davon weitergehen. Beispiele für Logikausdrücke sind:

- $n[0]$: "hier und jetzt gibt es einen leeren Ort namens n" [6]
- $n[A]$: erlaubt es, eine Stufe weiterzugehen (räumlich).
- $\diamondsuit A$: erlaubt beliebige Schritte im Raum
- $\diamond A$: erlaubt beliebig viele Reduktionen (d.h. wir reden über irgendeinen Zeitpunkt in der Zukunft von jetzt an).
- $fn(A)$: Freie Namen einer Formel sind alle, die gar nicht vorkommen. (Es gibt ja in diesem Zusammenhang gar kein Name Binding!)
- $fv(A)$: Freie Variablen sind alle, die nicht durch Quantifizierer gebunden sind. Eine Formel ist geschlossen, wenn gilt: $fv(A) = \diamond$.

4.3. Satisfaction und Validity

Satisfaction, *Validity* und, in einem geringeren Grad auch *Satisfiability*, sind die zentralen Aspekte der modalen Logik für Mobile Ambients. Nach einer kurzen Übersicht sollen die einzelnen Begriffe näher erläutert werden.

- Satisfaction: $P \neg A$, Prozess P befriedigt die geschlossene Formel A.
- Validity: $vld(A) \neg \diamond P : \pi .P \neg A$. Eine geschlossene Formel ist valid, wenn sie von jedem Prozess befriedigt wird. Dies wird für *inference rules* und *sequents* verwendet
- Satisfiability: $Sat A$, bedeutet, dass A satisfiable ist, d.h. $P \neg Sat A \diamond\!\!\!\!\diamond \quad P^{\diamond} : \pi .P^{\diamond} \neg A$, d.h. es gibt einen Prozess *P'*, der A satisfied (s. 4.4.1)

4.3.1. Satisfaction

Die verschiedenen Definitionen für *satisfaction* sind in Abb. 9 wiedergegeben, wobei 4 Fälle besonders interessant sind:

- $P \neg \diamond A$: Es gibt eine Folge von Reduktionen, die zu einem Prozess führen, der A befriedigt ("sometime").
- $P \neg \quad \diamondsuit A$: Es gibt irgendwo in P verschachtelt einen Prozess, der A befriedigt ("somewhere").
- $P \neg A@n$: A wird befriedigt, wenn P im Ambient n ist.
- $P \neg A \neg B$: egal, was für ein Prozess P' parallel zu P läuft, solange P' A befriedigt, befriedigt die Kombination aus beiden B. Diese Definition ist wichtig im Zusammenhang mit Security ("alle Angreifer die A befriedigen, können in der Konstellation P abgewehrt werden!").

Es gibt außerdem noch eine Reihe abgeleiteter Regeln (*Derived Connectives*). Die wichtigsten sind:

- $A \neg B \neg (\neg A \neg B)$: Für alle parallelen Zerlegungen eines Prozesses gilt, dass entweder ein Teil A oder der andere B befriedigt.
- $A^{\diamond} \neg A \diamond F$: Für jede Zerlegung muss eine Komponente A befriedigen (da F nie befriedigt werden kann!)

- $A^{\diamond} \urcorner A|\mathrm{T}$: Es gibt eine Dekomposition, bei der eine Komponente A befriedigt. (T wird von der verbleibenden automatisch befriedigt!)
- $A \urcorner B \urcorner (B \urcorner A)$ (Fusion): Es gibt einen Kontext der B befriedigt und bei dem trotzdem noch für den Gesamtprozess A gilt.
- $A|\diamond\ B \urcorner (A|\urcorner B)$ (Fusion Adjunct): Für alle Dekompositionen gilt, wenn ein Teil A befriedigt, muss der andere B befriedigen. (Es gibt keinen Fall, bei dem A befriedigt wird, aber B nicht.)
- Everytime und Everywhere (die durch ein auf der Seite liegendes Sometime, Somewhere symbolisiert werden): Es gibt keinen Zeitpunkt/Ort, wo A nicht gilt

Die Definition von Satisfaction ist noch einmal in Abb. 9 zusammengefasst. Als abschließende Bemerkung sei noch gesagt, dass Satisfaction invariant bei struktureller Kongruenz von Prozessen ist, eine Tatsache, die intuitiv als richtig erscheint.

