Die Spezifikation des CAVIAR-Systems mittels Abstrakter Maschinen

Studienarbeit

24. Februar 1992

Tanja Zarnowka
Kuehnstr. 112
2000 Hamburg 70

Universität Hamburg
FB Informatik
Vogt-Kölln-Str. 30
2000 Hamburg 54

Inhaltsverzeichnis

1. Einleitung....4

2. Die Spezifikationssprache Z....5
2.1 Das statische Schema....6
2.2 Die Kombination von Schemata....6
2.3 Das _-Schema....8
2.4 Das dynamische Schema....8

3. Die Spezifikation des CAVIAR-Systems in Z....10
3.1 Die Anforderungsanalyse....10
3.2 Die Integritätsbedingungen von CAVIAR....11
3.3 Die statischen Eigenschaften des CAVIAR-Systems....12
3.4 Die Struktur des CAVIAR-Systems....12
3.4.1 Die Bibliothek der generischen Systeme....12
3.4.1.1 Ein Ressource-Benutzer-System....12
3.4.1.2 Die Hierarchie der Ressource-Benutzer-Systeme....15
3.4.1.3 Das generische System 'Pool'....17
3.4.1.4 Das generische System 'Diary'....18
3.4.2 Die Kombination der instanziierten Einzelsysteme zum Gesamtsystem....19
3.4.2.1 Instanziierung durch abstrakte Objekttypen....19
3.4.2.2 Kombination von Teilsystemen....20
3.4.3 Die dynamische Beschreibung in der Spezifikation....24

4. Die Spezifikationssprache der Abstrakten Maschinen....27
4.1 Die statische Beschreibung....28
4.2 Die dynamische Beschreibung....28
4.3 Der Kontext einer Abstrakten Maschine....29
4.4 Die Kombination von Abstrakten Maschinen....30

5. Die Spezifikation des CAVIAR-Systems in Abstrakten Maschinen....
....33
5.1 Gegenüberstellung von Sprachkonzepten in Z und Abstrakten Maschinen....33
5.1.1 Die statische Spezifikation im Vergleich....33
5.1.2 Die dynamische Spezifikation im Vergleich....35
5.1.3 Die Kombination von Subsystemen....36
5.2 Die Strukturierung des CAVIAR-Systems in Abstrakten Maschinen....36
5.3 Die Ausformulierung des Systems in Abstrakten Maschinen....37
5.4 Anforderungen an Abstrakte Maschinen....41
5.4.1 Spezialisierung....41
5.4.2 Instanziierung....43
5.4.3 Kombination....45

6. Erweiterungen des Systems....46
6.1 Beschreibung....46
6.2 Gesamtstruktur....47
6.3 Formulierung in Beispielen....48
6.3.1 Das HR-V-Subsystem mit Attributverwaltung....48
6.3.2 Das CR-M-Subsystem mit Attributverwaltung....56

7. Ausblick....57

Anhang A: Das System CAVIAR als Abstrakte Maschine 59

I. Ein allgemeines Ressource-Benutzer-System R-U 59

II. Spezialisierung des allgemeinen R-U-Systems 60
A. Die Ressourcen sind nicht teilbar.....60
B. Ein Benutzer kann maximal eine Ressource belegen 61
C. Ein Benutzer kann maximal eine nicht-teilbare Ressource belegen 62

III. Erstellen eines Teilsystems aus CAVIAR.....63
A. Das Hotelreservierung-Subsystem HR-V 63
B. Das Transportreservierung-Subsystem TR-V 63
C. Ein Pool-System 64
D. Das Tagung-Besucher-Subsystem M-V 64
E. Ein Diary-System 65
F. Das Konferenzraum-Tagung-Subsystem CR-M 66
G. Das Speisesaal-Tagung-Subsystem DR-M 67
H. Der Besucher-Pool V-P 68

IV. Das vollständige CAVIAR-System.....69
A. Der Visitor-Teil V-SYS 69
B. Der Meeting-Teil M-SYS 70
C. Das Gesamtsystem CAVIAR 70

Anhang B: Das Subsystem CR-M als Abstrakte Maschine mit Attributverwaltung.....71

1. Die Abstrakte Maschine CR-attributes.....71

2. Die Abstrakte Maschine M-attributes 73

3. Die Abstrakte Maschine CR-M-attributes.....75

Literaturverzeichnis.....77

1. Einleitung

CAVIAR ist die Fallstudie eines '**C**omputer **A**ided **V**isitor **I**nformation **A**nd **R**etrieval System' von B. Flinn und I. H. Sørensen [Hay 87], welches aus der Analyse eines manuell betriebenen Buchungssystems in einem großen Industriezentrum entstand. Dieses System befaßt sich mit der Registrierung und Weiterverarbeitung von Daten, welche durch die Planung und Durchführung von Veranstaltungen in diesem Zentrum entstehen. Es müssen die Anmeldungen von Besuchern registriert und die Besucher der gewünschten Tagung zugeordnet werden. Dabei werden nicht nur die stattfindenden Tagungen und Konferenzräume, sondern auch die Belegung der Hotelzimmer und Speiseräume verwaltet. Zusätzlich wird für den Transport des Besuchers zum Veranstaltungszentrum gesorgt.

Die gesamte durch die Verwaltung entstehende Information wurde manuell verwaltet. Die Aufgabe der Studie war die Untersuchung, inwieweit dieses System auf einen Computer übertragbar ist. Bei der Spezifikation des einzurichtenden, computerunterstützten Systems waren daher die wechselseitigen Beziehungen der gespeicherten Information, die Qualität der Bedienoberfläche und der Umfang der Daten, der zur Berechnung nötig sein würde, von besonderem Interesse. Die in dieser Arbeit dargestellte, formale Spezifikation des CAVIAR-Systems gibt einen Abriß der Entwicklungsschritte in diesem iterativen Konstruktionsprozeß, der durch zahlreiche Versuche der Modellkonstruktion und häufige Dialoge mit dem Kunden zur Klärung von Detailfragen geprägt wurde.

Eine formale Spezifikation soll in erster Linie mathematisch korrekt die Eigenschaften eines Systems beschreiben, ohne dabei etwas über die Art und Weise zu sagen, wie diese erreicht werden sollen. Es wird in einer Spezifikation das, **was** ein System tut, und nicht das, **wie** das System es ausführt, beschrieben. Diese Eigenschaft der Spezifikationssprache macht es überflüssig, den gesamten Programmcode durchzuarbeiten, um Information über das Verhalten des Systems zu bekommen. Weiterhin macht eine formale, mathematische Notation es möglich, die Spezifikation bis hin zum Programmcode zu entwickeln, indem für jeden Verfeinerungsschritt der Nachweis erbracht werden kann, daß diese Verfeinerung, und damit auch das im letzten Schritt entstandene Programm, die Spezifikation erfüllt.

In dieser Studie wurde die Spezifikationssprache Z gewählt, um diesen Anforderungen gerecht zu werden. Sie soll eine formale Systembeschreibung liefern, welche die Grundlage des Entwurfs bildet. Darüberhinaus soll die Spezifikation eine informelle Beschreibung der Systemeigenschaften bieten, um sie durch den Kunden mit seinen Anforderungen überprüfen zu können. Eine Spezifikation in Z umfaßt diese beiden Eigenschaften, indem ergänzend Texte als Kommentare zu den mathematischen Beschreibungen eingefügt werden. So kann der informelle Text zu Rate gezogen werden, um herauszufinden, welche Aspekte dargestellt werden und in welcher Beziehung sie zu den Anforderungen stehen. Andererseits kann der formale Text mit seinen präzisen Definitionen die Unklarheiten in der informellen Beschreibung lösen.

In dieser Arbeit wird ein Überblick über die Spezifikation des CAVIAR-Systems in Z gegeben. Danach wird das System in Abstrakte Maschinen (AM) übertragen. Es soll dabei untersucht werden, inwieweit man die Struktur des Systems in der Spezifikationssprache der Abstrakten Maschinen beschreiben kann. Dazu wird die Spezifikation in Z schrittweise in Abstrakte Maschinen umgeformt.

Mit der Sprache der Abstrakten Maschinen sollen die Nachteile der Spezifikationssprache Z verbessert werden, die im wesentlichen darin bestehen, daß bei etwas größeren Systemen die Anzahl der zu erstellenden Schemata sehr groß und damit unübersichtlich wird. Obwohl viele Schemata inhaltlich eng zusammenhängen, müssen sie doch getrennt beschrieben werden. In AMs werden mehrere Aspekte eines Systems in einer Maschine zusammengefaßt, sodaß die Anzahl der zu beschreibenden Maschinen geringer ist als die Anzahl der Schemata in Z für dasselbe System.

Um diesen Übertragungsprozeß zu dokumentieren, wird im folgenden Kapitel 2 zunächst die Sprache Z in einem kurzen Überblick vorgestellt. Hier werden die in CAVIAR verwendeten Konstrukte näher gebracht. Im Kapitel 3 werden die Eigenschaften und Strukturen des CAVIAR-Systems vorgestellt, die einen maßgeblichen Einfluß auf die Spezifikation in AMs haben. Eine Einführung in die Spezifikationssprache der AM mit ihren wichtigsten Strukturen folgt im Kapitel 4. Das CAVIAR-System wird im Kapitel 5 in AMs spezifiziert. Darin werden zunächst die Konzepte beider Sprachen verglichen, um dann die vollständige Formulierung in AMs durchführen zu können. Die dabei auftretenden, besonderen Anforderungen an AMs werden noch einmal näher untersucht.

In der Z-Spezifikation werden Besucher, Tagungen, Konferenzräume usw. nur als abstrakte Mengen betrachtet, die keine weiteren Eigenschaften besitzen und keine Information außer ihrer Existenz enthalten. Wie kann man schon in der Spezifikation festhalten, daß Besucher Namen und Adressen haben oder daß Hotelzimmer unterschiedlichen Komfort bieten? Das Kapitel 6 gibt eine Antwort darauf, indem eine Erweiterung der Objekte des CAVIAR-Systems mit Attributen an Beispielen dargestellt wird. Im abschließenden Kapitel 7 werden die über den Rahmen dieser Arbeit hinausgehenden Fragen erörtert. Das gesamte, in Abstrakten Maschinen spezifizierte CAVIAR-System ist im Anhang A enthalten.

2. Die Spezifikationssprache Z

Die Notation in Z wurde in den achtziger Jahren zur Spezifikation und zum Entwurf von Software entwickelt. Ein Standard ist im Reference Manual [Spi89] enthalten. In der vorliegenden Spezifikation in Z sind die wichtigsten Aspekte des Systemverhaltens von CAVIAR in elementarer Mengentheorie und Prädikatenlogik erster Ordnung beschrieben. Hinzu kommen natürlichsprachliche Beschreibungen, welche die formalen Definitionen erläutern. Im Gegensatz zu einer Spezifikation in einer höheren Programmiersprache bleibt die Spezifikation verständlich und der Leser muß sich nicht durch die Details des Quellcodes durchlesen, um herauszufinden, was die Spezifikation leistet. Damit steht Z als Mittler zwischen der abstrakten, aber informellen Spezifikation und einer Spezifikation im Quellcode einer Programmiersprache.

Grundlage der Strukturierung in Z ist das Schema, welches Variable einführt und in Prädikaten deren Beziehungen untereinander festlegt. Zu diesem Zweck ist ein Schema in einen Deklarations- und einen Prädikatenteil gespalten. Ein Schema kann *statische* (d. h. Zustandsraum und invariante Relationen auf dem Zustand) oder *dynamische* (d. h. Operationen zur Zustandsänderung) Aspekte eines Systems beschreiben.

2.1 Das statische Schema

Statische Schemata werden charakterisiert durch Benennung der Objekte und Eigenschaften, die für das zu spezifizierende System relevant sind, sowie die Beschreibung der invarianten Beziehungen zwischen diesen Variablen als Prädikate. Ein Schema zur Beschreibung der statischen Eigenschaften eines zu spezifizierenden Systems hat zum Beispiel die folgende Form:

```
┌─Class──────────────────────┐      Schema-Name
│ n_beendet,                 │      hier im Deklarationsteil
│ beendet :  POW(Student)    │      stehen Variable und deren Typ
├────────
│ beendet /\ n_beendet = {} &      hier im Prädikatenteil
│ card(beendet \/ n_beendet)        stehen die Invarianten
│                    <= max
└────────────────────────────┘
```

Hier wird der Zustand eines Systems beschrieben, der aufzeichnet, welche Studenten einer Klasse oder eines Kurses eine Übung beendet haben und welche nicht. Dies wird durch die Mengen 'beendet' und 'n_beendet' dokumentiert. Da ein Student nur in einem von beiden Zuständen sein kann, muß der Mengendurchschnitt leer sein. Ebenso darf die Gesamtmenge der registrierten Studenten die maximal festgelegte Anzahl für eine Klassenstärke nicht überschreiten. Die Menge 'Student' wird als gegeben vorausgesetzt. Die Potenzmenge von 'Student' ist der Typ beider Variablen, da für jede Variable eine Typdeklaration angegeben sein muß.

2.2 Die Kombination von Schemata

Es werden verschiedene Operatoren zur Verknüpfung von Schemata in Z zur Verfügung gestellt, wie Negation, Konjunktion, Disjunktion, Implikation, Äquivalenz, Subtraktion und Join. Ein spezifiziertes Gesamtsystem besteht nun aus vielen solchen Verknüpfungen von Teilsystemen. Dabei können die Schemata auch in komplexen Ausdrücken verknüpft werden, z. B.

$$Search = (Search_0 \land Success) \lor NotFound \lor Empty.$$

Dabei wird in $Search_0$ der Wert einer Eingabevariablen aus einer Tabelle gesucht und ausgegeben. Dies kann aber nur bei erfolgreicher Operation (Success gibt ein 'ok' aus) geschehen, d. h. wenn die Eingabevariable gefunden wird, sonst wird NotFound ausgeführt. Ist die Tabelle leer, kann nur Empty ausgeführt werden.

Die einzige in CAVIAR verwendete Verknüpfung von Schemata ist die Konjunktion. Dabei werden die Deklarationen zweier Schemata im Deklarationsteil zusammengefaßt, wobei alle doppelt vorhandenen Variablennamen eliminiert werden. Vorausgesetzt wird, daß die Variablennamen eindeutig sind, d. h., wenn sie in beiden Schemata gleiche Namen existieren, dann müssen sie auch denselben Typ haben. Die Prädikate beider Schemata der Konjunktion werden im Prädikatenteil konjunktiv verknüpft.

Sei

```
┌─Aleph──────────────────────┐      Schema-Name
│ x,y : Z                    │      x,y sind ganze Zahlen
├────────────────────────────┤
│ x < y                      │      wobei x kleiner als y sei
└────────────────────────────┘
```

Und sei

```
┌─Gimel──────────────────────┐      Schema-Name
│ y : Z                      │      y ist eine ganze Zahl
│ z : 1..10                  │      z liegt zwischen 1 und 10
├────────────────────────────┤
│ y = z * z                  │      wobei y gleich z² sei
└────────────────────────────┘
```

Bei einer Konjunktion der beiden Schemata kann eine Deklaration von 'y' entfallen, da sie in beiden Schemata vorhanden ist. So entsteht das folgende Schema:

```
Aleph /\ Gimel =
┌─Aleph-Gimel────────────────┐
│ x,y : Z                    │      x,y sind ganze Zahlen
│ z : 1..10                  │      z liegt zwischen 1 und 10
├────────────────────────────┤
│ x < y & y = z * z          │     x kleiner y und y gleich z²
└────────────────────────────┘
```

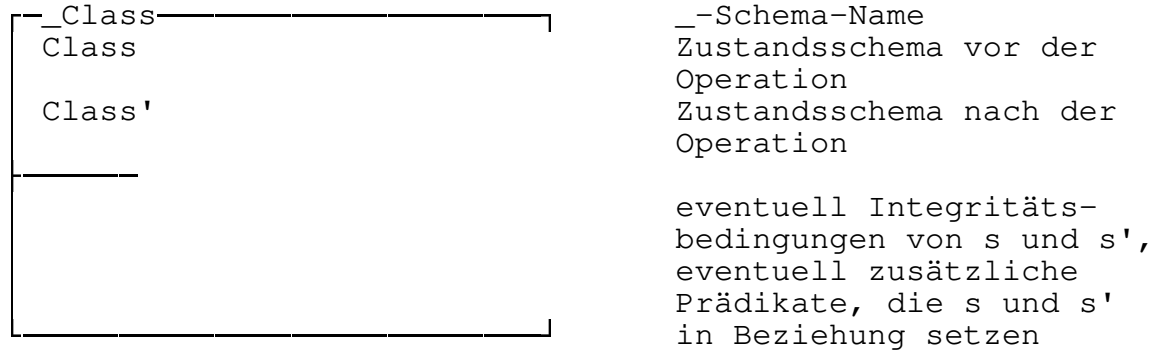
Alternativ dazu kann die Konjunktion zweier Schemata in Z durch die Angabe des einen Schema-Namens im Deklarationsteil des anderen Schemas definiert werden. Dann werden ebenso wie bei der zuvor beschriebenen Konjunktion die Variablen des eingebundenen Schemas im oberen Deklarationsteil eingefügt und die Prädikate dieses Schemas konjunktiv mit den neuen Prädikaten verknüpft. Das obige Schema für Aleph /\ Gimel hat somit die folgende, äquivalente Form:

```
Aleph /\ Gimel =
┌─Aleph-Gimel────────────────┐
│ Aleph                      │      Einbinden des Aleph-Schemas
│ z : 1..10                  │      zusätzlich z deklarieren
├────────────────────────────┤
│ y = z * z                  │      Beziehung zwischen y und z
└────────────────────────────┘
```

Diese zweite Form der Angabe von Schema-Namen in Schema-Definitionen wird zum einen in dynamischen Schemata benötigt, um das entsprechende _-Schema einzubinden. Zum anderen ermöglicht es, ein Schema zu spezialisieren, indem es in einem neuen Schema benutzt wird und mit weiteren Prädikaten versehen werden kann. Außerdem ist mit dieser Konstruktion die Kombination von mehreren Subschemata zu einem umfassenden Schema möglich.

2.3 Das _-Schema

Ein Schema, welches eine Operation darstellt, muß die Variablen s (Zustand vor der Operation) und s' (Zustand nach der Operation) desselben Typs deklarieren. Hierzu gibt es die Vereinbarung, ein solches Schema, welches die Zustandsvariablen vor und nach einer Operation zur Verfügung stellt, mit einem _ (eigentlich ein großes _, in diesem Text aus Darstellungsschwierigkeiten klein geschrieben) zu bezeichnen. Das _ ist als abkürzende Schreibweise vordefiniert und verknüpft konjunktiv ein Zustandsschema S und ein aus S erzeugtes Zustandsschema S'. Die Variablen für den Postzustand in S' erhalten dieselben Bezeichner wie im Präzustand, die nun aber zusätzlich mit einem Apostroph ' versehen werden. Durch das _-Schema werden implizit die Zustände vor und nach der Operation kombiniert. Ein _-Schema hat somit folgende ausführliche Form, die automatisch intern erzeugt wird durch das Einbinden von '_Class' in einem anderen Schema:



Falls alle Operationen des Systems gemeinsame Vor- und Nachbedingungen aufweisen, z. B. Erhöhung oder Nichtverändern einer Variablen, so kann das _-Schema erweitert werden, um diese Eigenschaften allgemein für das gesamte System festzulegen. Die geforderten Prädikate werden zusätzlich im Prädikatenteil des _-Schemas eingetragen, um die Variablen des Prä- und Postzustands zueinander in Beziehung zu setzen. Dann muß dieses _-Schema allerdings in der Spezifikation explizit notiert werden.

2.4 Das dynamische Schema

Zur Beschreibung des dynamischen Verhaltens wird für jede Operation des Systems ein gesondertes Schema erstellt, welches das zugehörige _-Schema einbinden muß. In einem solchen Schema werden die Zustände der Objekte vor und nach Ausführung der Operation in Beziehung gesetzt. Jede Operation kann nun durch die Erweiterung des _-Schemas mit Eingabe- und Ausgabe-Deklarationen sowie Vor- und Nachbedingungen spezifiziert werden.

Die Operation wird ebenfalls in zwei Teilen angegeben. Im Deklarationsteil werden die Variablen für das Schema deklariert und der Prädikatenteil enthält sämtliche Eigenschaften der Variablen und deren Beziehungen untereinander als Prädikate. Die Eingabevariablen werden dabei mit einem '?' nach dem Namen bezeichnet, die Ausgabevariablen mit einem '!'.