Satisfaction

$\forall P{:}\Pi.$	$P \models \mathbf{T}$		
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \models \neg\mathcal{A}$	$\triangleq$	$\neg P \models \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A},\mathcal{B}{:}\Phi.$	$P \models \mathcal{A}\vee\mathcal{B}$	$\triangleq$	$P \models \mathcal{A} \vee P \models \mathcal{B}$
$\forall P{:}\Pi.$	$P \models \mathbf{0}$	$\triangleq$	$P \equiv \mathbf{0}$
$\forall P{:}\Pi, n{:}\Lambda, \mathcal{A}{:}\Phi.$	$P \models n[\mathcal{A}]$	$\triangleq$	$\exists P'{:}\Pi.\ P \equiv n[P'] \wedge P' \models \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A},\mathcal{B}{:}\Phi.$	$P \models \mathcal{A}\,/\,\mathcal{B}$	$\triangleq$	$\exists P', P''{:}\Pi.\ P \equiv P'/P''$ $\wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B}$
$\forall P{:}\Pi, x{:}\vartheta, \mathcal{A}{:}\Phi.$	$P \models \forall x\mathcal{A}$	$\triangleq$	$\forall m{:}\Lambda.\ P \models \mathcal{A}\{x \leftarrow m\}$
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \models \Diamond\mathcal{A}$	$\triangleq$	$\exists P'{:}\Pi.\ P \rightarrow^{*} P' \wedge P' \models \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \models \Diamond\!\!\!\!-\mathcal{A}$	$\triangleq$	$\exists P'{:}\Pi.\ P \downarrow^{*} P' \wedge P' \models \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A}{:}\Phi.$	$P \models \mathcal{A}@n$	$\triangleq$	$n[P] \models \mathcal{A}$
$\forall P{:}\Pi, \mathcal{A},\mathcal{B}{:}\Phi.$	$P \models \mathcal{A}\triangleright\mathcal{B}$	$\triangleq$	$\forall P'{:}\Pi.\ P' \models \mathcal{A} \Rightarrow P/P' \models \mathcal{B}$

Abb. 9: Satisfaction [6]

4.3.2. Validity

Im Zusammenhang mit der Validität sind die folgenden Definitionen interessant:

Will man, ausgehend von einer Aussage über einen Prozess, eine Behauptung über diesen beweisen, so verwendet man im Allgemeinen eine Verkettung von *sequents*. Ein *sequent* ist definiert als

$$A \urcorner B \urcorner vld(A \diamond B), \text{ d.h.}$$

für alle Prozesse gilt, aus A folgt B. Bei einem Beweis kann diese Regel verwendet werden um aus einem Fakt *A* einen anderen (*B*) abzuleiten, da gilt: Befriedigt ein Prozess A, dann wird er *immer* auch B befriedigen!

Weitere Definitionen in diesem Zusammenhang sind:

- $A \urcorner B \urcorner A \urcorner B \urcorner B \urcorner A$: *double sequent*
- $A_1 \urcorner B_1; A_2 \urcorner B_2; A_3 \urcorner B_3 \urcorner A_0 \urcorner B_0$ $\urcorner A_1 \urcorner B_1 \urcorner A_2 \urcorner B_2 \urcorner A_3 \urcorner B_3$ $\diamond\ A_0 \urcorner B_0$: *inference rule*. Die Aussage der Formel ist: Wenn für alle Prozesse die Regeln auf der Linken gelten, dann gilt auch die Regel auf der rechten für alle Prozesse!
- Analog sind auch eine *double conclusion* (führt zu einem *double sequent*) und *double rule* (eine Regel, die in beide Richtungen gilt)

4.3.3. Logikregeln

Mit der Definition von *inference rules* wird eine Notation bereitgestellt, die es ermöglicht Logikregeln aufzustellen, welche im Rahmen eines Beweises von *assertions* zum Tragen kommen können. Aufgrund des Umfangs wäre es müßig, alle Regeln erläutern zu wollen, daher sei die Auswahl auf einige interessante beschränkt.

Es gibt vier Arten von Regeln (s. Abb. 10 -13).

Propositional rules (Abb. 10) beschreiben grundlegende Eigenschaften wie Assoziativität oder Kommutativität. Die meisten dieser Regeln sind intuitiv verständlich oder aus der allgemeinen modalen Logik bekannt.

Propositional Rules

(A-L)	$\mathcal{A}\wedge(\mathcal{C}\wedge\mathcal{D}) \vdash \mathcal{B} \;\{\}\; (\mathcal{A}\wedge\mathcal{C})\wedge\mathcal{D} \vdash \mathcal{B}$
(A-R)	$\mathcal{A} \vdash (\mathcal{C}\vee\mathcal{D})\vee\mathcal{B} \;\{\}\; \mathcal{A} \vdash \mathcal{C}\vee(\mathcal{D}\vee\mathcal{B})$
(X-L)	$\mathcal{A}\wedge\mathcal{C} \vdash \mathcal{B} \;\}\; \mathcal{C}\wedge\mathcal{A} \vdash \mathcal{B}$
(X-R)	$\mathcal{A} \vdash \mathcal{C}\vee\mathcal{B} \;\}\; \mathcal{A} \vdash \mathcal{B}\vee\mathcal{C}$
(C-L)	$\mathcal{A}\wedge\mathcal{A} \vdash \mathcal{B} \;\}\; \mathcal{A} \vdash \mathcal{B}$
(C-R)	$\mathcal{A} \vdash \mathcal{B}\vee\mathcal{B} \;\}\; \mathcal{A} \vdash \mathcal{B}$
(W-L)	$\mathcal{A} \vdash \mathcal{B} \;\}\; \mathcal{A}\wedge\mathcal{C} \vdash \mathcal{B}$
(W-R)	$\mathcal{A} \vdash \mathcal{B} \;\}\; \mathcal{A} \vdash \mathcal{C}\vee\mathcal{B}$
(Id)	$\} \; \mathcal{A} \vdash \mathcal{A}$
(Cut)	$\mathcal{A} \vdash \mathcal{C}\vee\mathcal{B};\ \mathcal{A}'\wedge\mathcal{C} \vdash \mathcal{B}' \;\}\; \mathcal{A}\wedge\mathcal{A}' \vdash \mathcal{B}\vee\mathcal{B}'$
(**T**)	$\mathcal{A}\wedge\mathbf{T} \vdash \mathcal{B} \;\}\; \mathcal{A} \vdash \mathcal{B}$
(**F**)	$\mathcal{A} \vdash \mathbf{F}\vee\mathcal{B} \;\}\; \mathcal{A} \vdash \mathcal{B}$
($\neg$-L)	$\mathcal{A} \vdash \mathcal{C}\vee\mathcal{B} \;\}\; \mathcal{A}\wedge\neg\mathcal{C} \vdash \mathcal{B}$
($\neg$-R)	$\mathcal{A}\wedge\mathcal{C} \vdash \mathcal{B} \;\}\; \mathcal{A} \vdash \neg\mathcal{C}\vee\mathcal{B}$