Im folgenden Beispiel wird die Einschreibung eines neuen Studenten in die zuvor definierte Klasse [siehe 2.1] spezifiziert. Dabei muß sichergestellt werden, daß der einzufügende Student noch nicht für diese Klasse eingeschrieben ist und daß noch ein Platz in dieser Klasse vorhanden ist. Da ein Student die Übung einer Klasse noch nicht bei seiner Immatrikulation fertiggestellt haben kann, wird er der Menge der noch nicht fertigen Studenten zugerechnet. Alle aufgeführten Vor- und Nachbedingungen werden durch Konjunktion zu einem Prädikat verbunden.

```
┌─Enrol_ok──────────────────────────┐  Name der Operation
│ _Class                            │  Vor- u. Nachbedingungen
│ s?:Student                        │  zusätzlich Variablen-
│                                   │  Deklaration,z.B. Ein/Ausgabe
├──────                             │
│ not(s? inc beendet\/n_beendet) &  │
│ card(beendet \/ n_beendet)        │  Prädikate
│                    < max &        │
│ beendet' = beendet &              │
│ n_beendet' = n_beendet \/ {s?}    │
└───────────────────────────────────┘
```

Ein gesamtes System wird durch Verbinden von ein oder mehreren Schemata statischer oder dynamischer Art, d. h. von Zustands-schemata, _-Schema sowie den Operationsschemata, beschrieben. Im vorliegenden Beispiel bilden die Schemata Class, _Class und Enrol_ok zusammen das System für die Aufzeichnung über den Arbeitsstand der Studenten einer Klasse sowie die Aufnahme eines neuen Studenten in eine Klasse.

3. Die Spezifikation des CAVIAR-Systems in Z

3.1 Die Anforderungsanalyse

Um die grundlegenden Mengen von Objekten der in Z spezifizierten Anwendung zu erkennen, erfolgt hier eine kurze Zusammenfassung der zu verwaltenden Aktionen im CAVIAR-System.

Ein Besucher meldet sich zunächst zu einer Tagung oder einer Firmenrepräsentation an. Er wird damit als Teilnehmer an einer Veranstaltung eingeschrieben. Für ihn muß auf Wunsch ein Hotelzimmer gebucht werden, sowie unter Umständen ein Verkehrsmittel, das ihn zur Firma bringt, sofern er nicht mit dem eigenen PKW anreist.

Jede Tagung findet in einem bestimmten Konferenzraum zu einer vorbestimmten Zeit statt. Sie erfordert bestimmte Arbeitsmittel, z. B. Projektoren oder pro Teilnehmer einen Schreibblock mit Bleistift. Diese Information über die Raumausstattung wird hier Sitzungsinformation genannt und pro Konferenzraum gespeichert.

Im Falle, daß die Tagung für mehrere Stunden oder Tage vorgesehen ist, benötigt sie für die Mittagspause einen Speisesaal an einem oder mehreren festgelegten Tagen. Um die Kapazität der Speisesäle auszunutzen, können die Teilnehmer mehrerer Tagungen einen Speisesaal teilen. Für die Belegung benötigt man genauere Information über das Mittagessen, z. B. die Anzahl der bestellten Mahlzeiten, hier mit Essensinformation bezeichnet. Diese wird pro Tagung gespeichert.

Somit muß das CAVIAR-System folgende, sehr unterschiedliche Mengen von Objekten verwalten, die für die beschriebenen Reservierungen benötigt werden.

Objekt	Kurzbez.	Beschreibung
Tagungen (Meetings)	M	stattfindende Tagungen
Besucher (Visitors)	V	angemeldete Besucher
Konferenzräume (Conference Rooms)	CR	Konferenzräume für Tagungen
Speisesäle (Dining Rooms)	DR	Speisesäle für Tagungen
Essensinformation (Lunch Information)	LI	Information über die Mahlzeiten
Sitzungsinformation (Session Information)	SI	Information über Tagungen
Hotelreservierung (Hotel Reservation)	HR	Reservierung eines Hotelzimmers
Transportreservierung (Transport Reservation)	TR	Transportreservierung zur Firma

Für die Formalisierung von Zeiträumen und Zeitpunkten wird angenommen, daß folgende drei Mengen mit unterschiedlichen Zeiteinheiten existieren:

Date	Tag, Datum Diese Zeiteinheit wird zum Buchen von Hotelzimmern eingeführt
Session	Zeitraum von Tagungen mit Datumskomponente, z. B. an mehreren Tagen verschiedene Stunden, die Einteilung kann fein in Stunden, halbe Stunden, oder auch grob in Vor- und Nachmittag erfolgen Diese Zeiteinheit wird zum Belegen der Konferenzräume benötigt
Time	Uhrzeit an einem bestimmten Tag Diese Zeiteinheit wird für die Transportreservierung benutzt

Zur Umwandlung von diesen Zeitmerkmalen auf die allen Zeitangaben gemeinsame Basisvariable 'Date' werden folgende Funktionen eingeführt:

date-of-session : Session --> Date

date-of-time : Time --> Date

Diese Funktionen sind total, da sie für jede definierte Zeit 'time' oder 'session' einen Wert aus 'Date' zurückliefern müssen.

3.2 Die Integritätsbedingungen von CAVIAR

Die wichtigsten Einschränkungen und Bedingungen der Beziehungen zwischen den Objekten ergaben sich aus dem Dialog mit dem Kunden wie folgt:

1. Zu einer Zeiteinheit kann in einem Konferenzraum nur eine Tagung stattfinden.

2. Zu einer Zeiteinheit kann eine Tagung mehrere Konferenzräume belegen.

3. Zu einer Zeiteinheit speisen alle Teilnehmer einer Tagung in demselben Speisesaal.

4. Zu einer Zeiteinheit können Teilnehmer von verschiedenen Tagungen denselben Speisesaal benutzen.

5. Zu einer Zeiteinheit kann ein Besucher nur an einer Tagung teilnehmen.

6. Zu einer Zeiteinheit kann eine Tagung mehrere Besucher haben.

7. Zu einer Zeiteinheit ist ein Hotelzimmer nur mit einem Besucher belegt und ein Besucher kann nur ein Hotelzimmer belegen.

8. Zu einer Zeiteinheit wird der Transport nur für einen Besucher reserviert und umgekehrt.

3.3 Die statischen Eigenschaften des CAVIAR-Systems

Aus diesen festen Beziehungen ergeben sich auf einleuchtende Weise die einzelnen Subsysteme von CAVIAR. Es wird jede in CAVIAR festgestellte Beziehung zwischen zwei Objekten als ein unabhängiges Subsystem in Z spezifiziert. Diese Beziehung steht in allen Subsystemen in funktionaler Abhängigkeit von der jeweils nötigen Zeiteinheit.

Systembezeichnung	Beziehung	Integritätsbedingung
CR-M	Konferenzraum-Tagung	1. und 2.
DR-M	Speisesaal-Tagung	3. und 4.
M-V	Tagung-Besucher	5. und 6.
HR-V	Hotelzimmer-Besucher	7.
TR-V	Transport-Besucher	8.

Jedes der hier aufgeführten Subsysteme wird zum Bilden des Gesamtsystems benötigt.

Die beschriebenen Objektmengen des CAVIAR-Systems stehen zwar in verschiedenen Beziehungen zueinander, doch durch weitere Analyse des Systems konnte man als Gemeinsamkeit aller Beziehungen extrahieren, daß jeweils eine Menge als Ressource und eine Menge als Benutzer dieser Ressource fungiert.

Systembezeichnung	Ressourcen	Benutzer
CR-M	Konferenzräume	Tagungen
DR-M	Speisesäle	Tagungen
M-V	Tagungen	Besucher
HR-V	Hotelzimmer	Besucher
TR-V	Transporte	Besucher

Wir haben damit eine Menge von Reservierungssystemen zu spezifizieren, deren Ressource-Benutzer-Beziehungen jeweils unterschiedlichen Einschränkungen unterliegen können. Zum Beispiel kann in einem Konferenzraum nur eine Tagung zur Zeit stattfinden, während sich durchaus die Besucher mehrerer Tagungen einen Speisesaal teilen können.

3.4 Die Struktur des CAVIAR-Systems

3.4.1 Die Bibliothek der generischen Systeme

3.4.1.1 Ein Ressource-Benutzer-System

Aus der grundlegenden, gemeinsamen Eigenschaft der CAVIAR-Subsysteme resultiert die Überlegung, ein allgemein zu verwendendes Ressource-Benutzer-System (Resource-User-System = R-U-System) zu erstellen, welches die Reservierungsart nicht einschränkt.

Ein solches Ressource-Benutzer-System ist ein System, das über die drei Mengen R, U und T parametrisiert ist. In dem R-U-System und allen im folgenden noch beschriebenen generischen Systemen der Bibliothek wird die Bezeichnung T als Variable für eine der drei Zeitmengen Time, Date oder Session benutzt. R bezeichnet die Menge der Ressourcen und U die Menge der Benutzer dieser Ressource. Ein allgemeines Ressource-Benutzer-System besteht damit aus einer Funktion von T in die Menge der Relationen zwischen R und U. So wird zu jeder Zeiteinheit aus T aufgezeichnet, welche Ressourcen gerade von welchen Benutzern belegt sind. In den spezifizierten Subsystemen von CAVIAR werden die Mengen R, U und T durch die jeweils als Ressource und Benutzer benötigte Menge sowie durch die spezifizierte Zeiteinheit instanziiert [siehe 3.3].

'R-U' in den Namen für die zugehörigen Schemata bezeichnet das allgemeine Ressource-Benutzer-System. Das Schema R-U spezifiziert den Zustand, in welchem eine Ressource mehrere Benutzer zur

selben Zeit haben und ein Benutzer mehrere Ressourcen gleichzeitig nutzen kann. Die pro Zeiteinheit bestehende Relation zwischen R und U wird mit der Zustandsvariablen 'ru' bezeichnet.[1]

```
┌─R-U──────────────────────────┐
│ ru    :T-->(R<->U)           │
│ in-use:T-->POW(R)            │
│ users :T-->POW(U)            │
│ ur    :T-->(U<-->R)          │
├──────                        │
│ !t:T•                        │
│    in-use(t) = dom(ru(t)) &  │
│    users(t) = rng(ru(t)) &   │
│    ur(t) = (ru(t))-1         │
└──────────────────────────────┘
```

Die Initialisierung für dieses R-U-Schema fordert für jede Zeiteinheit t die leere Menge als Relation ru(t).

```
Init-R-U = [R-U | rng(ru)={{}}]
```

Dies ist eine Kurzschreibweise für das Schema:

```
┌─Init-R-U─────────────────────┐
│ R-U                          │
├──────                        │
│ rng(ru) = {{}}               │
└──────────────────────────────┘
```

Im allgemeinen sind folgende Operationen für ein Reservierungssystem nötig:
- Buchen einer Reservierung
- Löschen einer Reservierung
- Löschen einer Ressource
- Löschen eines Benutzers

Die _-Schemata werden in einigen System-Spezifikationen durch Angabe von gemeinsamen Prädikaten für alle zu diesem System gehörenden Operationen erweitert. Dies geht über die implizite Definition von _-Schemata mit automatischer Deklaration der Variablen für die Vor- und Nachbedingungen hinaus [siehe 2.3].

Allen Operationen in R-U gemeinsam sind die Bezeichnungen der Prä- und Postzustände und eine Menge von Zeiteinheiten t? als Eingabe für die Operationen. Zu allen anderen Zeiteinheiten soll die Relation ru unverändert bleiben[2].

Eine weitere, zusätzliche Bedingung ist, daß die Menge aller Eingabezeiten t? nur Elemente aus der Menge T aller erlaubten Zeiten enthalten darf.

Das folgende _-Schema beschreibt diese beiden Integritätsbedingungen und stellt die Prä- und Postzustände zur Verfügung:

```
┌─_R-U─────────────────────────┐
│ R-U                          │
│ R-U'                         │
```

---

1 Der logische Allquantor wird mit dem Zeichen '!' aus dem ASCII-Zeichensatz dargestellt. Eine Verwechselung mit der Kennzeichnung für Ausgabevariable in einer Schema-Definition ist unmöglich, da die Stellung des Zeichens vor oder nach der Variable den Unterschied eindeutig kennzeichnet. Der Existenzquantor wird mit dem Zeichen '#' dargestellt.

2 Der Operator <<| bezeichnet die Domänensubtraktion, auch Anti-Restriktion genannt. t? <<| ru ist gerade die Menge aller Beziehungen ru(t), die zu den Zeiten t, welche nicht in t? liegen, existieren.
Definition: r : s <-> t & u inc t => u <<| r = id((dom(r) - u)) ; r , wobei der Operator ; die reverse Komposition und id die Identitätsrelation bezeichnet [Abr89b]

```
t? : POW(T)
─────────
t?<<|ru' = t?<<|ru
```

Mit dieser Spezifikation des _-Schemas kann auch die Buchungsoperation beschrieben werden:

```
R-U-Book──────────────────
_R-U
r? : R
u? : U
─────────
!t:t? •
  NOT((r?,u?) inc ru(t)) &
  ru'(t) = ru(t)\/{r?|->u?}
```

Die anderen für das R-U-System zu spezifizierenden Operationen R-U-Cancel, R-U-Del-Res und R-U-Del-User werden entsprechend beschrieben.

In dieser CAVIAR-Spezifikation werden redundante Variable benutzt. In der Definition des Schemas für ein generisches R-U-System werden zur Darstellung der Ressource-Benutzer-Beziehung vier Variable benutzt. Diese Variablen ru, ur, in-use und users beschreiben jede eine andere Sicht auf dieselbe Beziehung zwischen Ressourcen und Benutzern zu jeder Zeiteinheit. Aus der Zustands-komponente ru bzw. ur sind die jeweils drei anderen Variablenwerte ableitbar und daher redundant.

Eine Klausel von Invarianten legt die Beziehungen zwischen ru, ur, in-use und users fest. Diese Invarianten sind Teil jedes _-Schemas im Anfangs- als auch im Endzustand. Da dieses _-Schema in allen Operationsspezifikationen eingebunden ist, sind die Invarianten des _-Schemas darin ebenfalls enthalten. Somit sorgt die Spezifikationsweise in einer korrekten Implementation dafür, daß mit einer Vor- und Nachbedingung, welche die Veränderung einer der Variablen festlegt, die anderen, redundanten Variablen wegen der eingebundenen Invarianten ebenfalls "automatisch" mitverändert werden müssen.

Diese Veränderung muß also in der Spezifikation nicht explizit mit Prä- und Postprädikaten formuliert werden. Daher liegt der Vorteil der redundanten Variablen darin, daß in der weiteren Spezifikation die jeweils im Kontext anschaulichste Variable davon für eine Änderungsspezifikation gewählt werden kann.

3.4.1.2 Die Hierarchie der Ressource-Benutzer-Systeme

Dieses allgemeine Ressource-Benutzer-System muß weiter spezialisiert werden, um die verschiedenen Ressource-Benutzer-Beziehungen in den Subsystemen darstellen zu können:

a. R>U, d. h. Ressourcen sind nicht teilbar, sie können pro Zeiteinheit nur einem Benutzer zugeordnet werden, während ein Benutzer mehrere Ressourcen belegen darf.
Die Relation zwischen R und U wird als partielle Funktion von R nach U definiert, die Zustandsvariable ru besitzt daher die folgende Deklaration:
T --> (R +-> U)

Dieses R>U-System wird folgendermaßen aus dem R-U-System gebildet:

```
R>U = [ R-U | rng(ru) inc R+->U ]
```
[3]

```
Init-R>U = [ R>U | rng(ru)={{}} ]

_R>U = R>U & R>U'
```

Als Beispiel einer Operation wird die Buchungsoperation für R>U spezifiziert. Die Operation R-U-Book wird um die Bedingung erweitert, daß die zu buchende Ressource nicht bereits belegt sein darf, da sie nur von einem Benutzer gebucht werden kann.

```
R>U-Book = _R>U &
  [ R-U-Book | !t:t? • NOT(r? inc dom(ru(t))) ]
```

b. R<U, d. h. ein Benutzer kann pro Zeiteinheit maximal eine Ressource belegen, aber Ressourcen können von mehreren Benutzern geteilt werden.
Die Relation zwischen R und U wird als partielle Funktion von U nach R definiert, die Zustandsvariable ist ur : T --> (U +-> R).

Das R<U-System wird folgendermaßen als Schema beschrieben:

```
R<U = [ R-U | rng(ur) inc U+->R ]

Init-R<U = [ R<U | rng(ru)={{}} ]

_R<U = R<U & R<U'
```

---

[3] Dies ist wiederum die kurze Schreibweise für eine Schemadefinition.

Die Buchungsoperation für R<U erhält in diesem System die zusätzliche Bedingung, daß der Benutzer eine Ressource nur buchen darf, wenn er noch keine Ressource belegt hat.

```
R<U-Book = _R<U &
  [ R-U-Book | !t:t? • NOT(u? inc rng(ru(t))) ]
```

c. R=U, d. h. ein Benutzer kann pro Zeiteinheit maximal eine nicht-teilbare Ressource belegen. Dies beinhaltet die Eigenschaften beider Systeme R>U und R<U und ist daher eine Konjunktion der beiden Schemata.
Die Relation zwischen R und U wird als partielle, injektive Funktion von R nach U definiert, die Zustandsvariable ist ru : T --> (R >+> U).

Das R=U-System als Konjunktion der R>U- und R<U-Systeme zeigt folgendes Schema:

```
R=U = R>U & R<U
```

Dann ist auch der Initialisierungszustand wie bei den R>U- und R<U-Systemen zu beschreiben:

```
Init-R=U = [ R=U | rng(ru)={{}} ]
```

Die Buchungsoperation für R=U wird als Konjunktion der schon definierten Buchungsoperationen beschrieben. Man muß kein gesondertes _-Schema für die Operationen des R=U-Systems aufstellen, da die _-Schemata der R>U- und R<U-Systeme schon in den Operationsschemata der R>U- und R<U-Systeme enthalten sind.

```
R=U-Book = R>U-Book & R<U-Book
```

Die beschriebenen vier Arten des Ressource-Benutzer-Systems können als Basis für eine Spezifikationsbibliothek angesehen werden. Der Zusammenhang zwischen den vier Spezifikationen läßt sich wie folgt schematisch darstellen:

Abb. 1: Beziehungen zwischen den vier R-U-Systemen

In den folgenden Kapiteln werden weitere allgemeine, generische Systeme beschrieben, die ebenfalls zur Spezifikationsbibliothek gehören, um in CAVIAR instanziiert verwendet zu werden.

3.4.1.3 Das generische System 'Pool'

Pool ist ein System, welches die Verfügbarkeit von Objekten zu bestimmten Zeiten dem Gesamtsystem bekannt macht. Somit existiert eine Antwort auf die Frage, welche Objekte zu einem bestimmten Zeitpunkt zur Verfügung stehen.

Dieses Pool-System wird in CAVIAR zur Verwaltung der Tagungen und Besucher benötigt. Es soll sicherstellen, daß sich Besucher, die dem System bekannt gemacht wurden, nur für die ins System eingetragenen Tagungen anmelden können.

Die Parameter des Systems sind die Menge der Zeiteinheiten T und die Menge der Objekte X. Der Zustand wird nur durch die Funktionsvariable 'exists' beschrieben, welche jeder Zeiteinheit eine Menge von Objekten zuordnet.

```
┌─Pool──────────────────────┐
│ exists : T --> POW(X)     │
└───────────────────────────┘
```

Der Initialisierungszustand lautet wiederum:

```
Init-Pool = [ Pool | rng(exists)={{}} ]
```

Das _-Schema wird entsprechend den _-Schemata der R-U-Systeme erstellt:

```
_Pool = Pool & Pool'
```

Für Operationen, die keine Objekte hinzufügen oder löschen sollen, ist es nötig, ein Schema zu definieren, welches keine Änderungen der Zustandsvariablen 'exists' erlaubt, sondern nur Anfragen an den augenblicklichen Zustand dieser Variablen:

```
=Pool = [ _Pool | Pool=Pool' ]
```

Die Operationen zum Hinzufügen oder Löschen von Objekten lauten:

```
┌─Create────────────────────┐
│ _Pool                     │
│ t? : POW(T)               │
│ x? : X                    │
├───────────────────────────┤
│ exists' = exists <+       │
│   (%t:t? • exists(t)\/{x?}│
└───────────────────────────┘
```

```
┌─Destroy───────────────────┐
│ _Pool                     │
│ t? : POW(T)               │
│ x? : X                    │
├───────────────────────────┤
│ exists' = exists <+       │
│   (%t:t? • exists(t)-{x?} │
└───────────────────────────┘
```

3.4.1.4 Das generische System 'Diary'

Diary ist ein System, welches Information über Elemente einer Objektmenge aufzeichnet.

Es besteht aus den Funktionsvariablen 'info', welche die Information über einige Objekte zu gegebener Zeit speichert, und die daraus ableitbare Variable 'recorded', welche zu jeder Zeit alle Objekte, für die Information gespeichert ist, aufzeigt.