Abb. 10: Propositional Rules

Interessanter sind die *composition rules* (Abb. 11) und deren Korollare. Hier werden Aussagen über das Verhalten von Kompositionen mehrerer Prozesse gemacht. Da nahezu alle praktischen Anwendungen zusammengesetzt sind, werden diese Regeln sehr häufig zum Einsatz kommen. Vor allem die Regel $(|\urcorner)$ ist essentiell, wie im Beispiel in Abschnitt 5 klar wird.

Umformungsregeln bezüglich der Positionsangaben werden durch die *location rules* (Abb. 12) bereitgestellt. Auch hierzu sind einige Corollare definiert.

Zu guter Letzt gibt es *time and space modality rules* (Abb. 13), welche Aussagen über das Verhalten der *sometime* und *somewhere* Operatoren zulassen. Einige der Regeln sollen hier kurz erläutert werden:

- 'Immer' ist Double Sequent zu, 'es ist nie nicht'.

- 'Überall' ist Double Sequent zu, 'es ist nirgendwo nicht'.
- Wenn immer *A* gilt, dann gilt auch *A*: $\neg \Diamond A \neg A$
- Folgt aus *A* immer *B*, dann gilt immer *B* wenn immer *A* gilt: $A \neg B \neg \Diamond A \neg \Diamond B$

Composition Rules

$(\mid \mathbf{0})$	$\vdash \mathcal{A} \mid \mathbf{0} \dashv\vdash \mathcal{A}$
$(\mid \neg\mathbf{0})$	$\vdash \mathcal{A} \mid \neg\mathbf{0} \vdash \neg\mathbf{0}$
$(\mathrm{A} \mid)$	$\vdash \mathcal{A} \mid (\mathcal{B} \mid C) \dashv\vdash (\mathcal{A} \mid \mathcal{B}) \mid C$
$(\mathrm{X} \mid)$	$\vdash \mathcal{A} \mid \mathcal{B} \vdash \mathcal{B} \mid \mathcal{A}$
$(\mid \vdash)$	$\mathcal{A}' \vdash \mathcal{B}';\ \mathcal{A}'' \vdash \mathcal{B}'' \vdash \mathcal{A}' \mid \mathcal{A}'' \vdash \mathcal{B}' \mid \mathcal{B}''$
$(\mid \vee)$	$\vdash (\mathcal{A} \vee \mathcal{B}) \mid C \vdash \mathcal{A} \mid C \vee \mathcal{B} \mid C$
$(\mid \parallel)$	$\vdash \mathcal{A}' \mid \mathcal{A}'' \vdash (\mathcal{A}' \mid \mathcal{B}'') \vee (\mathcal{B}' \mid \mathcal{A}'') \vee (\neg\mathcal{B}' \mid \neg\mathcal{B}'')$
$(\mid \triangleright)$	$\mathcal{A} \mid C \vdash \mathcal{B} \dashv\vdash \mathcal{A} \vdash \mathcal{B}$

Abb. 11: Composition Rules

Location Rules

$(n[\,] \neg\mathbf{0})$	$\vdash n[\mathcal{A}] \vdash \neg\mathbf{0}$
$(n[\,] \neg \mid)$	$\vdash n[\mathcal{A}] \vdash \neg(\neg\mathbf{0} \mid \neg\mathbf{0})$
$(n[\,] \vdash)$	$\mathcal{A} \vdash \mathcal{B} \dashv\vdash n[\mathcal{A}] \vdash n[\mathcal{B}]$
$(n[\,] \wedge)$	$\vdash n[\mathcal{A}] \wedge n[\mathcal{B}] \vdash n[\mathcal{A} \wedge \mathcal{B}]$
$(n[\,] \vee)$	$\vdash n[\mathcal{A} \vee \mathcal{B}] \vdash n[\mathcal{A}] \vee n[\mathcal{B}]$
$(n[\,] @)$	$n[\mathcal{A}] \vdash \mathcal{B} \dashv\vdash \mathcal{A} \vdash \mathcal{B}@n$
$(\neg @)$	$\vdash \mathcal{A}@n \dashv\vdash \neg((\neg\mathcal{A})@n)$

Abb. 12: Location Rules

Time and Space Modality Rules

$(\Diamond)$	$\vdash \Diamond\mathcal{A} \dashv\vdash \neg\Box\neg\mathcal{A}$	$(\Diamond)$	$\vdash \Diamond\mathcal{A} \dashv\vdash \neg\Box\neg\mathcal{A}$
$(\Box\ \mathrm{K})$	$\vdash \Box(\mathcal{A} \Rightarrow \mathcal{B}) \vdash \Box\mathcal{A} \Rightarrow \Box\mathcal{B}$	$(\Box\ \mathrm{K})$	$\vdash \Box(\mathcal{A} \Rightarrow \mathcal{B}) \vdash \Box\mathcal{A} \Rightarrow \Box\mathcal{B}$
$(\Box\ \mathrm{T})$	$\vdash \Box\mathcal{A} \vdash \mathcal{A}$	$(\Box\ \mathrm{T})$	$\vdash \Box\mathcal{A} \vdash \mathcal{A}$
$(\Box\ 4)$	$\vdash \Box\mathcal{A} \vdash \Box\Box\mathcal{A}$	$(\Box\ 4)$	$\vdash \Box\mathcal{A} \vdash \Box\Box\mathcal{A}$
$(\Box\ \mathbf{T})$	$\vdash \mathbf{T} \vdash \Box\mathbf{T}$	$(\Box\ \mathbf{T})$	$\vdash \mathbf{T} \vdash \Box\mathbf{T}$
$(\Box \vdash)$	$\mathcal{A} \vdash \mathcal{B} \vdash \Box\mathcal{A} \vdash \Box\mathcal{B}$	$(\Box \vdash)$	$\mathcal{A} \vdash \mathcal{B} \vdash \Box\mathcal{A} \vdash \Box\mathcal{B}$
$(\Diamond n[])$	$\vdash n[\Diamond\mathcal{A}] \vdash \Diamond n[\mathcal{A}]$	$(\Diamond n[])$	$\vdash n[\Diamond\mathcal{A}] \vdash \Diamond\mathcal{A}$
$(\Diamond \mid)$	$\vdash \Diamond\mathcal{A} \mid \Diamond\mathcal{B} \vdash \Diamond(\mathcal{A} \mid \mathcal{B})$	$(\Diamond \mid)$	$\vdash \Diamond\mathcal{A} \mid \mathcal{B} \vdash \Diamond(\mathcal{A} \mid \mathbf{T})$
$(\Diamond\Diamond)$	$\vdash \Diamond\Diamond\mathcal{A} \vdash \Diamond\Diamond\mathcal{A}$		

Abb. 13: Modality Rules

4.4. Weitere Elemente

Mit den bisher definierten Elementen können bereits viele Beweise durchgeführt werden, z.B. das in [6] aufgeführte Beispiel oder die Analyse in Abschnitt 5 dieser Arbeit. Der Vollständigkeit halber sollen jedoch noch zwei weitere Elemente erwähnt werden, die in einer Logik generell definiert sind; *satisfiability* und Prädikate. Im Anschluss soll außerdem auf den Model Checker eingegangen werden, ein nützlicher Algorithmus zur automatischen Überprüfung von Aussagen.

4.4.1. Satisfiability

Ist ein Term *satisfiable* ($P \neg Sat\ A$), so gibt es mindestens einen Prozess, für den *A* wahr ist. Daraus abgeleitet kann man festellten, dass wenn *A unsatisfiable* ist, dann ist *A* false. Ausserdem gilt, wenn *A satisfiable* ist, dann ist es A^F nicht.

4.5. Prädikate

Die Prädikatenlogik ist das, was man generell mit dem Ausdruck 'Logik' in Verbindung bringt und eine entsprechende Erweiterung der Logik von Mobile Ambients macht daher auch Sinn. Es sind nun auch freie Variablen möglich und wir benötigen folgende Definitionen:

- $fn(A) = \{x_1, ..., x_n\}$ sind die freien Variablen von *A*.
- $\pi \neg fn(A) \Diamond \pi$ ist eine Substitution der Variablennamen
- $A_\pi \neg A\{x_1 \Diamond \pi(x_1), ...\}$
- D.h. Validität wird in diesem Zusammenhang definiert als:
  $vld(A) \neg \Diamond\pi \neg fn(A) \Diamond \pi .\Diamond P : \pi .P \neg A_\pi$,
  d.h. *A* ist valid für alle Ausprägungen seiner Variablen.

4.5.1. Der Model Checker

Mit dem Model Checker kann man algorithmisch Aussagen der folgenden Art daraufhin untersuchen, ob sie wahr oder falsch sind.

$$P \neg A$$

Die Regeln des Model Checkers sind umfangreich aber leicht zu verstehen und sollen daher hier außen vor gelassen werden. Vom Prinzip her handelt es sich um die rekursive Anwendung der jeweils passenden Checker-Regel auf Teilaussagen von *A*.

Leider funktioniert der Checker nur für ein Subset der Logik, Replikationen und der $\neg$ Operator werden nicht unterstützt. Dieses Problem kann jedoch zum Beispiel dadurch kompensiert werden, dass man vor der Anwendung des Algorithmus einige Umformungsschritte 'von Hand' ausführt, insbesondere unter Verwendung des $(\neg \mid)$ Korrolars, das bei einem entsprechenden parallelen Prozess zur Auflösung des Operators führt.

5. Anwendung von Mobile Ambients

Mobile Ambients umfassen ein Kalkül und eine modale Logik. Das Kalkül erlaubt es, verteilte Systeme mit Mobilität zu modellieren, die Logik kann dazu verwendet werden Aussagen über ein solches System zu beweisen. Im Folgenden soll analysiert werden, wie tauglich das Konzept für die gesteckten Ziele ist und welche Alternativen existieren. Als erster Schritt soll anhand eines Beispiels ein kleiner Beweis mittels Mobile Ambients getätigt werden.