Das Diary-System wird benötigt, um Information über eine Ressource-Benutzer-Beziehung zu speichern, wie es für die Subsysteme CR-M und DR-M nötig ist. Hierbei werden für den Ablauf besonders wichtige Daten gespeichert, wie Essensinformation oder Information über eine Tagung.

```
┌─Diary──────────────────────────┐
│ info    : T-->(X+->I_X)        │
│ recorded: T-->POW(X)           │
├────────                        │
│ !t:T • recorded(t)=dom(info(t))│
└────────────────────────────────┘
```

Der Initialisierungszustand ist wie in den vorher beschriebenen Systemen zu spezifizieren:

```
Init-Diary = [ Diary | rng(info)={{}} ]
```

Operationen zum Hinzufügen und Löschen von Information für bestimmte Zeiteinheiten müssen spezifiziert werden. Die Bedingung, daß die Änderungen für einen bestimmten Zeitraum, also eine Menge von Zeiteinheiten, durchzuführen sind, gilt für beide Operationen und kann daher im _-Schema spezifiziert werden:

```
_Diary = Diary & Diary' & [ t?:POW(T) ]
```

Für die Operation des Hinzufügens darf zu keiner Eingabezeiteinheit das Objekt, für welches die Information gespeichert werden soll, schon aufgezeichnete Information besitzen. Unter dieser Zusatzbedingung kann die Zustandsvariable info' neu zusammengesetzt werden aus den alten Werten von info für alle Zeiteinheiten, die nicht in t? liegen, und für die Zeiteinheiten t? aus der neu gebildeten Funktion, welche info mit der Beziehung zwischen Objekt x? und Information I_X? überschreibt.[4]

```
┌─Add────────────────────────────┐
│ _Diary                         │
│ x? : X                         │
│ i? : I_X                       │
├────────                        │
│ (!t:t? •                       │
│    NOT(x? inc recorded(t))) &  │
│ info'=info<+(%t:t? •           │
│       info(t)<+{x?|->i?})      │
└────────────────────────────────┘
```

---

[4] Der Operator <+ bewirkt ein Überschreiben einer Relation nach folgender Definition [Abr89]:

p:s<->t & q:s<->t => p<+q = (dom(q)<<|p) \/ q

Der Operator % bezeichnet die Lambda-Konstruktion einer Funktion.

Für die Operation des Löschens sind die Bedingungen ähnlich. Hier muß die zu löschende Information allerdings nicht explizit eingegeben werden, sondern mit dem Löschen des Objekts wird jede Information, die dazu für die gegebene Zeit gespeichert ist, auch mit gelöscht.

Die eingegebene Menge von zu löschenden Objekten ist zeitabhängig. Die Domäne muß daher die Menge der Eingabezeiteinheiten sein. Zu allen Objekten muß zu allen Eingabezeiteinheiten Information gespeichert sein. Unter dieser Zusatzbedingung kann die Zustandsvariable info' neu zusammengesetzt werden aus den alten Werten von info für alle Zeiteinheiten, die nicht in t? liegen, und für die Zeiteinheiten t? aus der neu gebildeten Funktion für info, welche die Menge der Information auf die nicht zu löschenden Objekte einschränkt.

```
┌─Erase──────────────────────────┐
│ _Diary                         │
│ x? : T +-> POW(X)              │
├────────                        │
│ dom(x?) = t? &                 │
│ (!t:t? •                       │
│    x?(t) inc recorded(t?)) &   │
│ info'=info<+(%t:t? •           │
│       x?(t)<<|info(t))         │
└────────────────────────────────┘
```

3.4.2 Die Kombination der instanziierten Einzelsysteme zum Gesamtsystem

3.4.2.1 Instanziierung durch abstrakte Objekttypen

Ein Subsystem von CAVIAR wird durch Instanziierung eines in 3.4.1 vorgestellten Systems mit abstrakten Objekttypen gebildet. Diese generischen Systemspezifikationen werden hier im folgenden mit 'Bibliothekssystem' bezeichnet. Zur Instanziierung werden die Systemkomponentennamen durch die abstrakten Objekttypennamen substituiert und gleichfalls alle Objekte mit dem Indexnamen des instanziierten Schemas indiziert.

Die Definition eines instanziierten Schemas erfolgt für das einfache Beispiel des Subsystems zur Hotelreservierung, HR-V, auf folgende Art:

$$HR\text{-}V = R{=}U_{HR-V}[Date, HR, V]$$

Das HR-V-Subsystem stellt eine ausgezeichnete Instanz des R=U-Schemas dar, wobei die Systemkomponenten T, R und U als Parameter durch die Mengen Date, HR und V instanziiert werden. Der Index dieses so parametrisierten Schemas R=U, 'HR-V', wird als Index für alle in R=U deklarierten Variablen eingesetzt, um diese zu instanziieren. Man erhält somit in der ausführlichen Schema-Schreibweise für das Subsystem HR-V ein R=U-System, welches einige Umbenennungen enthält:

```
HR-V
ru_HR-V      : Date --> (HR <-> V)
in-use_HR-V  : Date --> POT(HR)
users_HR-V   : Date --> POT(V)
ur_HR-V      : Date --> (V <-> HR)
-----------------------------------
rng(ru_HR-V) inc HR +-> V &
rng(ur_HR-V) inc V +-> HR &
(!t:Date; r:HR •
    r : in-use_HR-V(t) <=> r : dom(ru_HR-V(t)) ) &
(!t:Date; u:V •
    u : users_HR-V(t) <=> u : ran(ru_HR-V(t)) ) &
(!t:Date • ur_HR-V(t) = (ru_HR-V(t))^-1)
```

Die Initialisierung und die Operationen des HR-V-Subsystems werden in gleicher Weise definiert:

$Init\text{-}HR\text{-}V = Init\text{-}R{=}U_{HR\text{-}V}[Date, HR, V]$

$Book\text{-}Hotel\text{-}Room_0 = R{=}U\text{-}Book_{HR\text{-}V}[Date, HR, V]$

$Cancel\text{-}Hotel\text{-}Room_0 = R{=}U\text{-}Cancel_{HR\text{-}V}[Date, HR, V]$

3.4.2.2 Kombination von Teilsystemen

a) Kombination instanziierter Systeme mit zusätzlichen Integritätsbedingungen

Um ein Subsystem von CAVIAR zu erstellen, werden die jeweils nötigen Bibliothekssysteme in der oben beschriebenen Weise instanziiert und zu einem neuen Schema kombiniert, wie dies auch beim CR-M-Subsystem der Fall ist:

```
CR-M
R>U_CR-M[Session,CR,M]
Diary_CR[Session,CR,SI]
-----------------------------------
in-use_CR-M = recorded_CR
```

$Init\text{-}CR\text{-}M = Init\text{-}R{>}U_{CR\text{-}M}[Session,CR,M] \;\&\; Init\text{-}Diary_{CR}[Session,CR,SI]$

Für das statische Schema CR-M werden R>U und Diary kombiniert, um dann durch eine zusätzliche Integritätsbedingung ergänzt zu werden, welche beide Systeme zueinander in Beziehung setzt. Es muß für die Kombination im CR-M-System ausgedrückt werden, daß die Menge aller im R>U-System verzeichneten Benutzer, in diesem Falle die Tagungen, identisch sind mit der Menge von Objekten, für die im Diary-System Information abgespeichert ist.

Auch für die Operationen werden zusätzliche Integritätsbedingungen bei der Kombination von Bibliothekssystemen benötigt. Beim Buchen eines Konferenzraums wird die Sitzungsinformation gespeichert. Dabei müssen die einzugebenden Zeiteinheiten in beiden kombinierten Systemen gleich sein. Ebenso muß die Menge der belegten Konferenzräume identisch sein mit der Menge von Konferenzräumen, für die Information gespeichert ist.

```
_CR-M = CR-M & CR-M'
```

```
┌─Book-Conf-Room0──────────────────┐
│ _CR-M                            │
│ R>U-Book_CR-M[Session,CR,M]      │
│ Add_CR[Session,CR,SI]            │
├──────                            │
│ t?_CR-M = t?_CR &                │
│ r?_CR-M = x?_CR                  │
└──────────────────────────────────┘
```

Von der Operation zum Annullieren der Buchung wird die Sitzungsinformation wieder gelöscht. Beim Annullieren werden alle Konferenzräume, die von einer bestimmten Tagung über einen bestimmten Zeitraum belegt wurden, wieder freigegeben. Hier ist die Integritätsbedingung über die Zeitvariable dieselbe wie in Book-Conf-Room$_0$. Als zweite Bedingung muß angegeben werden, daß für alle von der Operation betroffenen Zeiteinheiten die Menge der Konferenzräume, für die Information abgelegt ist, gleich der Menge der von den bestimmten Tagungen belegten Räume ist. Im Gegensatz zur Buchungsoperation ist bei dieser Stornierung keine Eingabevariable für die Konferenzräume angegeben, weil aus der einzugebenden Menge der Tagungen die Menge der belegten Konferenzräume hervorgeht.[5]

```
┌─Cancel-Conf-Room0────────────────────────────────┐
│ _CR-M                                            │
│ R>U-Del-User_CR-M[Session,CR,M]                  │
│ Erase_CR[Session,CR,SI]                          │
├──────                                            │
│ t?_CR-M = t?_CR &                                │
│ (!t:t?_CR-M • x?_CR(t) = ur_CR-M(t)⦇{u?_CR-M}⦈ ) │
└──────────────────────────────────────────────────┘
```

b) Kombination instanziierter Systeme mit Einführung neuer Zustandsvariable

Außer den zusätzlichen Integritätsbedingungen können bei der Kombination von Systemen auch noch weitere Ergänzungen zur vollständigen Spezifikation des Subsystems eingefügt werden, wie es beim DR-M-System der Fall ist. In diesem System muß die Anzahl der belegten Plätze pro Speisesaal berechnet werden, um nicht bei der Buchung die Grenze der maximal belegbaren Plätze zu überschreiten. Hierbei wird folgende Funktion zur Speicherung der endlichen Kapazitäten von Speisesälen als gegeben vorausgesetzt:

```
max-no : DR --> N,
          wobei N die Menge der natürlichen Zahlen darstellt
```

Die Anzahl der schon reservierten Plätze wird mit der Variablen 'rsvd' bezeichnet.

---

5 Die Zeichen 'Ç' und '¶' sind als speziell definierte Klammern zu lesen und bilden den Operator, der das Bild einer Relation liefert. Er ist folgendermaßen definiert [Hay87]:
R:X<->Y & S:POW(X) => RÇS¶ = {y:Y | (#x:S • (x,y) : R) }

```
DR-M
R<U_DR-M[Date,DR,M]
Diary_DR[Date,M,LI]
rsvd : T --> (DR +-> N)
-------
users_DR-M = recorded_DR &
(!t:Date • dom(rsvd(t))=in-use_DR-M(t) &
  (!r:in-use_DR-M(t) • rsvd(t)(r)<=max-no(r)) )
```

```
Init-DR-M = Init-R<U_DR-M[Date,DR,M] & Init-Diary_DR[Date,M,LI]
```

Außerhalb der Eingabezeiten bleibt die Variable 'rsvd' sowohl für die Buchungs- als auch für die Stornierungsoperation unverändert. Diese zusätzliche Bedingung läßt sich im _-Schema für beide Operationen spezifizieren. An derselben Stelle wird ebenfalls für die zusätzlich im _-Schema aufgenommene Eingabevariable 'amount' die Bedingung, daß sie nur für die eingegebenen Zeiteinheiten gelten soll, festgelegt.

```
_DR-M
_R<U_DR-M[Date,DR,M]
_Diary_DR[Date,M,LI]
amount? : T +-> N
-------
t?_DR-M = t?_DR &
dom(amount?) = t?_DR-M &
t?_DR-M <<| rsvd' = t?_DR-M <<| rsvd
```

```
Book-Dining-Room_0
_DR-M
R<U-Book_DR-M[Date,DR,M]
Add_DR[Date,M,LI]
-------
x?_DR = u?_DR-M &
(!t:t?_DR-M • rsvd(t)(r?_DR-M)+amount?(t) <= max-no(r?_DR-M) &
  rsvd'(t)=rsvd(t) <+ {r?_DR-M|->rsvd(t)(r?_DR-M)+amount?(t)})
```

```
Cancel-Dining-Room_0
_DR-M
R<U-Cancel_DR-M[Date,DR,M]
Erase_DR[Date,M,LI]
-------
(!t:t?_DR-M • x?_DR(t) = {u?_DR-M} &
  rsvd'(t)=rsvd(t) <+ {r?_DR-M|->rsvd(t)(r?_DR-M)-amount?(t)})
```

c) Kombination der Subsysteme zum Gesamtsystem

Für das Gesamtsystem CAVIAR ergibt sich durch die vorangegangenen Spezifikationen eine Bibliothek mit den generischen Systemen R-U, R<U, R>U, R=U, Pool und Diary. Jedes System dieser Bibliothek wird für die jeweils zu bildenden Subsysteme des CAVIAR-Systems mit jeweils anderen Objektmengen instanziiert. Es lassen sich sämtliche Subsysteme aus einem oder mehreren der vorgestellten, generischen Komponenten der Spezifikationsbibliothek durch Kombination der instanziierten Bibliothekssysteme nach folgendem Schema konstruieren:

Abb. 2: Konstruktion der Subsysteme aus den Bibliothekssystemen

Schließlich wird aus den Subsystemen das einheitliche Gesamtsystem von CAVIAR, im folgenden kurz 'CAVIAR-System' genannt, zusammengesetzt. Der Teil der Spezifikation, der die Verwaltung der Besucher betrifft, wird mit V-SYS bezeichnet. Als invariante Bedingung gilt, daß ein Besucher, der bei einer Hotel- oder Transportreservierung als Benutzer eingetragen ist, dem System schon im Besucher-Pool bekannt gemacht sein muß.

```
┌─V-SYS──────────────────────────────────────────────────────────────┐
│ V-P                                                                │
│ HR-V                                                               │
│ TR-V                                                               │
├────────                                                            │
│ (!d:Date • users_HR-V(d) inc exists_V(d) ) &                       │
│ (!t:Time • users_TR-V(t) inc exists_V(date-of-time(t)) )           │
└────────────────────────────────────────────────────────────────────┘
```

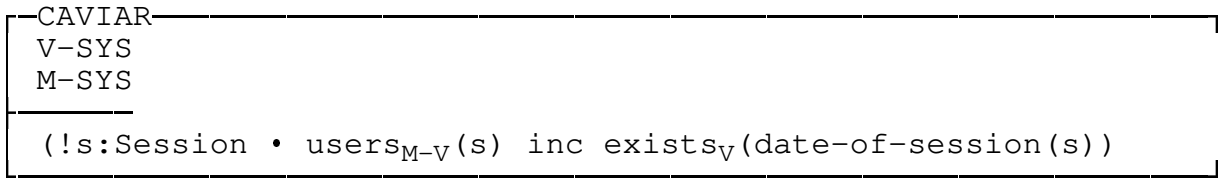
Desgleichen können sämtliche Subsysteme, welche die Verwaltung von Tagungen beinhalten, zum Schema M-SYS zusammengefaßt werden. Hier muß ebenfalls als invariante Bedingung vorausgesetzt werden, daß eine Tagung, welche Konferenzräume oder auch Speisesäle belegt, dem System zu dieser Zeiteinheit schon bekannt sein muß.[6]

```
┌─M-SYS──────────────────────────────────────────────────────────────┐
│ M-V                                                                │
│ CR-M                                                               │
│ DR-M                                                               │
├────────                                                            │
│ (!s:Session • users_CR-M(s) inc exists_M(s) ) &                    │
│ (!d:Date • users_DR-M(d) inc                                       │
│      union{s:Session | date-of-session(s)=d • exists_M(s)} )       │
└────────────────────────────────────────────────────────────────────┘
```

Die Kombination beider Schemata ergibt das CAVIAR-Gesamtsystem. Die Invariante, die beide Systeme V-SYS und M-SYS miteinander verbindet, muß sicherstellen, daß alle Besucher, die an einer Tagung teilnehmen (und daher in M-SYS verwaltet werden), schon dem System bekannt sein müssen (das bedeutet in V-SYS existieren).

---

6 Der mengentheoretische Ausdruck {D|P•t} umfaßt die Menge aller t, sodaß, wenn die Deklaration D gegeben ist, P weiterhin gilt [Hay87].



Die Initialisierung für CAVIAR ergibt sich aus der Konjunktion der Initialisierungen sämtlicher Subsysteme.

```
Init-CAVIAR = Init-HR-V & Init-TR-V & Init-M-V &
              Init-CR-M & Init-DR-M & Init-V-P
```

3.4.3 Die dynamische Beschreibung in der Spezifikation

Erweiterungen der _-Schemata von Bibliothekssystemen durch Angabe von Invarianten, die für alle Operationen des Systems zutreffen, gelten ebenfalls in den Subsystemen von CAVIAR, da sich diese Subsysteme aus den generischen Systemen kombinieren.

Als Beispiel enthält das M-V-System eine Instanz des Systems R<U. Die Operationen des M-V-Systems sind mit Hilfe von Instanzen der Operationen aus dem R<U-System spezifiziert. Da im R<U-System die Operationen wiederum mit Hilfe der Operationen aus dem R-U-System spezifiziert sind, enthält es die Invarianten des _R-U-Schemas und gibt diese Bedingungen an das M-V-System weiter.

Die in CAVIAR benötigten Operationen teilen sich ebenso wie die Zustandsschemata in zwei Gruppen. Die eine operiert auf dem durch M-SYS beschriebenen Zustand und läßt V-SYS unverändert. Die andere Gruppe von Operationen verändert V-SYS und beeinflußt M-SYS nicht.

a) Operationen für die Verwaltung von Tagungen

Diese Klasse von Operationen wird beschrieben durch:

```
M-OP = _CAVIAR & =V-SYS
```

wobei

```
_CAVIAR = CAVIAR & CAVIAR'
```

und

```
=V-SYS = [V-SYS & V-SYS' | V-SYS = V-SYS']
```

Um eine Tagung einzurichten und wieder abzusagen, werden folgende Operation benötigt:[7]

```
Create-Meeting = M-OP & Create-Meeting0 & =CR-M & =DR-M

Cancel-Meeting = M-OP & Cancel-Meeting0 & =CR-M & =DR-M
```

Zum An- und Abmelden eines Besuchers für eine Tagung sind die folgenden zwei Operationen spezifiziert:

```
Add-Visitor-to-Meeting = M-OP &
 Add-Visitor-to-Meeting0 & =CR-M & =DR-M
```

---

[7] Ein =-Schema in Z wird in der gleichen Weise wie ein _-Schema implizit definiert. Es kombiniert das _-Schema und legt fest, daß sich die Zustandsvariablen nicht ändern [Spi89].
Formal ausgedrückt: ÍState = [_State|State = State']
Die =-Schemata werden im weiteren nicht explizit ausgeführt, da sie implizit entsprechend ÍV-SYS gebildet werden.

```
Remove-Visitor-from-Meeting = M-OP &
 Remove-Visitor-from-Meeting0 & =CR-M & =DR-M
```

Das Belegen und Austragen der Konferenzräume geschieht durch:

```
Book-Conf-Room = M-OP & =M-V & Book-Conf-Room0 & =DR-M

Cancel-Conf-Room = M-OP & =M-V & Cancel-Conf-Room0 & =DR-M
```

Entsprechende Operationen werden auch für die Belegung von Speisesälen aufgestellt:

```
Book-Dining-Room = M-OP & =M-V & Book-Dining-Room0 & =CR-M

Cancel-Dining-Room = M-OP & =M-V & Cancel-Dining-Room0 & =CR-M
```

Um als letzte Operation das gleichzeitige Stornieren von Speisesälen und Konferenzräumen, die mit einer bestimmten Tagung assoziiert sind, durchzuführen, bedarf es einer eigenen Definition. Die Konjunktion beider Löschoperationen würde dies nicht ausführen können, da beide Operationen diejenigen Komponenten, auf denen sie nicht definiert sind, unverändert lassen. Es ist die letzte Definition einer Operation auf M-OP nötig:

```
Cancel-Meeting-Arrangements = M-OP & =M-V &
                        Cancel-Conf-Room0 & Cancel-Dining-Room0
```

b) Operationen für die Verwaltung von Besuchern

Diese zweite Klasse von Operationen verändert nur V-SYS und wird daher beschrieben durch:

```
V-OP = _CAVIAR & =M-SYS
```

Zwei Operationen werden eingeführt, um Besucher dem System bekannt zu machen und wieder zu entfernen:

```
Create-Visitor = V-OP & Create-Visitor0 & =HR-V & =TR-V

Destroy-Visitor = V-OP & Destroy-Visitor0 & =HR-V & =TR-M
```

Buchen und Stornieren eines Hotelzimmers für den Besucher geschieht mit:

```
Book-Hotel-Room = V-OP & Book-Hotel-Room0 & =V-P & =TR-V

Cancel-Hotel-Room = V-OP & Cancel-Hotel-Room0 & =V-P & =TR-V
```

Ebenfalls bestehen zwei Operationen, welche die Transportreservierung vornehmen:

```
Book-Transport = V-OP & =V-P & Book-Transport0 & =HR-V

Cancel-Transport = V-OP & =V-P & Cancel-Transport0 & =HR-V
```

c) Die Operation zum Entfernen eines Besuchers aus dem Gesamtsystem

Als letzte Operation soll das Löschen eines Besuchers aus dem System für eine bestimmte Menge von Zeiteinheiten bestimmt werden. Hierzu müssen die vorher bestimmten Löschoperationen für Besucher in einem Schema zusammengefaßt werden. Vom _-Schema für CAVIAR ausgehend werden diese Operationen hinzugefügt. Einzig die Systeme zur Verwaltung von Konferenzräumen und Speisesälen bleiben von dieser Operation unberührt, da in ihnen die Besucher keine Rolle spielen. Für diese Systeme muß daher das Í-Schema verknüpft werden.

```
┌─Delete-Visitor──────────────────────────────────────────────────────────┐
│ _CAVIAR                                                                 │
│ ═CR-M                                                                   │
│ ═DR-M                                                                   │
│ Cancel-Hotel-Room0                                                      │
│ Cancel-Transport0                                                       │
│ Remove-Visitor-from-Meeting0                                            │
│ Destroy-Visitor0                                                        │
├─────────                                                                │
│ x?V = u?HR-V = u?TR-V = u?M-V &                                         │
│ t?V = t?HR-V &                                                          │
│ t?TR-V = {d:t?V; t:Time | date-of-time(t)=d &                           │
│                           u?TR-V inc usersTR-V(t) • t } &                │
│ t?M-V = {d:t?V; s:Session | date-of-session(s)=d &                      │
│                           u?M-V inc usersM-V(s) • s }                   │
└─────────────────────────────────────────────────────────────────────────┘
```

4. Die Spezifikationssprache der Abstrakten Maschinen

Nach J. R. Abrial [Abr89a] bilden Abstrakte Maschinen(AM) eine Pseudo-Programmiersprache zur Spezifizierung von Software-Systemen. Die Programme in der Notation der AM können im Allgemeinen nicht direkt auf einem Rechner ausgeführt werden. Durch eine schrittweise Verfeinerung gelangt man zu einer "ausführbaren" Spezifikation. Diese Spezifikation, die zu einem ausführbaren Programm äqivalent ist, kann auf direkte Weise in eine höhere Programmiersprache oder auch Assembler Überführt werden. Die in AMs spezifizierten Pseudo-Programme sollen einer mathematischen Analyse unterzogen werden können. Um die Eigenschaften eines Software-Systems zu prüfen, muß die Notation eine Syntax für die formale Beschreibung besitzen und eine axiomatische Definition erhalten, um Theoreme prüfen zu können. In AMs wird die axiomatische Beschreibung durch die zugrundeliegende Mengentheorie und Prädikatenlogik ermöglicht. Zur Spezifikation von Operationen und deren Analyse wird das Kalkül der Prädikattransformer von E. W. Dijkstra [Dij76] verwendet.