5.1. Ausgangssituation

Das folgende Beispiel ist an die Architektur von Hive [8] angelehnt, einer Ubiquitous Computing Architektur, die ebenfalls im Rahmen dieses Seminars vorgestellt wurde.

Abb. 14 zeigt einen Modellierungsansatz um das System in der Notation des Mobile Ambient Calculus darzustellen.

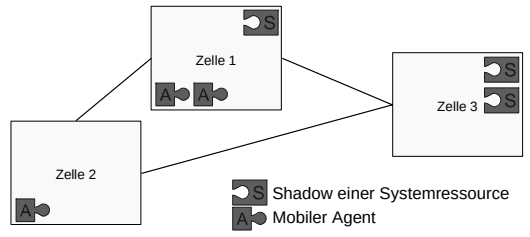

Abb 14: Die Architektur des HIVE-Systems

Die folgenden Elemente sind am Modell beteiligt:

- **Agenten** werden modelliert als:
  $\text{Agent}_k(\text{move-logic,keys}) \urcorner a_k[\text{move-logic} \mid !\text{key-logic} \mid \text{keys}]$
  d.h. ein Agent besteht aus seinem Pfad und seinen Schlüsseln. Der Einfachheit halber gehen wir davon aus, dass ein Schlüssel immer genutzt wird. Anderenfalls müsste man für jeden Schlüssel eine eigene key-logic erstellen und diese innerhalb der move-logic an die gewünschte Stelle setzen. Darauf aufbauende definieren wir noch:
  $$P_{kj} \urcorner \text{Agent}_k(\text{move-logic}, \{j\})$$
  sowie $P_{kj} \urcorner \text{Agent}_k(\text{move-logic}, \Diamond)$
  also Agenten mit bzw. ohne den Schlüssel *j*.
- Die **key-logic** ist wie folgt definiert:
  $\text{key-logic} \urcorner \textit{open } j.(a).\textit{in } a.\textit{out } a.0.$
  Sie öffnet das Ambient um den Schlüssel herum und verwendet die dort gespeicherte Information um den Shadow zu betreten und wieder zu verlassen, was eine Verwendung desselben darstellen soll.
- Eine **move-logic** wird wie folgt definiert:
  $\text{move-logic} \urcorner \{P \urcorner \pi : P = \textit{in } x, \Diamond x \urcorner Z\}$
- **Zellen** werden durch folgenden Ausdruck repräsentiert:
  $$\textit{cell}_i(\textit{Service}) \urcorner c_i[\textit{Service}],$$
  die Menge aller Zellen in einem System bezeichnen wir mit *Z*.
- **Schlüssel** lassen sich wie folgt darstellen:
  $$\textit{access } j \urcorner !j[< s_j > .0],$$
  also eine unbegrenzte Menge paralleler Ambients, die den Namen eines Services bereitstellen, wenn sie geöffnet werden.
- Ein **Service (bzw. sein Shadow)**, ist definiert als $\textit{Service}_j \urcorner s_j[0]$.

5.2. Zu untersuchendes Problem

Folgendes Teilproblem soll nun geklärt werden, auch wenn es intuitiv richtig erscheint, und zwar, dass kein Agent einen Service (bzw. den entsprechenden Shadow) ohne den zugehörigen Schlüssel erlangen kann. Wichtig ist hierbei, sich daran zu erinnern, das lediglich das Security-Konzept und nicht die technische Umsetzung geprüft wird!

5.2.1. Security-Konzept bewiesen

Folgendes ist kein ausgereifter Beweis, sondern nur eine Skizze, wie man zum Ziel gelangen kann. Die Behauptung des Beweises ist:

$$P_{k \neg j} | S_j \urcorner \Diamond s_j[a_k[T]]$$

D.h. wann immer $P_{k \neg j}$, also ein Agent k ohne Schlüssel für j, parallel zu einem Shadow zu finden ist, kann man sicher sein, dass der Agent nicht in den Shadow eindringen kann.

Hilfsbehauptungen und Observationen:

1. Für Agent *k* mit Schlüssel für Service *j* gilt: $P_{kj} \urcorner a_k[j[T]|T]$, d.h. der Agentenprozess lässt sich so zerlegen, dass ein Teilprozess im Agentenambient *access j* ist (nur in diesem Zusammenhang nämlich kommt ein Ambient *j* überhaupt vor).
2. Für Agent *k* ohne einen Schlüssel für Service *j* gilt: $P_{k \neg j} \urcorner a_k[\neg(j[T]|T)]$, d.h. es gibt keine Zerlegung, die ein Ambient *j* auftreten lassen würde.
3. Die key-logic erlaubt es, mittels eines Schlüssels einzudringen:
   $\text{key-logic} \urcorner j[T] \urcorner ((s_j \urcorner \Diamond s_j[a_k[T]])@a_k),$

d.h. wenn die key-logic parallel zu einem Ambient $j[T]$ läuft, und beide in a_k stecken, dann gilt für a_k, dass es in s_j eindringen kann.