Die Notation der AMs enthält einige Konstruktoren, die in fast allen Programmiersprachen zu finden sind, wie Zuweisung und Bedingung. Zusätzlich enthält sie auch unübliche, allerdings für den Systementwurf wichtige Konstruktoren, wie die Mehrfachzuweisung sowie parallele, partielle und nichtdeterministische Operationen. Diese führen dazu, daß die Spezifikationen in AMs nicht implementierbar oder auf einem Rechner ausführbar sind. Die auf der Basis der mehrfachen und der leeren Substitution anwendbaren Konstruktoren sind ebenso eine Verallgemeinerung als auch eine Vereinfachung der 'Guarded Commands' [Dij76] und werden als Generalisierte Substitutionen bezeichnet [Abr89c]. Jede Generalisierte Substitution definiert einen Prädikattransformer, welcher, angewendet auf eine Präkondition, die Weakest Precondition bildet.

Sequenzen und Schleifen sind in dieser Notation nicht vorgesehen. Derartige Konstrukte legen fest, wie etwas ausgeführt wird und sind somit charakteristisch für Programme in höheren Programmiersprachen. Auf Spezifikationsebene sind solche Konstrukte im Gegensatz zur Verfeinerung nicht nötig. Man muß zuerst einmal ausdrücken, was spezifiziert werden soll. Es gibt daher einige Techniken, um Pseudo-Programme mit Hilfe formaler Transformationen in ausführbare Programme zu überführen. J. R. Abrial beschreibt in seinem Artikel [Abr89c] einen solchen Zusammenschluß von Spezifikation und Programmierung. Zuerst wird ein mathematisches Modell der Daten mit den zugehörigen, möglichen Operationen zur Datenmodifikation aufgestellt. Zur Formalisierung des Modells wird Prädikatenlogik erster Ordnung auf die Mengentheorie angewendet. Am anderen Ende des Prozesses steht der Entwurf von Software, dessen Programmierkonzept das der Module ist. Ein mit Sequenzen und Schleifen erweitertes Modell wird Abstrakte Maschine genannt. Wird nun das Modell als Abstrakte Maschine ohne Sequenzen und Schleifen in den Operationen spezifiziert, so gelangt man durch schrittweise Verfeinerung zu einem Modul, welches ebenfalls als Abstrakte Maschine beschrieben wird. Dieses Modul hat die zusätzlichen Einschränkungen, daß jede Variable zu einer Menge gehört, die einen "Typ" formalisiert und daß seine Operationen nur programmierbare Instruktionen enthalten, welche auf boolesche Ausdrücke formalisierenden Prädikaten beruhen. Dieser Entwurf durch Verfeinerung läßt nach jedem Schritt eine neue Abstrakte Maschine entstehen, die weniger nicht-deterministisch als die vorhergehende Maschine ist.

Für die formale Darstellung und Bedeutung der Konstrukte von Abstrakten Maschinen wird in dieser Arbeit eine Version der AMs für den automatischen Beweiser 'devB' des B-Tools von Abrial benutzt [Abr90].

4.1 Die statische Beschreibung

Um den Zustand eines Systems zu beschreiben, gibt es in AMs die Konstrukte *variables*, in welchem die beteiligten Komponenten deklariert werden, und *invariant*, um die statischen Beziehungen im System festzuhalten. Alle benötigten Terme setzen sich aus Prädikatenlogik und Mengentheorie zusammen.

Eine AM, die ein schlichtes Buchungssystem beschreiben soll, welches nur die Anzahl n der gebuchten Objekte verwaltet, kann in einer ersten Entwicklungsphase folgendermaßen spezifiziert werden:

```
machine
    booking
variables
    n
invariant
    n : N
end
```

4.2 Die dynamische Beschreibung

Die dynamischen Eigenschaften eines Systems werden durch Operationen ausgedrückt. Zum Buchen und Freigeben von Objekten sind die Operationen 'book' und 'cancel' nötig. Dies sind die Namen der Operationen, die nach dem Gleichheitszeichen formal auf den deklarierten Variablen spezifiziert werden. Der Zweck einer Operation ist das Ändern des Zustands der Abstrakten Maschine innerhalb der von der Invarianten festgelegten Grenzen. In AMs werden Operationen in derselben Maschine wie die Zustandsvariablen in der Klausel *operations* spezifiziert.

Ein kurzes Beispiel:

```
machine
    booking
variables
    n
invariant
    n : N
operations
    book = ...;
    cancel = ...
end
```

Der Benutzer kann nur mit Hilfe der an dieser Stelle spezifizierten Operationen einer AM den Systemzustand erfragen oder verändern. Von außen kann er nicht direkt auf die Variablen zugreifen.

Aus der Benutzung von generalisierten Substitutionen zur Spezifikation von Operationen und der Semantik der Prädikattransformer folgt, daß in AMs implizit die **Frame Assumption** [Abr89a] gilt. Es werden nur die in den Operationen ausdrücklich zu substituierenden Systemvariablen verändert, alle anderen Zustandsvariablen der Maschine bleiben erhalten. Dies hat zur Folge, daß man nur die zu verändernden Werte einer Relation oder Funktion in einer Operationsspezifikation angeben muß.

4.3 Der Kontext einer Abstrakten Maschine

In dem simplen Buchungssystem kann es passieren, daß mehr Plätze gebucht werden sollen, als noch freie Plätze vorhanden sind. Darum soll es eine Funktion 'status' geben, die aufzeigt, ob die zuletzt durchgeführte Operation erfolgreich war. Zu diesem Zweck wird die Menge 'STATUS' definiert, welche die beiden Zustände 'erfolgreich' und 'nicht erfolgreich' enthält. Entscheidend für die Maschine ist die Existenz dieser Menge, nicht deren Elemente. Um aus diesem Grund die Menge außerhalb der gerade zu konstruierenden Maschine zu definieren, gibt es eine weitere, formale Struktur in der Sprache der Abstrakten Maschinen.

Die Struktur des Kontexts ist der Struktur einer Maschine vergleichbar und besitzt ebenso einen eigenen Namen. Ein solcher *context* gehört nicht zu genau einer Maschine, sondern hat den Vorteil, daß er von verschiedenen Maschinen gemeinsam genutzt werden kann. Eine Abstrakte Maschine kann einen Kontext nur sichtbar machen, aber nicht verändern. So spezifiziert ein Kontext die Umgebung einer oder mehrerer Maschinen, aber nicht deren eigentlichen Inhalt. Dieses Wissen über die Kontext-Umgebung dient ferner dazu, Beweise innerhalb des Gefüges der Abstrakten Maschinen durchzuführen.

Für das einfache Buchungssystem 'booking' wird der folgende Kontext sichtbar gemacht:

```
context
    booking
sets
    INDIVIDUAL;
    STATUS = {success, failure}
constants
    maxind
properties
    maxind : N;
    maxind <= card(INDIVIDUAL)
end
```

In der Klausel *sets* werden die Mengen der Umgebung zur Verfügung gestellt. Eine Menge kann durch Aufzählung der Elemente oder durch einen mengentheoretischen Ausdruck definiert werden. Werden dem Mengennamen keine Elemente zugewiesen, so wie im Beispiel INDIVIDUAL, dann ist dies eine **abstrakte** Menge, die implizit mit den Eigenschaften endlich und nicht-leer belegt wird. Die Mengen der natürlichen, ganzen, rationalen und realen Zahlen N, Z, Q und R sowie die booleschen Werte BOOL werden in einem vordefinierten Kontext implizit für alle Kontexte sichtbar zur Verfügung gestellt.

Konstante werden in der Klausel *constants* aufgelistet und können in der Klausel *properties* durch Prädikate näher beschrieben werden.

Mit der Klausel *sees* werden sowohl einer Maschine, als auch einem Kontext weitere Kontexte sichtbar gemacht. Diese Relation der Sichtbarkeit ist transitiv, sodaß eine Maschine, welche die Maschine 'booking' sichtbar gemacht hat, auch den Kontext 'booking' beinhalten würde.

Die Erweiterung der Maschine 'booking' mit der Statusfunktion würde nun folgendermaßen aussehen:

```
machine
    booking
sees
    booking
```

```
variables
    n,s
invariant
    n : N;
    s : STATUS
operations
    book(x) = ...;
    cancel(x) = ...;
    v <-- status = (v:=s)
end
```

4.4 Die Kombination von Abstrakten Maschinen

Die Kombination von Teilsystemen ist in AMs mit dem *uses*-Konstrukt zu erreichen, mit dessen Hilfe eine Maschine eine andere Maschine benutzen kann. 'Uses' bedeutet, daß eine benutzende Maschine alle Variablen und Operationen der benutzten Maschine in der folgenden Weise verwenden kann. In der aufrufenden Maschine können alle Zustandsvariable der benutzten Maschine gelesen werden. Wenn eine Operation der neuen Maschine eine Variable der benutzten Maschine verändern will, dann muß sie dies durch Aufruf der entsprechenden Operation aus der benutzten Maschine, zu der die Variable gehört. Dieses Prinzip, welches dem 'Benutzer' eines Moduls nicht erlaubt, die Daten dieses Moduls direkt zu verändern, nennt sich 'Hiding Principle' (= Prinzip des Versteckens) [Abr89a, Abr89c], da die Variablen vor direktem Zugriff geschützt werden. Das Prinzip des 'lax hiding' (= schwaches Verstecken) erlaubt zwar einen lesenden Zugriff auf die Variablen des benutzten Moduls, aber verhindert das direkte Verändern, welches nur durch die im benutzten Modul angebotenen Routinen möglich ist.

Jede Operation der benutzten Maschine kann zusätzlich mit der Klausel *promotes* in die neue Maschine eingefügt werden. Dadurch wird diese Operation in dem Interface dieser Maschine nach außen hin sichtbar und kann wiederum an andere Maschinen, die diese Maschine benutzen, weitergereicht werden. Diese mit *promotes* deklarierte Operation hat dieselben Rechte wie alle dort lokal deklarierten Operationen.

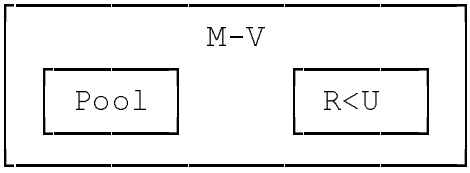
Zu beachten ist bei der Namensgebung, daß die Namen der beiden Maschinen verschieden sein sollten, sonst muß ein Präfix vor den Namen der benutzten Maschine gesetzt werden, der dann auch für alle Variablennamen und Operationsnamen automatisch hinzugefügt wird, um die Herkunft der Variablen weiterhin unterscheiden zu können. Bei Namensgleichheit von Operationen wird automatisch der Maschinenname als Präfix vor den Operationsnamen gesetzt.

Die Kombination von Maschinen wird am Beispiel des M-V-Systems erläutert:

```
machine M-V
sees
    M-V-context
uses
    R<U (Session, M, V)
    Pool (Session, M)
invariant
    !t.(t:Session => dom(ru_M-V(t)) inc exists_M-V(t))
```

```
operations
    Add-Visitor-to-Meeting0(session,m,v) =
        pre
            session : Session &
            m : M &
            v : V &
            !ti.(ti:session => (m:existsM-V(ti)))
        then
            R<U-Book(session,m,v)
        end;
    Create-Meeting0(session,m,v) = Create(session,m,v);
    ;(* weitere Operationen *)
end
```

Das M-V-System wird durch die Kombination von zwei Subsystemen gebildet. Durch das Zusammenfügen der Systeme wird die zusätzliche Bedingung nötig, daß eine Tagung, für die Besucher angemeldet sind, auch schon im Pool existieren muß. Diese Invariante ist in den Einzelsystemen Pool und R<U nicht ausdrückbar, sodaß sie zum Gesamtsystem M-V zugefügt werden muß. Beide Einzelsysteme gehören zur Bibliothek und werden hier in instanziierter Form verknüpft. Die folgende Darstellung soll deutlich machen, daß sich die Maschine M-V aus den beiden Maschinen Pool und R<U zusammensetzt.



Das Prinzip des 'lax hiding' hat für die Maschinen Pool und R<U zur Folge, daß die Variablen aus beiden Maschinen durch Invarianten in M-V zueinander in Beziehung gesetzt werden können und daß ihre Operationen in den Anweisungen für Operationen in M-V aufgerufen werden. Die zusätzliche Invariante für M-V setzt die Variablen 'ru' aus R<U und 'exists' aus Pool für die Maschine M-V zueinander in Beziehung. Es darf selbstverständlich keine Tagung geben, die zwar von mindestens einem Besucher belegt wird, die aber noch nicht im Pool existiert.

Durch das Benutzen von Pool und R<U in instanziierter Form werden sämtliche Operationen dieser Maschinen in M-V zur Verfügung gestellt. So ermöglicht das 'uses'-Konstrukt die Definition von Operationen in M-V, die im Grunde nur eine Umbenennung von Operationen aus R<U oder Pool sind, aber dieselbe Aufgabe erfüllen. Diese Umbenennung vernachlässigt zwar den engen Zusammenhang beider Operationen aus der benutzten und der neuen, benutzenden Maschine, aber macht die direkte Zugehörigkeit zu der Maschine M-V deutlich. Man erhält Operationen, die einzig und allein auf die Variablen von M-V zugreifen dürfen.

Als Beispiel solcher Umbenennung ist hier die Operation 'Create-Meeting$_0$' aufzuführen. Sie ist eine Umbenennung der Operation 'Create' aus der Maschine Pool und soll durch ihren Namen ausdrücken, daß sie zur Maschine der Tagungen- und Besucherverwaltung (Meeting-Visitor) gehört. Man hätte die Operation 'Create' auch mit einem *promotes* der Maschine M-V zugehörig machen können. Dann müßte in jedem folgenden Kontext, der die M-V-Maschine benötigt, der Name 'Create-Meeting$_0$' durch 'M-V.Create' ersetzt werden.

Eine Umbenennung, die etwas komplizierter ist, erfolgt für die Operation 'R<U-Book' in 'Add-Visitor-to-Meeting'. Hier ist eine zusätzliche Vorbedingung in der Klausel *precondition* unentbehrlich, damit die Operation zum Konsistenznachweis die zusätzliche Invariante von M-V erfüllt. Diese Invariante von M-V

$$!t_i.(t_i:session \Rightarrow (m:exists_{M-V}(t_i)))$$

besagt, daß alle Zeiteinheiten für die Operation 'Add-Visitor-to-Meeting' vom Typ 'session' sein müssen und die fragliche Tagung m vereinbarungsgemäß schon im Pool der Tagungen existieren muß. Gleichfalls gilt durch den Aufruf von R<U-Book im 'then'-Block die Vorbedingung

$$!t_i.(t_i:session \Rightarrow not((m,v):(ru_{M-V}(t_i))))$$

aus R<U-Book. Sie wird automatisch mit den hier aufgeführten Vorbedingungen konjunktiv verknüpft. Die Umbenennung der Operation geschieht aus demselben Grund wie für 'Create'.

5. Die Spezifikation des CAVIAR-Systems in Abstrakten Maschinen

5.1 Gegenüberstellung von Sprachkonzepten in Z und Abstrakten Maschinen

In Z werden statische und dynamische Beschreibungen in gesonderten Schemata spezifiziert. In übergeordneten Schemata werden statische mit statischen und dynamische mit dynamischen Schemata kombiniert, um so ein Gesamtsystem zu bilden. Dagegen werden in AMs statische und dynamische Beschreibungen in einem gemeinsamen Modul, der Maschine, zusammengefaßt. Man benötigt im Grunde nur eine Maschine, aber da das Prinzip der Modularisierung die Übersichtlichkeit erhöht, wird diese Maschine meist aus mehreren kleineren Maschinen zusammengesetzt.

Ein- bzw. Ausgabevariable werden in Z-Schemata mit einem '?' bzw. '!' besonders ausgezeichnet. In AMs werden die Eingabevariable als Parameter einer Operation beschrieben, die dann in der 'pre'-Deklaration einen bestimmten Wertebereich zugeordnet bekommen. Ausgabevariable sind die Ergebnisse einer Operation. Eine Ergebnisvariable e wird mit 'e <--' vor dem Operationsnamen ausgegeben. Die Unterscheidung mit einem besonderen Zeichen im Variablennamen ist in AMs nicht vorgesehen, könnte aber vom Systemdesigner ebenfalls definiert werden, da Namen für Variable frei wählbar sind.

Als inhaltlicher Unterschied der Spezifikationssprachen muß hervorgehoben werden, daß in Z für eine Operationsdefinition ein _-Schema angegeben werden muß, welches explizit angibt, für welche Werte sich eine Variable durch die Operationsausführung ändert und für welche Werte sie unverändert bleibt. Der in AMs zur Spezifikation von Operationen verwendete Formalismus erfüllt die Eigenschaft der **Frame Assumption**, die besagt, daß sich alle nicht aufgeführten Werte einer Relation oder Funktion nicht ändern [siehe 4.2]. Dadurch entfällt i. A. die Übertragung der _-Schemata aus Z in AMs, sofern sie nur diese Bedingung festlegen.

5.1.1 Die statische Spezifikation im Vergleich

Die Definition von Variablen ist in Z und AMs sehr ähnlich.

In AMs gibt es zwar keine Typbezeichnung, man kann aber dieselbe Eigenschaft auch als eine Teilmengen- und Element von-Beziehung ausdrücken.

Integritätsbedingungen werden in AMs in der dafür eingerichteten Klausel *invariant* beschrieben, während sie in Z im zweiten Teil der Schema-Definition stehen [siehe 2.1].

Invarianten in Z drücken z. B. Typzuordnungen von Variablen aus. Im statischen Schema des Pool-Systems bedeutet z. B.

exists : T --> POW(X)

eine totale Funktion von T in die Potenzmenge von X. D. h., daß in dem Schema 'Pool' mit der Zustandsvariablen 'exists' zu jeder Zeiteinheit eine Menge von Objekten, die alle aus der Menge X sein müssen, verwaltet wird.

In AMs wird eine solche Typdefinition als Invariante in der Klausel *invariant* angegeben:

exists inc T --> POW(X),

wobei 'exists' bereits in der Klausel *variables* deklariert sein muß und in der Klausel *invariant* die Mengen-Beziehung festgelegt wird.

Die Spezifikation der Initialisierungsoperation wird, wie alle Operationen, die in Z auf demselben Zustandsschema definiert sind, innerhalb der entsprechenden Maschine definiert. Hierfür ist eine eigene Klausel *initialisation* vorgesehen.

Die Möglichkeit in Z, mehrere redundante Variable für verschiedene Sichtweisen einer Beziehung in der Spezifikation anzugeben [siehe 3.4.1.1], kann nicht auf einfache Weise in die Darstellung von AMs übertragen werden. Es gibt grundsätzlich zwei Möglichkeiten, diese redundanten Variablen in AMs darzustellen:

a. Die redundanten Variablen werden als Invarianten dargestellt, weil sich diese Beziehungen als Konsistenzbedingungen nie ändern.
   In diesem Fall muß aber, im Gegensatz zu einer Z-Spezifikation, die Änderung aller anderen, redundanten Variablen explizit formuliert werden. D. h., wenn eine Operation auf einer dieser zugrundeliegenden Variablen ausgeführt wird, z. B. beim Buchen einer Ressource für einen User, daß alle anderen, davon abhängigen Variablen gleichzeitig durch ein weiteres Prädikat mit geändert werden müssen, um die Invarianten zu erhalten. Dies bedeutet mehrfache Prädikate für nur eine Zustandsänderung zu spezifizieren. Dieser Aufwand muß in jeder Maschine in allen Operationen, die eine dieser redundanten Variablen benutzen, durchgeführt werden.
b. Die redundanten Variablen werden als abgeleitete Variable dargestellt.
   Bei dieser Möglichkeit der Übertragung in AMs können die abgeleiteten Variablen aber nicht an Stelle der Zustandsvariablen in einer Operation benutzt werden, sondern es muß bei Änderungen immer diese eine Zustandsvariable für die Beziehung benutzt werden. Gerade das Benutzen verschiedener Variable für verschiedene Sichten ist aber der Zweck der redundanten Variablen: sie sollen zu einem besseren, inhaltlichen Verständnis verhelfen.

Insofern erübrigt sich durch diese Überlegungen das Einführen von redundanten Variablen, die im CAVIAR-System nur dem besseren Verständnis in der Formulierung von Operationen dienen sollten. Daher sind in-use, users und ur zwar als redundante Variable in CAVIAR zusätzlich definiert, sollen aber in den hier spezifizierten AMs nicht weiter verwendet werden. Der Sinn dieser redundanten Sichtweise auf die Daten, einen bequemeren Zugang zu den Daten zu erhalten, wäre durch die zusätzlichen, expliziten Aufrufe verloren. In AM's könnten solche redundanten Variablen nur für Abfragen genutzt werden und nicht in Definitionen stehen oder zur Veränderung der Daten dienen.

5.1.2 Die dynamische Spezifikation im Vergleich

Die dynamische Darstellung in Z und AMs ist verschieden. Jede Operation in Z wird durch ein eigenes, dynamisches Schema beschrieben, welches jeweils ein _-Schema enthalten muß. Die Zugehörigkeit zu dem Zustandsschema wird durch die Kombination dieses statischen Schemas im _-Schema erreicht. Innerhalb eines _-Schemas werden die Veränderungen von Variablen dadurch ausgedrückt, daß die Zustände vor und nach der Operation (mit z und z') bezeichnet werden und in der Invarianten der Zusammenhang beider Zustände beschrieben wird. In AMs werden Operationen durch generalisierte Substitutionen, deren Semantik durch Prädikattransformer definiert ist, spezifiziert. Alle zu einem System gehörenden Operationen werden nacheinander in der Klausel *operations* einer Maschine aufgelistet.

Anhand des Beispiels der Buchungsfunktion für das R-U-System werden beide Modellierungsarten zueinander in Beziehung gesetzt. In Z wird für alle Zeiteinheiten t? die Relation (r,u) der Funktion ru zugefügt, sofern sie noch nicht enthalten ist.

```
┌─R-U-Book──────────────────┐
│ _R-U                      │
│ r?:R                      │
│ u?:U                      │
├───────────────────────────┤
│ ! t:t? •                  │
│   not((r?,u?) inc ru(t)) &│
│   ru'(t) = ru(t) \/ (r?,u?)│
└───────────────────────────┘
```

Für die AMs beschreibt ein Teil der Klausel *operations* die gleiche Operation:

```
operations
    R-U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.((t_i:t) => (not ({ r |-> u }:ru(t_i)) ))
        then
            ru := %t_i.(t_i:t | ru(t_i) \/ {r |-> u}) \/ t <<| ru
        end;
    R-U-Cancel(t,r,u) = ...
```

Man sieht an diesem Beispiel, daß in Z bei Operationen die Vorbedingungen und die Beziehungen der Zustandsvariablen vor und nach der Operation konjunktiv verknüpft werden. In AMs ist für Vorbedingungen von Operationen die Struktur 'pre...then...else' vorgesehen. Mit 'pre' ('precondition', engl., = Vorbedingung, Präkondition) werden dabei alle Bedingungen zusammengefaßt, die zur Ausführung der Operation erfüllt sein müssen. Im ausführenden 'then'-Teil drückt die Zuweisung mit ':=' die Substitution der Variablen für die Relation 'ru' mit dem neu ermittelten Wert für 'ru' aus, sodaß man keine gesonderte Variable für den Zustand nach der Operation zur Beschreibung benötigt.

5.1.3 Die Kombination von Subsystemen

Beide Sprachen erlauben eine Kombination von Systemen mit Hilfe der Kombination von Schemata bzw. des 'uses'-Konstrukts. Durch Einbringen neuer Variable und Hinzufügen von Invarianten, um die Einzelsysteme zueinander in Beziehung zu setzen, wird eine Spezialisierung erreicht, da dieses Verbinden von Systemen konjunktiv durchgeführt wird und somit eine Verschärfung der Bedingungen darstellt. Auf dieser System-Ebene kann die Kombination, die in Z eingesetzt wurde, in die Abstrakten Maschinen übernommen werden.

In Z wird die konjunktive Kombination ebenfalls für die Bildung eines _-Schemas benötigt, welches die Spezifikation einer Operation überhaupt erst ermöglicht. Dieses _-Schema wird durch Kombination in ein Operationsschema eingebunden. Ein Operationsschema kann nun wiederum beliebig in einem anderen Schema kombiniert sein, sodaß die Schema-Kombination auf vielen Ebenen in Z eingesetzt wird. Dagegen wird in AMs aus allen diesen Einzel-schemata, die zueinander in Beziehung stehen, eine einzige Maschine erstellt. Hier wird keine Kombination benötigt, da eine Maschine schon die Konstrukte für Variable und die dazugehörigen Operationen bereit hält. Hier ist die einzige Anwendung der Kombination in der Spezialisierung zu sehen.

Eine in Z über einem _-Schema gebildete Operation gehört zu dem im _-Schema genannten Zustandsschema. Sie kann in weiteren Schemata kombiniert werden, wobei die Schemanamen im _-Schema übereinstimmen müssen. In AMs hingegen gibt es eine klare Kapselung der Operationen. Die in einer Maschine deklarierten Operationen können nur hier benutzt werden. Sollen sie in einer anderen Maschine verwendet werden, so muß der Designer sie dort explizit mit einer Import-Deklaration aufführen. Dadurch entsteht ein genau definiertes Interface der Aufrufgültigkeit von Operationen.

5.2 Die Strukturierung des CAVIAR-Systems in Abstrakten Maschinen

Die Struktur des CAVIAR-Systems wird aus der Z-Spezifikation in die Spezifikation in Abstrakten Maschinen übernommen. Dazu ist es nötig, daß jedes Subsystem eine eigene Maschinendefinition bildet. Die generischen Systeme der Spezifikationsbibliothek werden jeweils als eine Abstrakte Maschine nachgebildet und in verschiedenen Teilsystemen mit jeweils anderen Mengen instanziiert. Das Gesamtsystem wird aus diesen Einzelkomponenten gebildet. Der hierarchische Aufbau des CAVIAR-Systems, welches aus mehreren zusammengesetzten Systemen gebildet wird, ist in der Abb. 3 wiedergegeben. Jedes System aus CAVIAR, welches in AMs als eine Maschine spezifiziert ist, wird als ein Rechteck dargestellt. Ein solches System kann wiederum aus mehreren Maschinen zusammengesetzt sein, die dann als darin verschachtelte Rechtecke dargestellt werden. Ein generisches System, welches nur parametrisiert in anderen Maschinen verwendet wird, erhält eine gestrichelte Umrahmung.

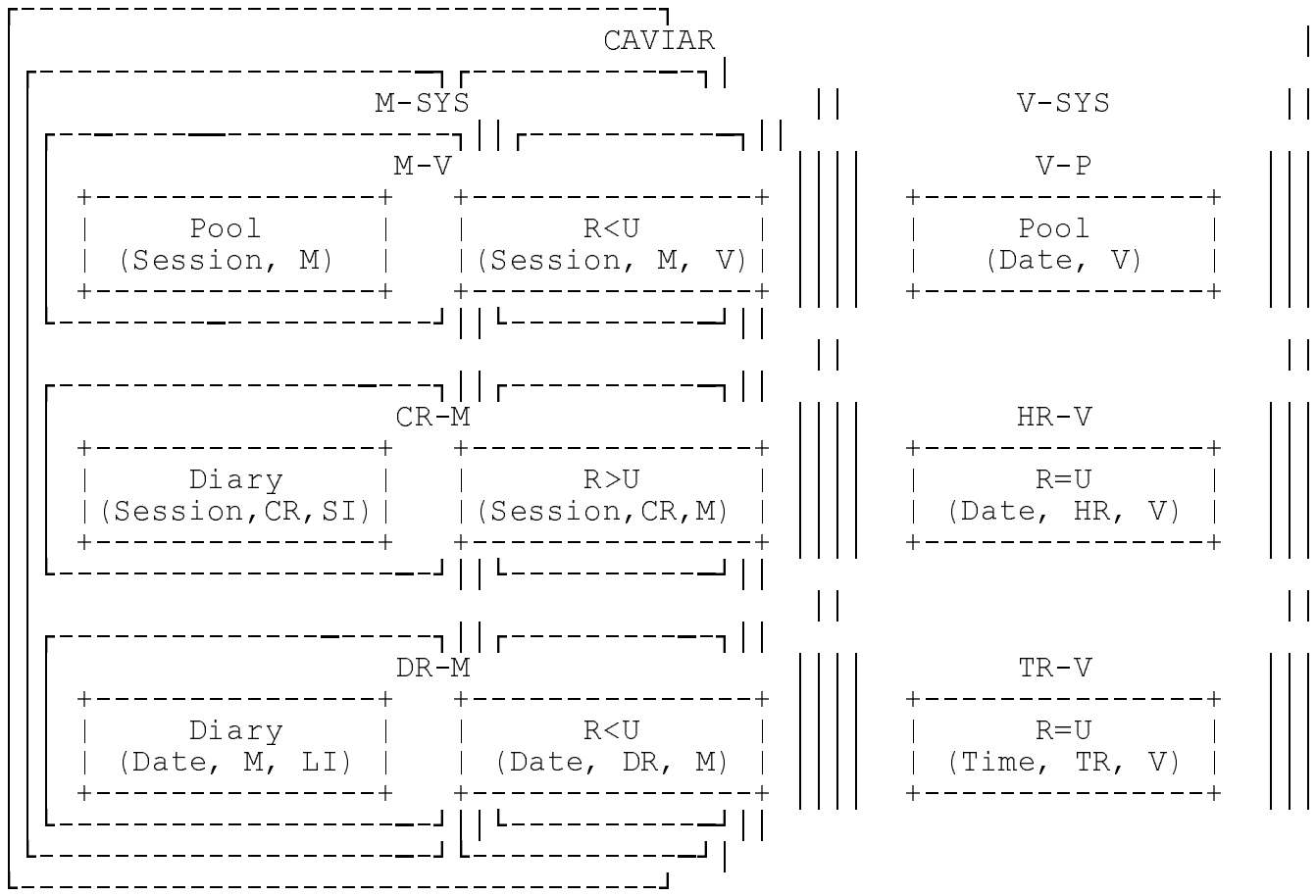


Abb. 3: Schachtelung der AMs aller Subsysteme

5.3 Die Ausformulierung des Systems in Abstrakten Maschinen

Es wird nur eine grobe Übersicht zum Konstruieren einer AM aus einer Schema-Definition in Z angegeben und kein präziser, allgemeiner Abbildungsalgorithmus aufgeführt. Dabei soll die Struktur des CAVIAR-Beispiels auch in den AMs erhalten bleiben, indem die in Z spezifizierten Zustandsschemata zu eigenen Maschinen werden, die zum Gesamtsystem kombiniert werden. Die _-Schemata und die Schemata zur Initialisierung werden als Operationen innerhalb dieser Maschinen der Teilsysteme integriert.

Die Beschreibung eines Zustands, einer Initialisierung, von Vor- und Nachbedingung für Operationen, sowie die Beschreibung einer Operation selbst werden nun exemplarisch in Z aufgeführt und in eine AM übertragen, um die dabei befolgten Regeln deutlich zu machen.

a) Beschreibung eines Zustands:

Z-Schema	Abstrakte Maschinen
R-U	machine R-U, Name des Schemas
ru : T->(R<->U) in-use: T->POW(R) users : T->POW(U) ur : T->(U<->R)	Deklaration aller Variablen in der Klausel *variables* und Definition in Klausel *invariant*
!t:T. in-use(t) = dom(ru(t)) & users(t) = rng(ru(t)) & ur(t) = $(ru(t))^{-1}$	Invarianten werden in der Klausel *invariant* beschrieben

Ein statisches Schema enthält viele Informationen, die auch in einer AM benötigt werden. Dazu gehört die Deklaration der Variablen, die Festlegung der Wertebereiche und auch die Auflistung der Invarianten. Die Relation ru beschreibt für alle Zeitpunkte die zu diesem Zeitpunkt existierenden Beziehungen zwischen Ressourcen und Benutzern. Davon abgeleitet beschreibt in-use für alle Zeitpunkte die belegten Ressourcen und users für alle Zeitpunkte die vorhandenen Benutzer. ur ist die Umkehrfunktion zu ru und enthält für alle Zeitpunkte die vorhandenen Beziehungen zwischen Benutzern und Ressourcen.

Die in diesem Beispiel benötigten Variablennamen werden in der Maschine R-U in der Klausel *variables* aufgelistet. Die Definition ihrer Wertebereiche ist eine Beschreibung von Invarianten und erfolgt daher in der entsprechenden Klausel *invariant* in AMs. In der AM R-U sieht das obige Schema folgendermaßen aus:

```
variables
    ru, in-use, users, ur
invariant
    ru  inc T --> (R <-> U);
    in-use  inc T --> POW(R);
    users   inc T --> POW(U);
    ur      inc T --> (U <-> R)
```

Auch die statischen Invarianten des Schemas von Z werden in derselben Klausel *invariant* der AM definiert. Dort werden in der Notation der AMs die folgenden Beziehungen festgelegt:

```
invariant
    !t.(t : T => (in-use(t) = dom(ru(t))));
    !t.(t : T => (users(t) = rng(ru(t))));
    !t.(t : T => (ur(t) = (ru(t))^-1))
```

In diesem Beispiel sind die Invarianten Konsistenzbedingungen, welche die Beziehungen zwischen den Funktionen ru, in-use, users und ur festlegen sollen [siehe auch 5.1.1].

b) Beschreibung einer Initialisierung:

Z-Schema	Abstrakte Maschinen
`Init-R-U = [R-U|rng(ru) = {{}}]`	Initialisierung von Variablen in der Klausel *initialisation*

Beim Start des Systems soll ru noch keine Beziehungen enthalten. Die Initialisierung der Relation ru mit der leeren Menge erfolgt in AMs in der Klausel *initialisation*.

c) Beschreibung von Vor- und Nachbedingung für Operationen:

Z-Schema	Abstrakte Maschinen
`_R-U`	
`R-U`	Zustand vor und nach
`R-U'`	einer Operation
`t?:POW(T)`	Eingabevariable
`t? <<| ru' = t? <<| ru`	Bereich der sich verändernden Relation [siehe 4.2.]

Die Beschreibung der _-Schemata erfolgte bereits in Kapitel 4.2. Es stellt Zustandsschemata für den Prä- und Postzustand zur Verfügung und deklariert darüber hinaus Eingabe- und Ausgabevariable, sowie Invarianten, die allen Operationen auf dem zugehörigen Zustandsschema gemeinsam sind. Da in AMs die Frame Assumption gilt, existiert keine gesonderte Klausel hierfür. Lediglich die zusätzlich deklarierte Variable t?, die für alle Operationen, die das _-Schema benutzen, als Eingabevariable nötig ist, muß in der AM für das R-U-System berücksichtigt werden, da sie angibt, für welche Menge von Zeitpunkten sich die Relation ru ändern darf. t? muß in der AM R-U daher für alle Operationen in der Präkondition als Variable deklariert werden.

d) Beschreibung einer Operation:

Z-Schema	Abstrakte Maschinen
`R-U-Book`	Name der Operation in R-U
`_-R-U`	*uses* der benötigten AM
`r?:R`	*pre* der entsprechenden Funktion
`u?:U`	
`!t:t?.(NOT((r?,u?):ru(t))&`	*pre* der entsprechenden Funktion
`ru'(t) = ru(t) \/`	Ausführung der Operation im *then*
`{(r?,u?)} )`	

Operationschemata wie R-U-Book, welche die _-Schemata benutzen und konkrete Operationen beschreiben, werden in die AM R-U als Operationen integriert.

Das Schema _R-U, welches hier für R-U-Book benutzt wird, gibt es in AMs nicht als eigenständige Maschine. Es ist implizit in der Maschine R-U enthalten, indem dort Operationen spezifiziert sind, die eine Änderung von Variablen bewirken [siehe 4.2]. Allerdings müßte jedes andere Schema, welches hier ebenfalls zur Kombination aufgeführt sein könnte, in der entsprechenden Maschine in der *uses*-Klausel aufgeführt werden.

Die Eingabevariablen r? und u? werden, im Gegensatz zur Eingabevariablen t?, nur im Schema R-U-Book benötigt. Sie müssen in der AM in der Präkondition der entsprechenden Operation vereinbart werden.

Um für das Buchen eine Beziehung zwischen r und u einzufügen, wird in Z die Vorbedingung und die Beziehung zwischen Prä- und Postzustand in Form eines Prädikats definiert:

```
!t:t? . NOT((r?,u?):ru(t) ) & ( ru'(t)=ru(t) \/ {r?|->u?} )
```

Für alle Eingabezeitpunkte t aus der Menge aller gewünschten Zeitpunkte t? soll die Beziehung r?|->u? eingefügt werden, wenn sie nicht schon vorhanden ist.

Konjunkte, die sich auf Eingabevariablen beziehen und nicht Vor- und Nachzustände in Beziehung setzen, werden als Präkondition einer Vorbedingungssubstitution ausgedrückt. Daher wird die bedingende Eigenschaft NOT((r?,u?) : ru(t)), die prüft, ob eine Beziehung zwischen r? und u? für die Zeitpunkte t? schon existiert, in AMs als Vorbedingung für die Operation R-U-Book ausgedrückt und muß nicht, wie in Z, in der Operation selbst konjunktiv verknüpft werden. Dieses Prädikat wird darum in der 'pre'-Klausel der Operationsdefinition wiedergegeben als:

$$!t_i.((t_i:t) \Rightarrow (\text{not } (\{ r \mid\!\!-\!\!> u \}:ru(t_i))))$$

Hier entsprechen die Variablen t, r und u den Eingabevariablen t?, r? und u? in Z.

In Z wird für eine Operation, die eine Relation ändern soll, ein logischer Ausdruck benutzt, um den Postzustand mit Hilfe eines Terms über dem Präzustand zu beschreiben. In AMs muß in der Operationsdefinition der Funktion ru ein bestimmter Wert direkt zugewiesen werden, weil es sich um generalisierte Substitutionen handelt. Somit muß in AMs eine neue Funktion aus der Vereinigung aller vorhandenen Werten der Relation mit allen neu hinzuzufügenden Werten gebildet werden.
Die obige Operation für R-U-Book lautet dann in AMs:

$$ru := \%t_i.(t_i:t \mid ru(t_i) \setminus\!/ \{r \mid\!\!-\!\!> u\}) \setminus\!/ t \ll\!\mid ru$$

Es wird zur Neubildung von ru für alle t_i aus t die Relation (r,u) der Menge der vorhandenen Relationen $ru(t_i)$ zugefügt und mit der Menge der vorhandenen (r,u) für alle t_i nicht in t vereinigt. Die Spezifikation enthält dabei absichtlich die Funktion des Überschreibens vom Funktionswert (r,u), wenn er für einige t_i schon vorhanden war.

Damit wird die AM R-U mit der Operation R-U-Book folgendermaßen spezifiziert:

```
machine
    R-U
variables
    ru,
    in-use, users, ur
invariant
    ru  inc T --> (R <-> U);
    in-use  inc T --> POW(R);
                ;(* = alle zur Zeit belegten Ressourcen *)
    users   inc T --> POW(U);
                ;(* = alle zur Zeit vorhandenen Benutzern *)
    ur      inc T --> (U <-> R);
                ;(* "Umkehrfunktion" zu ru
                 = alle zur Zeit vorhandenen Beziehungen
                 zwischen Benutzern und Ressourcen *)
    !t.(t : T => (in-use(t) = dom(ru(t))));
    !t.(t : T => (users(t) = rng(ru(t))));
    !t.(t : T => (ur(t) = (ru(t))~1))
initialisation
    ru := %t.(t : T | {} );
operations
    R-U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.((t_i:t) => (not ({ r |-> u }:ru(t_i)) ))
        then
            ru := %t_i.(t_i:t | ru(t_i) \/ {r |-> u}) \/ t <<| ru
        end;

end
```

5.4 Anforderungen an Abstrakte Maschinen

Die Strukturierungsmerkmale Spezialisierung, Instanziierung und Kombination sind besonders typisch für dieses Beispiel und werden daher im Vergleich beider Sprachen näher untersucht. Dabei wird deutlich, welche Anforderungen, die aus dem Beispiel des CAVIAR-Systems erwachsen, zwar noch nicht von der Sprache der Abstrakten Maschinen erfüllt werden, aber wünschenswert wären.