4. Für die key-logic kann man folgende Aussage machen:
   $$\text{key-logic} \neg (s_j \neg \Diamond s_j[a_k[T]])@a_k,$$
   d.h. die key-logic wird nie *von sich aus* (d.h. ohne key) dazu führen, dass der Service betreten wird.
5. Analoges gilt auch für die move-logic:
   $$\text{move-logic} \neg (s_j \neg \Diamond s_j[a_k[T]])@a_k$$
6. Insbesondere wird auch ein paralleles Laufen beider somit keinen Schaden anrichten.

Beweise:

1. Die erste Aussage ist offensichtlich, wenn man die Definition von Satisfaction betrachtet. Es ist zu zeigen:
   $$P_{kj} \neg a_k[j[T]|T]$$
   $$\Leftrightarrow \quad P^{\Diamond}: P^{\Diamond} \neg j[T]|T \neg P_{kj} = a_k[P^{\Diamond}]$$
   $$\Leftrightarrow \quad P^{\Diamond}, P^{\Diamond} : P^{\Diamond} = j[T] \neg P^{\Diamond} = T$$
   $$\neg P^{\Diamond} = P^{\Diamond}|P^{\Diamond} \neg P_{jk} = a_k[P^{\Diamond}]$$
   Es ist offensichtlich, dass die Wahl
   $$P^{\Diamond} = j[< s_j > .0] \neg P^{\Diamond} = \text{move-logic}|\text{key-logic}$$
   die Bedingung in der letzten Zeile erfüllt.
2. Die zweite Aussage ist analog zu sehen, bzw. sie kann über einen 'brute-force approach' bewiesen werden. Man kann alle Umformungsregeln (Congruence und Reductions) anwenden und wird nie eine Zerlegung erreichen, welche die Bedingung erfüllt.
3. Es ist zu zeigen, dass es eine Reduction gibt, die in den entsprechenden Zustand führt. Dies ist trivial, da die key-logic ja genau dafür konzipiert wurde.
4. Hier ist zu zeigen:
   $$\Diamond P^{\Diamond}: P \Diamond \neg P^{\Diamond} \neg P^{\Diamond} \neg s_j[a_k[T]],$$
   wobei unser P wie folgt definiert ist:
   $$P = a_k[open\ j.(a).in\ a.out\ a.0]|s_j[0]$$
   Es gibt keine Reduction Relation, die sich ohne den key ausführen läßt (das *open* kann erst in Anwesenheit eines Ambients *j* ausgeführt werden), ergo bleibt dieser Prozeß inert.
5. Analog zu Punkt 4.
6. Da trivialerweise gilt: $\text{move-logic} \neg \Diamond j[T]$ und die move-logic nur aus einer Capability besteht, lässt sich schließen, dass die Prozesse sich nicht gegenseitig beeinflussen.

Damit gilt:

$$S_j|P_{k \neg j} = s_j[T]|a_k[\text{key-logic}|\text{move-logic}],$$

und damit

$$S_j|P_{k \neg j} \neg s_j[T]|$$
$$a_k[(s_j \neg s_j[a_k])@a_k|(s_j \neg s_j[a_k])@a_k]$$
$$(\neg)(n[\ ]\neg)(n[A@n]) \quad \neg \quad s_j[T]|s_j \neg \Diamond s_j[a_k] \neg \Diamond s_j[a_k]$$

Als Randbemerkung sei noch erwähnt, dass ein $P_{k \neg j}$ nie ein P_{kj} werden kann (s. Observationen 1 und 2).

5.3. Erkenntnisse durch den Beweis

Die obige Beweisskizze mag 'konstruiert' wirken, aber im Laufe des Beweises haben sich folgende Dinge ergeben:

- Die Notwendigkeit der Trennung des Agenten in verschiedenen Aspekte (move-logic, key-logic und keys) wurde offensichtlich. Nur so kann das Sicherheitskonzept sinnvoll überprüft werden.
- Die Erkenntnis, dass eine defekte oder manipulierte move-logic zu Ärger führen kann!
- Noch zu untersuchen wären:
  - Gegenseitige Beeinflussung von Agenten (kann ein Agent einen anderen hineinziehen, bzw. ihm einen Schlüssel stehlen?)
  - komplexere Systeme mit mehreren Zellen, Shadows etc.

6. Bewertung

Mobile Ambients sind eine interessante Möglichkeit zum theoretischen Überprüfen von verteilten Anwendungen. Im Bezug auf diese Aufgabe weisen sie allerdings verschiedene Stärken und Schwächen auf.

Ein Nachteil, der sofort ins Auge fällt, ist die Komplexität der Materie, insbesondere was die Logik angeht. Ohne umfangreiches Wissen im Bereich der Logik bzw. einem in mathematischen Dingen geschulten Geist, fällt es schwer überhaupt die Grundkonzepte zu erfassen, geschweige denn diese anzuwenden.

Darüber hinaus muss man sich bei Mobile Ambients - wie übrigens bei jeder Form von Modell - die Frage stellen, ob die Umsetzung eines Systems der Wirklichkeit oder dem Wunschdenken des Modellierers entspricht. Tatsächlich gibt es keine Umsetzung, welche eine 1-zu-1 Korrespondenz zwischen Elementen des Modells und der Wirklichkeit gewährleistet. Vielmehr muss bei Modellierung darauf geachtet werden, dass ein Element alle Eigenschaften hat, die man bei seinem realen Pendant kennt. Werden Eigenschaften vergessen oder falsch interpretiert, so ist das Modell falsch und der

Wert der aus ihm gewonnenen Erkenntnisse in Frage gestellt.