5.4.1 Spezialisierung

In Z gibt es kein eigenständiges Konzept der "Spezialisierung". Es ist aber möglich, ein Schema zu spezialisieren, indem es kombiniert wird. Dabei können die Invarianten und Operationen mit zusätzlichen Vorbedingungen eingeschränkt werden, sodaß ein neues Schema entsteht. Eine spezialisierte Operation verhält sich für den Benutzer annähernd wie die ursprüngliche Operation. Die Anfangszustände von spezialisierten, dynamischen Schemata können eingeschränkt sein, wodurch das spezialisierte Schema partieller ist und der Nichtdeterminismus reduziert sein kann. Somit terminiert die spezialisierte Operatione auf einer Teilmenge der Anfangszustände, in denen die ursprüngliche Operation terminiert.

Eine Spezialisierung hat verschiedene Auswirkungen:
- Die Invarianten werden im spezialisierten Schema verschärft, d.h. daß sie den gültigen Zustandsbereich weiter einschränken. Die verschärften Invarianten implizieren die Invarianten des Ausgangsschemas.
- Operationen werden spezialisiert, d. h. ihre Domäne wird eingeschränkt.
- Neue Zustandsvariable zur Beschreibung des spezielleren Sachverhaltes können hinzugefügt werden, die eventuell das Einfügen neuer Operationen (z. B. für das Ändern dieser neuen Zustandsvariablen) nötig machen.
- Um einer Spezialisierung zu genügen, dürfen die neu hinzugefügten Operationen die Anfangszustände des zu spezialisierenden Systems nicht erweitern.

Eine Spezialisierung von Maschinen kann mit Hilfe des *uses*-Konstrukts modelliert werden. Es wird die zu spezialisierende Maschine in der neuen Maschine mit *uses* eingebunden und erlaubt so, alle Variablen und Operationen aufzurufen. Dadurch können neue Operationen spezifiziert werden, die zusätzliche Vorbedingungen enthalten, um eine Spezialisierung der beschriebenen Objekte und Operationen zu erreichen.

Beispiel: In Z ist

```
R>U = [R-U | rng(ru) inc R+->U]
```

die Spezialisierung des Zustands eines allgemeinen R-U-Systems [siehe 3.4.1.1] auf nicht-teilbare Ressourcen. Die Beziehung ru wird von einer Relation zwischen R und U zu einer partiellen Funktion auf denselben Mengen geändert, wodurch der Wertebereich eingeschränkt wird.

In Z wird der Initialisierungszustand für das R>U-System angegeben mit

```
Init-R>U = [R-U | rng(ru) = {{ }} ]
```

Der Zustand wird in der AM R>U initialisiert mit der Klausel

```
initialisation
    rng(ru) = {}
```

Für die Operationen in Z gelten die statischen Invarianten des zu spezialisierenden Schemas weiterhin. Um die Invarianten des spezialisierten Schemas zu erhalten, müssen den Operationen, die die Invarianten des zu spezialisierenden Schemas erfüllen, neue Prädikate hinzugefügt werden.

Sei

```
_R>U = R>U /\ R>U'
```

dann wird die Buchungsoperation für das R>U-System angegeben mit

```
R>U-Book = _R>U /\ [R-U-Book | ! t:t?. not(r?:dom(ru(t)))]
```

Die Operation R>U-Book erhält hier das zusätzliche Prädikat
! t:t?. not(r?:dom(ru(t))),
welches verhindern soll, daß Beziehungen zwischen Ressourcen und Benutzern eingefügt werden, wenn die Ressource schon in einer Beziehung zu irgendeinem Benutzer steht. Dies ist gerade die Invariante der Spezifikation von R>U, die sicherstellt, daß eine Ressource nur maximal einen Benutzer haben darf.

In AMs muß die Operation unter Verwendung der Operation aus der 'uses'-Maschine mit Definition der zusätzlichen Vorbedingungen neu geschrieben werden und erhält, wie in Z, einen neuen Namen. Im Gegensatz zur Spezifikation in Z müssen Prädikate, die zur Definition der Operationsparameter dienen und in der AM R-U bereits spezifiziert wurden, in dieser spezialisierten AM R>U ebenfalls aufgeführt werden. Das Prädikat der spezialisierten Operation wird hinzugefügt. In der Notation der AMs lautet es:

```
!t_i.(t_i:t => not(r:dom(ru(t_i))))
```

Diese Buchungsoperation wird in der AM R>U unter der entsprechenden Klausel *operations* eingefügt. Die folgende Maschine R>U entsteht:

```
machine
    R>U
uses
    R-U
invariant
    rng(ru) inc R +-> U
    ;(* partielle Funktion als zusätzliche Einschränkung zu R-U *)
initialisation
    rng(ru) = {}
operations
    R>U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.(t_i:t => not(r:dom(ru(t_i))))
                ;(* neue Vorbedingung zur Spezialisierung *)
        then
            R-U-Book(t,r,u)
        end;
    ;(* weitere Operationen werden entsprechend spezialisiert *)
end
```

5.4.2 Instanziierung

In der Z-Beschreibung des CAVIAR-Beispiels werden Schemata durch Parameterangaben instanziiert. Zur Unterscheidung der neu entstandenen Schemata und Zustandsvariable werden diese mit einem entsprechenden Index versehen.
Beispiel:

$$HR\text{-}V = R{=}U_{HR\text{-}V}[Date,HR,V]$$

Das HR-V-Subsystem ist eine Instanz des R=U-Systems [siehe 3.4.2.1]. Dies bedeutet, daß darin alle Variablen von R=U mit dem Index 'HR-V' versehen werden und die Mengenbezeichner T, R und U jeweils durch Date, HR und V ersetzt werden. In der gleichen Weise werden auch die Bezeichnungen in den instanziierten Schemata zur Initialisierung und zur Operationsdefinition ersetzt. Durch diese Instanziierung der Variablen wird allerdings das Verhalten des Systems nicht verändert.

Wie können nun die generischen Maschinen, wie z. B. R=U, in der Sprache der AMs parametrisiert werden?
Es gibt die Möglichkeit, innerhalb der Maschinen die Operationen mit den jeweiligen Parametern neu zu spezifizieren. Dies wäre allerdings wenig übersichtlich, um Bibliothekssysteme zu nutzen. Eine Verallgemeinerung ist sinnvoll, etwa Parameter für AMs zu erlauben, um die Instanziierung auch sichtbar und für einen 'Compiler' leichter erkennbar und übersetzbar zu machen.

Abrial läßt die Möglichkeit, Parameter an eine AM zu übergeben, in sehr eingeschränkter Weise[8] zu. Die einzig erlaubten Werte sind natürliche Zahlen und Intervalle, um nicht den Typ, sondern nur die Dimension von Variablen der Maschine bei der Instanziierung zu bestimmen. Diese Art der Wertparameter ist für das CAVIAR-Beispiel nicht nützlich. Daher muß eine neue Form der Parametrisierung in der im folgenden beschriebenen Art definiert werden.

Wenn HR-V aus der R=U-Maschine durch Instanziierung entsteht, dann sollen die in R=U verwendeten Operationen auf die Typen aus HR-V angewendet werden. Es müssen also Typparameter definiert sein. Dieses bedeutet, daß man schon bei der Spezifikation der Maschine R=U die Mengen als Parameter angeben muß, die von anderen Maschinen, die R=U instanziieren, explizit als Parameter angegeben werden sollen:

```
machine R=U (T, R, U)
```

Jedes Vorkommen von T, R oder U in der Beschreibung der Maschine R=U wird bei der Ausführung mit dem Aufruf in HR-V

```
uses R=U (Date, HR, V)
```

mit jeweils Date, HR oder V ersetzt. Die Mengen Date, HR und V werden in einem speziellen *context* für HR-V definiert. Die Variablen erhalten automatisch einen Index, der aus dem Namen der Maschine, in welcher das 'uses'-Konstrukt steht, gebildet wird. Die Operationsnamen erhalten in Z keinen Index, da sie ein eigenständiges Schema bilden und daher explizit als Schema instanziiert werden. In AMs werden Operationen als Klausel eingebunden, sodaß man hier bei Instanziierung einer AM die Möglichkeit hätte, auch die Operationsnamen automatisch mit einem Index zu versehen. Um möglichst nahe an der Struktur und den Namen der Z-Spezifikation zu bleiben, wird diese Möglichkeit aber nicht gewählt. Die instanziierte Maschine erhält neue Operationsnamen. Man sollte möglichst keine Operationen aus der zu instanziierenden Maschine mit *promotes* übernehmen, um durch eine Neudefinition der Operation eine Verwechselung wegen der Namensgleichheit in beiden Maschinen auszuschließen. Ansonsten muß die von Abrial beschriebene Möglichkeit des Setzens eines Präfix genutzt werden [Abr90].

---

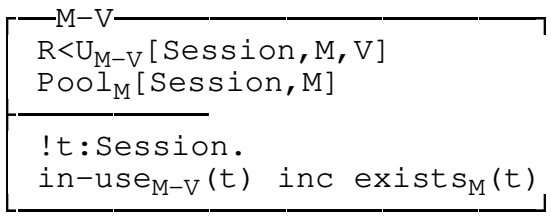
8 In [Abr91] wird eine Form der Parametrisierung von Maschinen eingeführt. In diesem Vorschlag wird von Abrial eine extends-Klausel definiert, welche dieselben Eigenschaften wie die uses-Klausel besitzt und mit der darüberhinaus als Parameter einer zu kombinierenden Maschine die Größe eines abstrakten Objekts angegeben werden kann.

5.4.3 Kombination

Die Kombination von mehreren Systemen zu einem neuen Supersystem erfolgt in Z durch Einbinden der vorhandenen Schemata in das neue Schema. In allen Schema-Kombinationen des CAVIAR-Beispiels werden zwei oder mehrere Schemata durch Konjunktion zu einem neuen Schema verknüpft. Dies ist nach der Sprachbeschreibung [Spi89] nur möglich, wenn alle Variablen, die in mehreren Schemata vorkommen, auch denselben Typ haben. Das neue Schema enthält dann alle Deklarationen und eine Konjunktion aller Prädikate sämtlicher verbundener Schemata.

Zum Beispiel ist das Tagung-Besucher-Subsystem (M-V-System) eine Kombination der beiden Systeme Pool und R<U der generischen Ebene, die beide instanziiert sind. Ein zusätzliches Prädikat verhindert, daß Besucher eine nicht existierende Tagung buchen. Somit ist dies ebenfalls als eine Spezialisierung der Kombination beider Schemata anzusehen.

Das M-V-System:



In AMs werden Maschinen allgemein mit dem *uses*-Konstrukt kombiniert. Dies bedeutet, daß die so eingebundenen Maschinen alle Variablen und Operationen zur Verfügung stellen. Im Beispiel des M-V-Systems werden die Maschinen R<U und Pool in der im Kapitel 5.4.2 vorgestellten Weise mit denselben Parametern wie in Z instanziiert.

```
machine
    M-V
sees
    M-V-context
uses
    R<U (Session, M, V)
    Pool (Session, M)
invariant
    !t(t:Session => dom(ru_M-V(t)) inc exists_M-V(t))
operations
    Add-Visitor-to-Meeting_0(session,m,v) =
        pre
            session : Session &
            m : M &
            v : V &
            !t_i.(t_i:session => (m:exists_M-V(t_i))) &
        then
            R<U-Book(session,m,v)
        end;
    Remove-Visitor-from-Meeting_0(session,m,v) =
         R<U-Cancel(session,m,v)
    Create-Meeting_0(session,m,v) = Create(session,m,v)
    Cancel-Meeting_0(session,m,v) = Cancel(session,m,v)
end
```

6. Erweiterungen des Systems

Die nähere Beschreibung der in den einzelnen Subsystemen eingeführten Ressource- und Benutzer-Objekte fehlt in der Z-Spezifikation von CAVIAR und soll in AMs an zwei Beispielen dargestellt werden, um die Möglichkeit einer genauen Datendefinition zu zeigen.

Attribute werden für sämtliche Subsysteme des CAVIAR-Systems benötigt. Exemplarisch werden für alle abstrakten Mengen, die in CAVIAR existieren, einige mögliche Eigenschaften, welche für die Systembenutzer, d. h. die Veranstalter der Tagungen, von Interesse sein können, aufgezeigt. Diese Liste ist sicher nicht vollständig und kann von einem Nicht-Experten, der die CAVIAR-Verwaltung nicht aus eigener Anschauung kennt, nur annähernd geführt werden, da hier die spezielle Information vor Ort berücksichtigt werden muß. Die Aufzählung der Eigenschaften soll hier nur als Hinweis auf die hinter den bislang aufgeführten Strukturen liegenden Verwaltungsmöglichkeiten verstanden werden.

Attribute für die verschiedenen Objekte des CAVIAR-Systems könnten im einzelnen sein:

Hotelreservierung:
Bettenanzahl, Zimmernummer, Dusche oder Bad vorhanden, WC vorhanden, Verpflegung (Frühstück, Halb- oder Vollpension möglich)

Transportreservierung:
Kennummer des Transportmittels, Art des Transportmittels(PKW, Bus), Anzahl verfügbarer Plätze

Speisezimmer:
Raumnr., Anzahl Plätze

Konferenzraum:
Raumnr., Anzahl Plätze, Ausstattung (Overhead-, Dia-Projektor, Tafel)

Tagung:
Titel, Vortragsnr., Referent, Zeitraum

Tagungszeit:
Beginn, Ende

Besucher:
Name, Vorname, Adresse, Telefon, Hotel bezahlt, Tagung bezahlt

6.1 Beschreibung

Bisher haben wir sämtliche am CAVIAR-System beteiligten Objekte als abstrakte Mengen beschrieben. Als ein neues Konzept soll jetzt eine Attributcharakterisierung der Objekte zum bereits beschriebenen CAVIAR-System hinzugefügt werden. Gleichzeitig soll das System als neue Funktionalität die Verwaltung der näher beschriebenen Objekte erhalten, da das bereits spezifizierte System des Pools hierfür nicht mehr ausreicht. Dies erfordert zuerst einmal die Attributspezifikation, um eine neue Maschine mit den Funktionen und Operationen dieser Attribute zu erstellen. In diese Maschine wird ein separater Kontext mit den nötigen Definitionen eingebunden. Zudem muß ein neuer, allgemein gültiger Kontext erstellt werden, der grundlegende Definitionen, wie z. B. Typdefinitionen und Konstanten, enthält. Für jede abstrakte Menge in CAVIAR wird eine zusätzliche Maschine spezifiziert, die zur genauen Beschreibung der Objekteigenschaften Funktionsvariable und Operationen enthält. Mit dem Strukturierungskonzept der Kombination durch die *uses*-Klausel können diese neuen Maschinen mit den vorhandenen Systemmaschinen zusammengefügt werden. Dabei können zusätzliche Integritätsbedingungen eingefügt werden, die den Zusammenhang der Objekte genau spezifizieren. Gleichfalls werden bestehende Operationen der existierenden Subsysteme mit den eingefügten Eigenschaften verfeinert.

Ein Subsystem in CAVIAR ist eine Instanziierung eines oder mehrerer Bibliothekssysteme mit abstrakten Parametern. Dies bedeutet eine Umbenennung der Systemkomponenten der zu

instanziierenden Systeme und ein Ersetzen der Platzhalter durch Bezeichnungen für abstrakte Mengen [siehe 5.4.2]. Mit der Einführung von Attributen werden die Objekte der abstrakten Mengen, welche die Instanziierung bilden, mit zusätzlichen Variablen differenzierter beschrieben. Durch die Spezifikation einer Verwaltung für diese "strukturierten" Objekte erhalten wir weitere Subsysteme, bei deren Kombination zusätzliche Integritätsbedingungen entstehen und die Operationen der zu instanziierenden Systeme spezialisiert werden müssen.

6.2 Gesamtstruktur

Durch die Spezifikation der Attribute ergibt sich eine neue Struktur bzw. Spezifikationsebene für jedes Subsystem von CAVIAR, in unserem Beispiel für das HR-V-Subsystem. Die Maschine HR-V-attributes beinhaltet die Gesamtspezifikation des HR-V-Subsystems. Sie benutzt sowohl zwei neu zu erstellende Maschinen HR-attributes und V-attributes, welche die Attributspezifikationen der Objekte HR und V enthalten, als auch die Maschine HR-V der zuvor spezifizierten Ebene [siehe Anhang A].

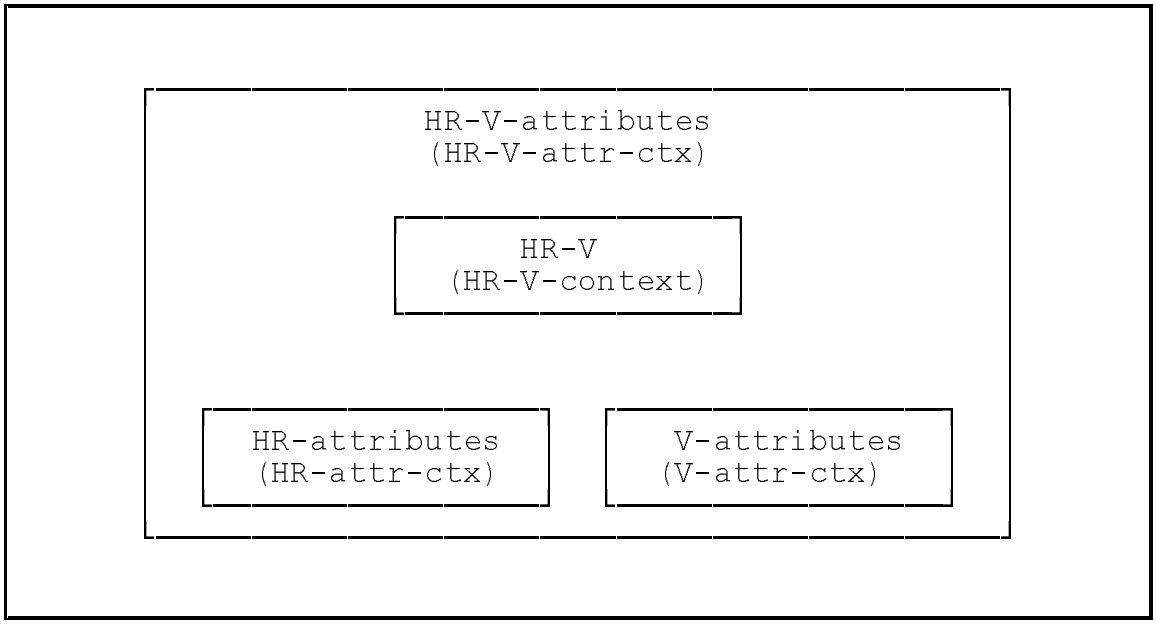


Abb. 4: Die Hierarchie des HR-V-Subsystems inklusive der Attributverwaltung

Für die anderen Teilsysteme des CAVIAR-Systems können entsprechende Maschinen auf dieser Ebene spezifiziert werden. Um das gesamte CAVIAR-System zu verwalten, fehlen im einzelnen noch die Definitionen der Attribute mit dazugehörigen Verwaltungsfunktionen für die Grundmengen

- Tagung
- Konferenzraum
- Speisesaal
- Transportreservierung
- Sitzungsinformation
- Essensinformation,

um daraus die übrigen Subsysteme CR-M [siehe 6.3.2], DR-M, TR-V, M-V und V-P mit Attributverwaltung zu erstellen.

Das System CAVIAR setzt sich auf der Ebene der Attributverwaltung aus den im Anhang A vorgestellten Maschinen und den in diesem Kapitel in Grundzügen beschriebenen Kontexten und Maschinen als Ergänzung zusammen.

6.3 Formulierung in Beispielen

Bisher wurde jedes Subsystem von CAVIAR durch eine Maschine spezifiziert. Nun müssen zu dieser Subsystem-Maschine jeweils zwei zusätzlich spezifizierte Maschinen kombiniert werden, um die Attribute für die jeweils zwei an diesem Subsystem beteiligten Objektmengen näher zu spezifizieren. Diese zwei Objektmengen sind gerade die Stellvertreter für die Ressource und den Benutzer des spezifizierten Ressource-Benutzer-Systems. Hinzu kommt für jede dieser neuen Maschinen ein Kontext, der die Mengen der Attributwerte festlegt.

Am Beispiel der beiden Subsysteme HR-V und CR-M soll im folgenden aufgezeigt werden, welche möglichen Attribute hierfür vorgesehen werden und wie diese Eigenschaften in der beschriebenen Weise in Abstrakten Maschinen spezifiziert werden können.

6.3.1 Das HR-V-Subsystem mit Attributverwaltung

Für das HR-V-Subsystem, welches Hotelzimmer, Besucher und Buchungszeiten verwaltet, wird in diesem Abschnitt eine Maschine für die Attribute der Hotelzimmerreservierung und ebenso eine für die Besucher-Attribute erstellt. Jede dieser beiden Maschinen erhält einen eigenen Kontext. Ziel ist die Spezifikation der Maschine 'HR-V-attributes', welche das HR-V-Subsystem inklusive seiner Attribute spezifiziert. Zu diesem Zweck werden die Maschinen über die jeweiligen Attribute, 'HR-attributes' und 'V-attributes', mit dem bereits spezifizierten Teilsystem 'HR-V' zusammengebunden.

Zuerst wird jedoch ein allgemeiner Kontext 'CAVIAR-sets' für das gesamte CAVIAR-System erstellt, um eine Wiederholung von Grundelementen in jedem Teilsystem zu vermeiden. Er enthält die Definitionen von Basismengen, welche in verschiedenen Teilsystemen zur Mengendefinition verwendet werden. Zum Beispiel wird eine Zeichenkette zur Beschreibung von Eigennamen, Ortsnamen oder Tagungsthemen benötigt. Da solche Typen, wie z. B. 'STRING', in AMs nicht standardmäßig definiert sind, müssen sie für jede Anwendung eigens eingeführt werden. Die Aufzählung der möglichen Objekte zur Hotelzimmerausstattung wird ebenfalls in diesem allgemeinen Kontext festgelegt, da diese Menge in mehreren Maschinen als Grundlage dient. Zum einen wird hiermit die Eigenschaft eines Hotelzimmers in HR-attr-ctx beschrieben. Zum anderen dient sie zum Festhalten des Wunsches eines Besuchers, einen bestimmten Komfort im Zimmer vorzufinden. In der Maschine HR-V-attributes kann dann z. B. beim Buchen eines Hotelzimmers genau das Zimmer ausgesucht werden, welches den Ansprüchen des Besuchers genügt. Der allgemeine Kontext CAVIAR-sets stellt somit für CAVIAR spezifische Mengen von Werten zur Verfügung, von denen jeweils einige in den Subsystemen der Objektdefinition dienen.

```
context
    CAVIAR-sets
sets
    TAG = {1,...,31};
```

```
    MONAT = {1,...,12};
    JAHR = {1970,...,2099};
    STUNDE = {0,...,23};
    MINUTE = {1,...,59};

    DATUM = TAG x MONAT x JAHR;
    UHRZEIT = STUNDE x MINUTE;

    ZEITPUNKT = DATUM x UHRZEIT;
    ANFANGSZEIT = ZEITPUNKT;
    ENDZEIT = ZEITPUNKT;
    ZEITRAUM = ANFANGSZEIT x ENDZEIT;

    CHAR = {a,..,z,A,..Z,0,..,9};
    STRING = POW(CHAR);

    ZIMMERAUSSTATTUNG = {Bad, Dusche, WC, Bar, TV};
                ;(* für Subsystem HR-V *)
    CR-AUSSTATTUNG = {Overhead, Tafel, Kreide, Schwamm, Dia};
                ;(* für Subsystem CR-M *)
end
```

Als erste Maschine für das HR-V-Subsystem wird eine Maschine zur Spezifizierung des Objekts HR erstellt. Das Objekt HR wird im Kontext HR-attr-ctx als abstrakte Menge definiert, d. h. es ist eine endliche, nicht-leere Menge. Sie repräsentiert die Menge aller denkbaren Hotelzimmer. Die Mengen von Eigenschaften eines Hotelzimmers werden ebenfalls im Kontext definiert. Um die Spezifikation hier einfach zu halten, wird vorausgesetzt, daß nur ein Hotel zur Zimmerbelegung zur Verfügung steht, sodaß weder Hotelname, noch Hoteladresse angegeben werden müssen. Zur Spezifikation der Eigenschaften benötigt man den allgemeinen Kontext CAVIAR-sets, der mit der Klausel *sees* eingebunden wird. Die Maschine HR-attributes definiert dann mit 'hr' die Menge sämtlicher Hotelzimmer, die für CAVIAR verwaltet werden. Somit muß es eine Einfüge- und eine Löschoperation für ein Hotelzimmer geben. Beim Eintragen eines Zimmers in die Menge aller zu verwaltenden Hotelzimmer werden sämtliche Attribute für dieses Zimmer gesetzt. Mit Hilfe der 'modify_'-Operationen können einzelne Eigenschaften eines Zimmers nachträglich geändert werden. Weiterhin gibt es für jedes Attribut eine Operation, die den Wert dieses Attributs für ein bestimmtes Zimmer liefert.

```
context
    HR-attr-ctx
sees
    CAVIAR-sets
    HR-V-context
sets
    STOCKWERK = N;
    BETTENZAHL = N;
    AUSSTATTUNG = ZIMMERAUSSTATTUNG;
end

machine
    HR-attributes
sees
    HR-attr-ctx
variables
    hr, stockwerk, bettenzahl, ausstattung
invariant
    hr inc HR;
    stockwerk : hr --> STOCKWERK;
    bettenzahl : hr --> BETTENZAHL;
    ausstattung : hr --> AUSSTATTUNG
initialization
    hr, stockwerk, bettenzahl, ausstattung := {}, {}, {}, {}

operations
    z <-- create_zimmer(s,b,a) =
        pre
            z : HR &
            s : STOCKWERK &
            b : BETTENZAHL &
            a : AUSSTATTUNG
        then
            any h where
                h : HR - hr
            then
                hr := hr \/ {h} ||
                stockwerk(h) := s ||
                bettenzahl(h) := b ||
                ausstattung(h) := a ||
                z := h
            end
        end;

    destroy_zimmer(z) =
        pre
            z : hr
        then
            hr := hr - {z} ||
            stockwerk := (hr - {z}) <| stockwerk ||
            bettenzahl := (hr - {z}) <| bettenzahl ||
            ausstattung := (hr - {z}) <| ausstattung
        end;

    modify_bettenzahl(z,b) =
        pre
            z : hr &
            b : BETTENZAHL
        then
            bettenzahl(z) := b
        end;
```

```
    modify_ausstattg(z,a) =
        pre
            z : hr &
            a : AUSSTATTUNG
        then
            ausstattung(z) := a
        end;

    v <-- stockwerk_(z) =
        pre
            z : hr
        then
            v := stockwerk(z)
        end;

    v <-- bettenzahl_(z) =
        pre
            z : hr
        then
            v := bettenzahl(z)
        end;

    v <-- ausstattg_(z) =
        pre
            z : hr
        then
            v := ausstattung(z)
        end;

end
```

Als nächste Maschine für das HR-V-Subsystem wird die Maschine zur Spezifizierung des Objekts V mit Attributen erstellt. Auch für die Besucher wird ein eigener Kontext erstellt. Mit 'visitors' wird wiederum die Menge aller mit der Einfügeoperation eingetragenen Besucher als Teilmenge aller imaginären Besucher spezifiziert. Neben der Einfüge- und eine Löschoperation werden wieder die 'modify_'- und Wertanfrage-Operationen spezifiziert.

```
context
    V-attr-ctx
sees
    CAVIAR-sets
    HR-V-context
sets
    NAME = STRING;
    STRASSE = STRING;
    ORT = STRING;
    ADRESSE = STRASSE x ORT;
    AUSSTATTG_WUNSCH = ZIMMERAUSSTATTUNG;
end
```

```
machine
    V-attributes
sees
    V-attr-ctx
variables
    name, adresse, ausstattg_wunsch, visitors

invariant
    visitors inc V;
    name : visitors --> NAME;
    adresse : visitors --> ADRESSE;
    ausstattg_wunsch : visitors --> AUSSTATTG_WUNSCH

initialization
    visitors := {}
    name := {}
    adresse := {}
    ausstattg_wunsch := {}

operations
    b <-- create_besucher(n,a,w) =
        pre
            n : NAME &
            a : ADRESSE &
            w : AUSSTATTG_WUNSCH
        then
            any v where
                v : V - visitors
            then
                visitors := visitors \/ {v} ||
                name(v) := n ||
                adresse(v) := a ||
                ausstattg_wunsch(v) := w ||
                b := v
            end
        end;

    destroy_besucher(b) =
        pre
            b : visitors
        then
            visitors := visitors - {b} ||
            name := (visitors - {b}) <| name ||
            adresse := (visitors - {b}) <| adresse ||
            ausstattg_wunsch := (visitors -{b}) <| ausstattg_wunsch
        end;

    modify_name(b,n) =
        pre
            b : visitors &
            n : NAME
        then
            name := name <+ {b |-> n}
        end;
```

```
    modify_adresse(b,a) =
        pre
            b : visitors &
            a : ADRESSE
        then
            adresse := adresse <+ {b |-> a}
        end;

    modify_ausstattg_wunsch(b,w) =
        pre
            b : visitors &
            w : AUSSTATTG_WUNSCH
        then
            ausstattg_wunsch := ausstattg_wunsch <+ {b |-> w}
        end;

    v <-- name_(b) =
        pre
            b : visitors &
        then
            v := name(b)
        end;

    v <-- adresse_(a) =
        pre
            b : visitors &
        then
            v := adresse(b)
        end;

    v <-- ausstattg_wunsch_(b) =
        pre
            b : visitors &
        then
            v := ausstattg_wunsch(b)
        end;
```

end

Nachdem Maschinen für Besucher und Hotelzimmer spezifiziert sind, wird nun im nächsten Schritt HR-V, welches eine Instanziierung von R=U war [siehe Anhang A], mit V-attributes und HR-attributes zusammengefügt. Dafür müssen die refernetiellen Integritätsbedingungen, welche den Zusammenhang zwischen Besuchern und Hotelzimmern festlegen sollen, unter *invariant* eingetragen werden. Zum einen müssen Besucher, welche ein Hotelzimmer gebucht haben, auch vorhandene, eingetragene Besucher sein. Zum anderen wird als Integritätsbedingung definiert, daß sämtliche Hotelzimmer existieren müssen, wenn sie von einem Besucher gebucht worden sind.

Alle Operationen, die Werte zurückliefern oder auch nur Eigenschaften eines Objekts (Besucher oder Hotelzimmer) ändern bzw. ein Objekt eintragen oder wieder löschen, sollen auch zu dieser Maschine gehören, um sie auf dieser Ebene dem Benutzer zur Verfügung zu stellen. Es kann dann eine Zimmerbuchung auf der gleichen Ebene durchgeführt werden wie das Eintragen eines neuen Besuchers. Daher werden sämtliche Operationen der AMs 'HR-attributes' und 'V-attributes' in der *promotes*-Klausel aufgezählt. Würde man die gesamte Verwaltung für den Anwender modular aufbauen, dann könnte man auf dieses Weitergeben von Operationen verzichten, indem man an einer Stelle der Verwaltung nur das Buchen von Zimmern erlaubt und an anderer Stelle in einem anderen Modul das Einfügen und Ändern von Besuchern und Hotelzimmern mit ihren Attributen durchführt.

Als zusätzliche Eigenschaft der Beziehung zwischen Besuchern und Hotelzimmern wird die Variable 'verpflegung' eingeführt. Sie beschreibt, ob z. B. der Gast im Zimmer Nr. 100 Voll- oder Halbpension bzw. nur mit Frühstück gebucht hat. Dieses Attribut bleibt über den gesamten

Buchungszeitraum gleich, da es ungewöhnlich ist, für einen Tag Vollpension und für den nächsten Tag nur mit Frühstück zu buchen. Dieses Attribut ist somit eine Eigenschaft der Relation r|->u[9].

Als Operationen werden beispielhaft das Buchen eines Zimmers und das Stornieren einer Zimmerbuchung angegeben. In diesen Operationen muß auch die Variable 'verpflegung' mit dem neuesten Wert belegt werden. Bei der Buchung muß der Wunsch des Besuchers nach einem bestimmten Zimmerkomfort erfüllt werden, um ein Zimmer erfolgreich belegen zu können. Da bei dieser Operation keine weiteren Bedingungen berücksichtigt werden, wird als ein spezieller Fall einer Zimmerbuchung das Buchen eines Einzelzimmers behandelt, um beispielhaft den Wunsch des Gastes nach einem Einzelzimmer zu berücksichtigen. Für weitere, mögliche Anforderungen des Gastes an sein Hotelzimmer werden ebenfalls Operationen zur Ausführung nötig. In diesem Kapitel sind sie jedoch zur Vereinfachung ausgelassen.

```
context
    HR-V-attr-ctx
sees
    HR-attr-ctx
    V-attr-ctx
sets
    VERPFLEGUNG = {Frühstück, Halbpension, Vollpension};
    DATE = DATUM;
end

machine
    HR-V-attributes
sees
    HR-V-attr-ctx
uses
    HR-attributes,
    V-attributes,
    HR-V
promotes
    create_zimmer, destroy_zimmer,
    modify_bettenzahl, modify_ausstattg,
    stockwerk_, bettenzahl_, ausstattg_,

    create_besucher, destroy_besucher,
    modify_name, modify_adresse, modify_ausstattg_wunsch,
    name_, adresse_, ausstattg_wunsch_
```

---

9 Wenn derselbe Besucher zu verschiedenen Zeiten bucht und ihm zufällig dasselbe Zimmer reserviert wird, dann entsteht das unerwünschte Verhalten, daß er in beiden Besuchen dieselbe Verpflegung bekommt. Er hat dann nicht die Möglichkeit, z. B. zur Sommertagung Halbpension und zur Wintertagung Vollpension zu buchen. Da es aber sehr unwahrscheinlich ist, daß ein Besucher mehrmals dasselbe Zimmer bekommt (dies könnte auch mit einem eingefügten Algorithmus vermieden werden), wird dieser Umstand in der vorliegenden Spezifikation vernachlässigt.

```
variables
    verpflegung
invariant
    verpflegung : ran(ru_HR-V) --> VERPFLEGUNG;
    ran(ran(ru_HR-V)) inc visitors;
    dom(ran(ru_HR-V)) inc hr;
    !z.(z:hr => (!b.(b:visitors =>
        ausstattung(z) = ausstattg_wunsch(b))))
        ;(* nur für Übereinstimmung der Zimmerausstattung mit *)
        ;(* dem Wunsch des Besuchers werden Buchungs- und *)
        ;(* Stornierungsoperationen durchgeführt *)

initialization
    verpflegung := {}
operations
    r <-- buche_zimmer(b,d,v) =
        pre
            b : visitors &
            d : DATE &
            v : VERPFLEGUNG
        then
            any z where
                z : hr &    ;(* z muß existieren *)
                !d_i.((d_i:d) => NOT(z : dom(ru_HR-V(d_i)))
                            ;(* z muß auch frei sein[10] *)
            then
                r := z ||
                Book-Hotel-Room_0(d,z,b) ||
                verpflegung := verpflegung <+
                    %x.(x:ru_HR-V(d) & b:ran(x) | x |-> v)
            end;
        end;

    r <-- buche_einzelzi(b,d,v) =
        pre
            b : visitors &
            d : DATE &
            v : VERPFLEGUNG
        then
            any z where
                z : hr &
                bettenzahl(z) = 1 &
                !d_i.((d_i:d) => NOT(z : dom(ru_HR-V(d_i)))
                            ;(* z muß auch frei sein *)
            then
                r := z ||
                Book-Hotel-Room_0(d,z,b) ||
                verpflegung := verpflegung <+
                    %x.(x:ru_HR-V(d) & b:ran(x) | x |-> v)
            end;
        end
```

[10] Die Integritätsbedingung, daß ein Zimmer frei sein muß, bevor es gebucht werden kann, steht ebenfalls in der Operation Book-Hotel-Room$_0$. Da aber bei Abrial nicht genau beschrieben wird, welches Verahlten eine AM hat, wenn eine der parallelen Anweisungen im 'then'-Teil der Operation nicht erfolgreich ausgeführt werden kann, muß diese Vorbedingung hier wiederholt werden. Werden die anderen parallelen Operationen im Fehlerfall nicht zurückgesetzt, dann führt es in diesem Fall dazu, daß ein Zimmer r als belegt zurückgegeben wird und auch das Attribut 'verpflegung' dafür belegt wird, obwohl die Buchungsoperation Book-Hotel-Room$_0$ nicht ausgeführt wurde.

```
    storniere_zimmer(z,b,d) =
        pre
            z : hr &
            b : visitors &
            d : DATE
        then
            Cancel-Hotel-Room0(d,z,b) ||
            verpflegung := (z |-> b) <<| verpflegung
        end;
end
```

6.3.2 Das CR-M-Subsystem mit Attributverwaltung

Ein weiteres Beispiel für die Attributverwaltung soll zum Vergleich das CR-M-Subsystem sein. In diesem Teilsystem werden die Tagungen den Konferenzräumen zu den gegebenen Veranstaltungszeiten zugeordnet. Die vollständige Spezifikation dieses Subsystems ist im Anhang B zu finden.

Mit 'CR-attributes' wird eine Maschine zur Spezifizierung der Eigenschaften von Konferenzräumen erstellt. Wie schon im HR-V-Subsystem beschrieben, werden auch hier Einfüge- und Lösch-operationen benötigt. Auf die Operationen zur Änderung von Gebäude und Stockwerk wurde verzichtet, weil es als eine feste Zuordnung angesehen wird. Wenn ein Konferenzraum z. B. in ein anderes Gebäude wechselt, dann bedeutet dieser Vorgang für das CAVIAR-System das Einfügen eines neuen Konferenzraums, d.h. der alte Raum muß gelöscht und ein neuer Raum eingefügt werden.

Als weitere, neue Maschine wird 'M-attributes' zur Spezifizierung der Eigenschaften von Tagungen erstellt. Es werden wieder Einfüge- und Löschoperationen, sowie Operationen zur Änderung und zur Abfrage von Eigenschaften einer Tagung spezifiziert.

Wie schon bei dem HR-V-Subsystem erfolgt die Zusammensetzung des CR-M-Subsystems durch Kombination von 'CR-M' mit 'M-attributes' und 'CR-attributes'. Dabei werden zusätzliche Invarianten benötigt, die sicherstellen sollen, daß alle belegten Konferenzräume eingetragene Räume sind und daß alle Tagungen, die einen Raum belegen, auch existieren. In diesem CR-M-Subsystem haben wir die Möglichkeit, bei der Zusammenführung von Tagungen und Konferenzräumen in eine einzige Maschine mit weiteren Integritätsbedingungen diese Beziehung einzugrenzen. So kann beschrieben werden, daß Tagungen einer bestimmten Größenordnung nur in Räumen stattfinden können, die genügend Platz bieten. Die zusätzliche Invariante legt daher die Zuordnung für sämtliche Beziehungen von Räumen zu Tagungen fest. Kleine Tagungen finden in Räumen mit maximal 20 Sitzplätzen statt. Mittlere Tagungen können bis zu 80 Teilnehmer haben und große Tagungen haben einen Platzbedarf über 80 hinaus. Diese Eigenschaft muß bei jeder Operation erhalten bleiben.

Die Operation 'exist_cr' ist eine Anfrage, ob für eine festgelegte Tagung ein Konferenzraum existiert. Bei Bedarf kann diese Operation zur Anfrage erweitert werden, ob ein Konferenzraum genau in der gewünschten Tagungszeit frei ist. In jedem Fall soll über die definierten Eigenschaften abgefragt werden, ob die Ausstattung des Raumes den Ansprüchen dieser Tagung genügt.

7. Ausblick

Durch die Übertragung der Spezifikation des CAVIAR-Systems von Z in Abstrakte Maschinen entsteht die Möglichkeit, beide Sprachen hinsichtlich einer Spezifikation miteinander zu vergleichen. Es wird deutlich, daß die als Grundlage gewählte Spezifikation des CAVIAR-Systems von der Sprache Z geprägt ist, sodaß an denjenigen Stellen, an denen es in AMs keine mit Z vergleichbaren Konstrukte gibt, ersatzweise andere Sprachkonzepte und teilweise auch neue Konstrukte benötigt werden, um eine Entsprechung in der Spezifikation zu erhalten. Insofern ist eine Spezifikation abhängig von der Sprache, in der sie entworfen wird, obwohl dieses nicht bedeutet, daß keine andere Sprache zur Spezifikation des CAVIAR-Systems gewählt werden könnte.

Die vorliegende Arbeit geht über die reine Traduktion der Z-Spezifikation des CAVIAR-Systems in die Spezifikationssprache der AMs und den Vergleich dieser Sprachen hinaus, indem sie die CAVIAR-Systemmodellierung um eine Verwaltung von Attributen ergänzt. Diese Erweiterung soll zeigen, wie detailgenau eine Spezifikation, die i. A. zu den ersten Schritten des gesamten Systementwurfes gehört, durchgeführt werden kann. Es ist dabei deutlich geworden, daß man auch in der Spezifikation die Attribute der Objekte einbinden kann. Je komplexer allerdings diese Objekte und ihre Beziehungen werden, desto umfangreicher und leider auch unübersichtlicher wird deren Attributverwaltung, sodaß man sich in der Phase der Spezifikation auf die wichtigsten Attribute beschränken sollte.