Zu guter Letzt ist zu kritisieren, dass die Komplexität eines Modells gerade explodiert, wenn man mehr als die trivialen Aspekte eines Problems beleuchten möchte. Dies ist der Grund, warum eine Prüfung der Deatails eines Problems schwerer ist als eine Prüfung der generellen Konzepts. Hierbei wäre sicher eine Modularisierung hilfreich, d.h. man verwendet bei komplexeren Modellen fertige Komponenten, für die bereits alle relevanten Eigenschaften bewiesen wurden.

6.1. Schwächen des Kalküls

Das Kalkül hat neben den allgemeinen Problemen noch ein paar besondere Schwächen, auf die hier kurz eingegangen werden soll.

Bugliesi, Castagna und Crafa [6] sehen beispielsweise die Kommunikation und die *open-Capability* als problematische Aspekte des Kalküls an.

Letzere erscheint ihnen sehr unrealistisch im Bezug auf Security-Modelling. Nach ihrer Auffassung ist das *open* lediglich aufgrund der gewählten Definition der Kommunikation relevant und könnte bei einer Abwandlung dieser weggelassen werden.

Tatsächlich ist die Art, in der Kommunikation bei Mobile Ambients modelliert wird, nicht besonders realistisch. Zum einen ist das der - aus den Reduktionsregeln ersichtliche - nicht-deterministische Charakter einer Kommunikation. Warten z.B. zwei Prozesse in einem Ambient auf einen Input und werden 2 Werte in den Äther dieses Ambients gestellt, so kann keine Aussage darüber getroffen werden, welcher Wert wo aufgenommen wird. Zum anderen enthält ein Paket immer ausführbaren Code, was bei wirklichen verteilten Systemen eher der Ausnahmefall ist (s. [3], Seite 13f). Eine Kommunikationsform, die nicht auf "Freisetzung" beliebigen Codes am Ziel basiert, wäre sicherlich wünschenswert.

Diese Erkenntnis weist meines Erachtens auf ein generelles Problem hin - Mobile Ambients sind zu generisch definiert. Ein Ambient z.B. kann ein Paket, ein Laptop, ein Speicherbereich oder eine Funkzelle sein. Auch wenn hierdurch das System weniger komplex ist, erscheint die Modellierung in manchen Fällen einfach nicht stimmig. Ein Laptop kann *nie* einen Speicherbereich betreten, bei Mobile Ambients ist dies aber zumindest theoretisch möglich. Modelliert man alle Aspekte von Hand (d.h. der hypothetische Laptop kann die Ambients $a_0, a_1, ..., a_n$ nicht betreten, da sie Speicherbereiche sind, wird das Modell schnell unüberschaubar. U.U. wäre hier eine Art 'Vererbungshierarchie' interessant, mit der man bestimmten Elementen feste Eigenschaften aufgrund ihrer Gruppenzugehörigkeit geben kann.

6.2. Schwächen der modalen Logik

An der modalen Logik stört vor allem die Tatsache, dass manche Teilaspekte bislang unerforscht sind, vor allem was die Logik angeht. Am gravierendsten ist das Fehlen von Namensbeschränkungen, was gerade bei der Modellierung von Sicherheitsaspekten eine enorme Erleichterung bieten würde. Auch der Modelchecker unterstützt nur einen Teilbereich der Logik, was seinen Nutzen teilweise einschränkt.

Diese Beschränkung auf eine Untermenge des Kalküls ergibt zwischen diesem und der Logik einen unschönen Bruch; schon beider Modellierung eines Systems müssen die Beschränkungen der Logik in Betracht gezogen werden. Damit wird z.B. eine Arbeitsteilung erschwert, da jeder am Prozess Beteiligte das Gesamtkonzept verstanden haben muss. Insbesondere die fehlende Namensbeschränkung führt dazu, dass Dinge, welche man im Kalkül an sich elegant modellieren könnte auf umständlichen Wegen dargestellt werden müssen um mit der Logik verwendet werden zu können.

Handelt es sich bei den obigen Problemen um Aspekte, die sich durch eine Weiterentwicklung lösen werden, so gibt es noch mindestens eine Schwäche, die direkt auf das Design zurückzuführen ist. Hat man eine Aussage vom Typ

$$P \neg \Diamond A,$$

so liest man diese zwar als "*P satisfies sometimes A*", tatsächlich aber ist gemeint "*P satisfies A sometimes in the future*", wie sich aus der Definition über die transitive Reduktion leicht erkennen lässt. Leider gibt es keine Umkehrung hierzu, die Aussagen über die Vergangenheit zulässt. Dies wäre der Fall, wenn es ein Pendant zu der Regel

$$P \neg A \neg P \neg \Diamond A$$

gäbe, welche es erlaubt, das Einfügen des *sometimes* wieder zurückzunehmen, zumindest partiell.

Welche Konsequenz hat dies? Widerspruchsbeweise werden im Zusammenhang mit *sometimes* unmöglich aufzulösen. Man betrachte das folgende abstrakte Beispiel:

Unsere Behauptung sei:

$$P \neg \Diamond A$$

D.h. für einen Widerspruch gehen wir aus von

$$P \neg \Diamond A$$

Ist P ungünstig definiert müssen wir bei irgendeinem Schritt mit der vorhin beschriebenen *inference rule*, ein *sometimes* einfügen, dass wir nie wieder loswerden, da

wir keine Aussagen über die Vergangenheit machen können. Damit wird es unmöglich eine Aussage herzuleiten, die einer anderen Vergangenheitsaussage widerspricht.

6.3. Fazit

Es mag so aussehen als wären Mobile Ambients ein wenig nützliches Konstrukt. Man muss eingestehen, dass Mobile Ambients in der Tat von einigen Problemen geplagt werden.

Trotz all dieser Nachteile ist jedoch der Ansatz von Mobile Ambients durchaus zu begrüßen. Prinzipiell ist zu sagen, dass eine logische Analyse eines Problems, die sich nach festen Regeln richtet immer ein guter Ansatz ist. Auch wenn es gegenteilige Stimmen gibt (s. Extereme Programming [9]), kann nur durch stichhaltige Entwürfe ein Programm als sicher gelten. Gerade bei kritischen Anwendungen im Finanz- oder Energiebereich sollte ein Funktionalitätsnachweis zwingend sein.

Ein logischer Ansatz beweist nicht nur, dass sich ein Programm so verhält wie geplant, sondern kann auch dazu dienen Probleme zu erkennen, die man bisher nicht als solche wahrgenommen hatte. Dies gilt insbesondere für Seiteneffekte paralleler Prozesse. Das Modell kann zwar nicht immer beweisen, dass im endgültigen Programm an dieser Stelle ein Fehler auftritt, man kann aber sehr wohl erkennen, an welchen Stellen eine unerwünschte Wechselwirkung auftreten könnte!

Zu guter Letzt sei an dieser Stelle noch einmal der Modelchecker erwähnt. Obwohl er lediglich für eine Untermenge der Logik funktioniert, ist der Modelchecker ein äußerst nützliches Werkzeug, da er eine automatische Prüfung von Aussagen erlaubt - für die Erstellung einer Software zur Überprüfung von Modellen ist dies geradezu essentiell.

6.4. Ähnliche Konzepte und Weiterentwicklungen

Mobile Ambients ist keine völlig neue Idee sondern vielmehr eine Weiterentwicklung bestehender Konzepte. Das π-Calculus (oder Kalkül) [X2] stellt eine sehr wichtige Grundlage der Mobile Ambients dar. Viele Konzepte des Letzteren sind aus diesem übernommen oder an es angelehnt.

Zum Beispiel werden Parallelität und inaktive Prozesse in beiden Kalkülen gleich ausgedrückt. Bei Namensbeschränkungen geht das sogar so weit, dass für die Definition auf das π-Kalkül explizit hingewiesen wird. Auch Konzepte wie Kanäle wurden stark vom π-Kalkül beeinflusst.

Interessanterweise ist eine andere Inspirationsquelle für Mobile Ambients die Chemical Abstract Machine. Dieses Verfahren hat nichts mit Computern zu tun, sondern beschreibt Zellmembranen, Lösungen und andere organische Vorgänge. Die Konzepte weisen jedoch eine Überschneidung auf, die von Cardelli und Gordon genutzt wurde [3].

Weitere Arbeiten, die mit Mobile Ambients verwandt sind wären:

- spi Calculus
- (distributed) join-calculus
- LLinda

Neben Vorgängern gibt es auch Techniken, die direkt aus den Mobile Ambients entstanden sind. Ein Beispiel hierzu wären die Boxed Ambients [5] oder die Secure Safe Ambients [10].

7. References

[1] Michelle Delio (2002): "Gates Finally Discovers Security", in WiredNews, http://www.wired.com/news/infostructure/0,1377,49823,00.html

[2] Flemming Nielson, Hanne Riis Nielson, René Rydhof Hansen, Jacob Gryholt Jensen (1999): "Validating Firewalls in Mobile Ambients" in CONCUR 1999: 463-477

[3] Cardelli, Luca, Gordon, Andrew (2000): "Mobile Ambients", in: Theoretical Computer Science, Special Issue on Coordination, D. Le Métayer Editor. Vol 240/1, June 2000. pp 177-213

[4] Milner, Robin, Parrow, Joachim, Walker, David (1992): "A Calculus of Mobile Processes, Part I", in: Information and Computation 199:1-77 (1992).

[5] M.Bugliesi, G. Castagna and S. Crafa (2001): "Boxed Ambients.", in: TACS 2001, LNCS n. 2215, pages 38-63, Springer 2001.

[6] Luca Cardelli, Andrew Gordon (2000): "Anytime, anywhere: Modal logics for mobile ambients" in: Proceedings POPL'00, ACM Press, pages 365--377, 2000.

[7] Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/logics-modal/

[8] Nelson Minar, Matthew Gray, Oliver Roup, Raffi Krikorian, Pattie Maes (1999): "Hive: Distributed Objects for Networking Things" in ASA/MA '99

[9] Ohne Verfasser: "What is Extreme Programming?" http://www.extremeprogramming.org/what.html

[10] Michele Bugliesi, Giuseppe Castagna (2001): "Secure Safe Ambients" in Proc. of the 28th ACM SIGPLAN-SIGACT Symposion on Principles of Programming Languages, 2001