Das Konzept der generischen Spezifikation und ihrer Instanziierung erweist sich als sinnvoller Ansatz, da insbesondere in dem Beispiel des CAVIAR-Systems die Spezifikation des Gesamtsystems eine klare Struktur erhält. Jedes Teilsystem entsteht in genau festgelegter Weise durch Kombination von instanziierten, generischen Spezifikationen. Um eine übersichtliche Struktur auch in anderen, größeren Systemspezifikationen zu erreichen, ist gerade dieser Ansatz wertvoll. Es lassen sich in vielen Anwendungen Gemeinsamkeiten in Teilsystemen finden, die als generische Spezifikationen zu verschiedenen Teilsystemen generiert werden können. Gerade Ressource-Benutzer-Systeme, welche ebenfalls als Reservierungssysteme angesehen werden können, sind in vielen zu spezifizierenden Systemen enthalten. Dabei spart es erheblichen Aufwand, ein generisches Bibliothekssystem zu erstellen, welches sich zu verschiedenen Teilsystemen instanziieren läßt, ohne jedes Teilsystem für sich allein spezifizieren zu müssen. Die Wahrscheinlichkeit, daß man eine solche generische Spezifikation für weitere Gesamtsysteme ohne zusätzlichen Kostenaufwand wiederverwenden kann, ist umso größer, je allgemeiner diese Spezifikation gehalten ist.

Die durch das CAVIAR-Beispiel entstandene Forderung einer Instanziierung durch Parametrisierung in Abstrakten Maschinen [siehe 5.4.2] ist bisher noch nicht in der Literatur vertreten. Es bedarf weiterer Formalisierung, um dieses Konzept in AMs zu integrieren. Da die Parametrisierung in vielen Programmiersprachen erfolgreich eingesetzt wird, wäre es durchaus wünschenswert, dieses Konzept auch in Abstrakten Maschinen zur Verfügung zu haben, zumal es die Grundlage der Instanziierung bildet und somit für generische Spezifikationen die hierfür genannten Vorteile bieten würde.

Neben Spracherweiterungen und einer Sprachunterstützung für generische Spezifikationen wäre eine interaktive Modellierungsumgebung sehr wünschenswert. Durch moderne Fenstertechnik könnte die Lesbarkeit der Spezifikation erhöht werden. Eine gleichzeitige Anzeige mehrerer, bereits für das Gesamtsystem spezifizierter Maschinen oder Kontexte bietet dem Designer sofort die übersichtliche Information, die er zur weiteren Spezifikation benötigt.

Es sollten ebenfalls Anfragen an die bereits vorliegenden Spezifikationsteile gestellt werden können, z. B. um die Formulierung von systemübergreifenden Integritätsbedingungen zu unterstützen.

Für den gesamten Vorgang der Spezifikation wäre ein Werkzeug zur Erstellung von mathematischen Ausdrücken hilfreich, welches z. B. bei der Maschinenspezifikation zu jeder Klausel eine Liste der erlaubten Operatoren angibt und einen davon auswählen läßt.

Ferner bietet sich bei der Verwendung formaler Spezifikationen an, in die Modellierungsumgebung Konsistenzüberprüfungen, z. B. hinsichtlich des Erhalts der Invarianten, einzubinden.

Im Zusammenhang mit der Erstellung einer Gesamtspezifikation aus Bibliothekssystemen sollte eine Werkzeugunterstützung zur Auswahl und Instanziierung von Bibliothekssystemen angeboten werden. Dieses Werkzeug beinhaltet die Verwaltung der Bibliothekssysteme, um ein System hinzufügen, ändern oder löschen zu können, wobei jedes Bibliothekssystem mit Erläuterungen über mögliche Parameter versehen wird. Desweiteren ist die Auswahl und Instanziierung generischer Systeme vom Werkzeug zu unterstützen. Eines der generischen Bibliothekssysteme kann unter Angabe der gewünschten, aktuellen Parameter, welche evtl. auf Korrektheit abgeprüft werden können, ausgewählt werden, um aus der generischen Spezifikation eine Instanz zu erzeugen. Die Instanziierung sollte durch automatische Ersetzung der Variablennamen erfolgen können, z. B. durch Indizierung der Namen wie in den Z-Spezifikationen, wobei auch die Möglichkeit des expliziten Benennens vorgesehen sein sollte, um Namensgleichheiten zu vermeiden oder besondere Wünsche bzw. Vereinbarungen berücksichtigen zu können.

Zu jeder generischen Spezifikation wäre zudem eine automatisch erzeugte Übersicht über sämtliche Spezialisierungen mit Erläuterungen sehr nützlich. So kann vermieden werden, daß eine Spezialisierung mehrmals vorliegt und das Gesamtsystem unnötig aufbläht.

Ebenfalls könnte ein Pool aus Integritätsbedingungen zur Unterstützung der Auswahl angeboten werden. Die Übersicht sollte allgemeine und die im speziellen Zusammenhang sinnvollen Integritäts-bedingungen umfassen. Selbstverständlich sollte dieser Pool jederzeit mit einem einfachen Werkzeug verändert oder ergänzt werden dürfen.

Anhang A: Das System CAVIAR als Abstrakte Maschine

I. Ein allgemeines Ressource-Benutzer-System R-U

```
context
    R-U-context
sets
    T;  (* T ist Menge von Zeitpunkten *)
    R;  (* R ist Menge von Ressourcen *)
    U;  (* U ist Menge von Benutzern *)
end

machine
    R-U
variables
    ru,
    in-use, users, ur
invariant
    ru  inc T --> (R <-> U)
                ;(* totale Funktion von einer Zeitmenge zu der
                 Relation zwischen Ressourcen u. Benutzern. *)

;(* folgende Funktionen sind als redundante Funktionen in CAVIAR
zusätzlich definiert, werden in Abstrakten Maschinen aber wegen
des hohen Aufwands zur Aktualisierung nicht weiter verwendet.
Siehe auch Kap. 5.1.1 *)
    in-use  inc T --> POW(R)
                ;(* alle zur Zeit belegten Ressourcen *)
    users   inc T --> POW(U)
                ;(* alle zur Zeit vorhandenen Benutzer *)
    ur      inc T --> (U <-> R)
                ;(* "Umkehrfunktion" zu ru
                 = alle zur Zeit vorhandenen Beziehungen
                 zwischen Benutzern und Ressourcen *)

;(* folgende Konsistenzbedingungen gelten für diese Funktionen *)
    !t.(t : T => (in-use(t) = dom(ru(t))))
    !t.(t : T => (users(t) = rng(ru(t))))
    !t.(t : T => (ur(t) = (ru(t))~1))
initialisation
    ru = %t.(t : T | {} );
operations
    R-U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.((t_i:t) => (not ({ r |-> u }:ru(t_i)) ))
```

```
        then
            ru := %t_i.(t_i:t | ru(t_i) \/ {r |-> u}) \/ t <<| ru
            ;(* für alle t_i die Relation (r,u) zufügen und mit der
             Menge der (r,u) für alle t_i nicht in t vereinigen *)
            ;(* ACHTUNG : Wenn für einige t_i der Funktionswert
             (r,u) schon vorhanden ist, wird er überschrieben. *)
        end;
    R-U-Cancel(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.((t_i:t) => ({r|->u}:ru(t_i)))
        then
            ru := (%t_i.(t_i:t | (ru(t_i) - {r|->u}))) \/ t<<|ru
        end;
    R-U-Del-Res(t,r) =  ;(* r wird nicht aus R,
                           sondern nur aus ru gelöscht ! *)
        pre
            t : POW(T) &
            r : R &
            !t_i.(t_i:t => r:dom(ru(t_i)))
        then
            ru := (%t_i.(t_i:t | r<<|ru)) \/ t<<|ru
        end;
    R-U-Del-User(t,u) =
        pre
            t : POW(T) &
            u : U &
            !t_i.(t_i:t => u:rng(ru(t_i)))
        then
            ru := (%t_i.(t_i:t | (ru(t_i) |>> u))) \/ t <<|[1] ru
        end
end
```

II. Spezialisierung des allgemeinen R-U-Systems

A. Die Ressourcen sind nicht teilbar

```
machine
    R>U ( T, R, U )
uses
    R-U
invariant
    rng(ru) inc R +-> U
    ;(* partielle Funktion als zusätzliche Einschränkung zu R-U *)
initialisation
    rng(ru) = {}
```

---

[1] Der Operator <<| bezeichnet die Wertebereichsubtraktion, auch Anti-Co-Restriktion genannt.
ru(t_i)|>>u ist gerade die Menge aller Beziehungen (r,x) aus ru(t_i), deren Relationswert nicht in u liegt.
Definition: r : s <-> t & v inc t => r |>> v = r;id((ran(r) - v)),
wobei der Operator ';' die reverse Komposition und 'id' die Identitätsrelation bezeichnet.

```
operations
    R>U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.(t_i:t => not(r:dom(ru(t_i))))
        then
            R-U-Book(t,r,u)
        end;
    R>U-Cancel(t,r,u) =
        R-U-Cancel(t,r,u);
        ;(* erhält zusätzlich nur die Einschränkungen durch die
         Invariante von R>U gegenüber R-U-Cancel *)
    R>U-Del-User(t,u) =
        R-U-Del-User(t,u)
        ;(* erhält zusätzlich nur die Einschränkungen durch die
         Invariante von R>U gegenüber R-U-Del-User *)
    ;(* R>U-Del-Res entfällt, da bei maximal einem Benutzer pro
     Ressource mit R>U-Cancel automatisch auch die Ressource
     gelöscht wird!*)
end
```

B. Ein Benutzer kann maximal eine Ressource belegen

```
machine
    R<U ( T, R, U )
uses
    R-U
invariant
    rng((ru)^-1) inc U +-> R
    ;(* partielle Funktion als zusätzliche Einschränkung zu R-U *)
initialisation
    rng(ru) = {}
operations
    R<U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.(t_i:t => not(u:rng(ru(t_i))))
        then
            R<U-Book := R-U-Book(t,r,u)
        end;
    R<U-Cancel(t,r,u) =
        R-U-Cancel(t,r,u);
        ;(* erhält zusätzlich nur die Einschränkungen durch die
         Invariante von R<U gegenüber R-U-Cancel *)
    R<U-Del-Resource(t,u) =
        R-U-Del-Resource(t,u)
        ;(* erhält zusätzlich nur die Einschränkungen durch die
         Invariante von R<U gegenüber R-U-Del-Resource *)
    ;(* R<U-Del-User
     entfällt, da bei maximal einer Ressource pro Benutzer
     r automatisch für ein u bekannt ist und die Löschung
     eines Benutzers mit R<U-Cancel vorgenommen werden kann! *)
end
```

C. Ein Benutzer kann maximal eine nicht-teilbare Ressource belegen

```
machine
    R=U ( T, R, U )
uses
    R-U
;(* R=U als Maschine von R>U und gleichzeitig R<U mit dem uses-
Konstrukt auszudrücken, ist in AMs nicht möglich, weil im
Ausführungsteil ('then') zur Definition der Operationen nur eine
Operation (entweder aus R>U oder R<U) einer benutzten Maschine
aufgerufen werden kann. Es gibt keine Möglichkeit zur Konjunktion
zweier Operationen. Im Prädikatenteil muß dann die Bedingung der
anderen Operation zusätzlich aufgeführt werden. Aus diesem Grund
wird R=U einfach als Spezialisierung von R-U geschrieben, sodaß
die Invarianten und Präkonditionen sowohl von R>U, als auch von
R<U hier erneut aufgeführt werden müssen. *)
invariant
    rng(ru) inc R >+> U
    ;(* hier ist ru eine partielle injektive Funktion *)
initialisation
    rng(ru) = {}
operations
    R=U-Book(t,r,u) =
        pre
            t : POW(T) &
            r : R &
            u : U &
            !t_i.(t_i:t => not(r:dom(ru(t_i)))) &
                          ;(* Präkondition von R>U-Book und *)
            !t_i.(t_i:t => not(u:rng(ru(t_i))))
                          ;(* R<U-Book konjunktiv verknüpft *)
        then
            R-U-Book(t,r,u)
        end;
    R=U-Cancel(t,r,u) =
        R-U-Cancel(t,r,u);
        ;(* erhält zusätzlich nur die Einschränkungen durch die
         Invariante von R=U gegenüber R-U-Cancel *)
    ;(* R=U-Del-Res und
     R=U-Del-User entfallen, da diese Löschungen gleichbedeutend
     sind mit dem Löschen genau einer r-u-Beziehung und mit
     R=U-Cancel vorgenommen werden *)
end
```

III. Erstellen eines Teilsystems aus CAVIAR

A. Das Hotelreservierung-Subsystem HR-V

```
context
    HR-V-context
sets
    Date;    (* Date ist Menge von Zeitpunkten *)
    HR;      (* HR ist Menge von Hotelreservierungen *)
    V;       (* V ist Menge von Besuchern *)
end


machine
    HR-V
sees
    HR-V-context
uses
    R=U (Date, HR, V)
variables
;(* Die Variable ru_HR-V wird automatisch durch die Parametrisierung
zur Verfügung gestellt *)
operations
    Book-Hotel-Room_0(date,hr,v) = R=U-Book(date,hr,v)
    Cancel-Hotel-Room_0(date,hr,v) = R=U-Cancel(date,hr,v)
end
```

B. Das Transportreservierung-Subsystem TR-V

```
context
    TR-V-context
sets
    Time;    (* Time ist Menge von Zeitpunkten *)
    TR;      (* TR ist Menge von Transportreservierungen *)
    V;       (* V ist Menge von Besuchern *)
end


machine
    TR-V
sees
    TR-V-context
uses
    R=U (Time, TR, V)
variables
;(* ru_TR-V wird durch Parametrisierung zur Verfügung gestellt *)
operations
    Book-Hotel-Room_0(time,tr,v) = R=U-Book(time,tr,v);
    Cancel-Hotel-Room_0(time,tr,v) = R=U-Cancel(time,tr,v)
end
```

C. Ein Pool-System

```
machine
    Pool ( T, X )
variables
    exists
invariant
    exists inc T --> POW(X)
            ;(* Funktion von Zeitmenge zu einer Grundmenge X *)
initialisation
    exists = %t_i.(t_i : T | {} )
operations
    Create(t,x) =
        pre
            t : POW(T) &
            x : X
            ;(* Wenn x eventuell für einige Zeiten aus t schon
             existiert, wird der Funktionswert überschrieben *)
        then
            exists := exists <+ (%t_i.(t_i : t | exists(t_i) \/ {x}))
            ;(* für alle t_i das Element x zufügen *)
        end;
    Destroy(t,x) =
        pre
            t : POW(T) &
            x : X
            ;(* Auch wenn x eventuell für einige Zeiten aus t nicht
             existiert, wird x für alle t_i aus t entfernt *)
        then
            exists := exists <+ (%t_i.(t_i : t | (exists(t_i) - {x})))
        end;
end
```

D. Das Tagung-Besucher-Subsystem M-V

```
context
    M-V-context
sets
    Session;     (* Session ist Menge von Zeitpunkten *)
    M;           (* M ist Menge von Tagungen *)
    V;           (* V ist Menge von Besuchern *)
end


machine
    M-V
sees
    M-V-context
uses
    R<U (Session, M, V)
    Pool (Session, M)
variables
;(* ru_M-V und exists_M-V werden durch Parametrisierung von R<U und
Pool zur Verfügung gestellt *)
invariant
    !t.(t:Session => dom(ru_M-V(t)) inc exists_M-V(t))
    ;(* alle Tagungen in ru_M-V existieren *)
```

```
operations
    ;(* sämtliche Operationen dieser Maschine sind Umbenennungen
     von Operationen aus R<U und Pool *)
    Add-Visitor-to-Meeting0(session,m,v) =
        pre
            session : Session &
            m : M &
            v : V &
            !ti.(ti:session => (m:existsM-V(ti)))
            ;(* Präkondition, damit die Operation zum
             Konsistenznachweis die Invariante von M-V erfüllt *)
        then
            R<U-Book(session,m,v)
        end;
    Remove-Visitor-from-Meeting0(session,m,v) =
         R<U-Cancel(session,m,v)
    Create-Meeting0(session,m,v) = Create(session,m,v)
    Cancel-Meeting0(session,m,v) = Cancel(session,m,v)
end
```

E. Ein Diary-System

```
machine
    Diary ( T, X, Ix )
variables
    info,
    recorded
invariant
    info inc T --> (X +-> Ix)
        ;(* Funktion von Zeitmenge zu einer partiellen Funktion
         zwischen Grundmenge X und der zugehörigen
         Informationsmenge Ix *)
    ;(* als redundante Funktion ist weiterhin definiert : *)
    recorded inc T --> POW(X)
    ;(* für recorded gilt folgende Konsistenzbedingung :*)
    !ti.(ti : T => recorded(ti) = dom(info(ti))
initialisation
    info = %ti.(ti : T | {} )
operations
    Add(t,x,i) =
        pre
            t : T &
            x : X &
            i : Ix &
            !ti.(ti:t => not(x inc dom(info(ti))))
            ;(* für x existiert noch keine Information *)
        then
            info := info <+ (%ti.(ti : t | info(ti) <+ {x,i}))
            ;(* für alle ti die Information i für
             das Element x zufügen *)
        end;
```

```
    Erase(t,x) =
        pre
            t : T &
            x : T +-> POW(X) &
            dom(x) = t &
            !t_i.(t_i:t => x(t_i) inc dom(info(t_i)))
            ;(* für x existiert eine Information für
             die Zeitmenge t *)
        then
            info := info <+ (%t_i.(t_i : t | (x(t_i) <<| info(t_i) )))
        end;
end
```

F. Das Konferenzraum-Tagung-Subsystem CR-M

```
context
    CR-M-context
sets
    Session;     (* Session ist Menge von Zeitpunkten *)
    CR;          (* CR ist Menge von Konferenzräumen *)
    M;           (* M ist Menge von Tagungen *)
    SI;          (* SI ist Menge der Sitzungsinformation *)
end


machine
    CR-M
sees
    CR-M-context
uses
    R>U (Session, CR, M)
    Diary (Session, CR, SI)
variables
;(* ru_CR-M und info_CR-M werden durch Parametrisierung von R>U und
Diary zur Verfügung gestellt *)
invariant
    !t.(t:Session => dom(ru_CR-M(t)) = dom(info_CR-M(t)))
    ;(* für alle benutzten Konferenzräume existiert auch
     Information zur Sitzung *)
operations
    Book-Conf-Room_0(session,cr,m,si) =
        pre
            session : Session &
            cr : CR &
            m : M &
            si : SI
        then
            R<U-Book(session,cr,m) ||
            Add(session,cr,si)
        end;
```

```
    Cancel-Conf-Room0(session,cr,m,si) =
        pre
            session : Session &
            cr : CR &
            m : M &
            si : SI &
            !ti.(ti:session => dom(info_CR-M(ti)) = (ru_CR-M(ti)^-1)(m))
            ;(* zu allen Zeiten ist die Menge der Konferenzräume,
             für die Information aufgezeichnet ist, gleich der
             von den Tagungen m benutzten Räume *)
        then
            R<U-Del-User(session,cr,m) ||
            Erase(session,cr,si)
        end;
end
```

G. Das Speisesaal-Tagung-Subsystem DR-M

```
context
    DR-M-context
sets
    Date;   (* Date ist Menge von Zeitpunkten *)
    DR;     (* DR ist Menge von Speisesälen *)
    M;      (* M ist Menge von Tagungen *)
    LI;     (* LI ist Menge von Essensinformation *)
end


machine
    DR-M
sees
    DR-M-context
uses
    R<U (Date, DR, M);
    Diary (Date, M, LI)
variables
;(* ru_DR-M und info_DR-M werden durch Parametrisierung von R<U und
Diary zur Verfügung gestellt *)
    max-no, ;(* bezeichnet die Kapazität pro DR *)
    rsvd,   ;(* enthält die Anzahl schon reservierter Plätze *)
    amount  ;(* Anzahl zu belegender oder zu löschender Plätze *)
invariant
    max-no inc DR --> N &
        ;(* Diese Funktion enthält die maximale Anzahl von Plätzen
         pro Speisesaal *)
    amount inc Date +-> N &
        ;(* zu reservierende(löschende) Plätze zur Zeit t *)
    rsvd inc (Date --> (DR +-> N)) &
        ;(* reservierte Plätze zur Zeit t *)
    !t.(t:Date => dom(ru_DR-M(t)) = dom(info_DR-M(t))) &
        ;(* für die gebuchten Speisesäle existiert Information *)
    !ti.(ti:Date => dom(rsvd(ti))=dom(ru_DR-M(ti)) &
        !r.(r:dom(ru_DR-M(ti)) => rsvd(ti)(r) <= max-no(r) ) )
         ;(* alle DR, in denen Plätze reserviert sind,
         werden von einer Tagung belegt und für alle belegten DR
         ist die Anzahl der reservierten Plätze kleiner als
         die angegebene Kapazität *)
```

```
operations
    Book-Dining-Room0(date,dr,m,li) =
        pre
            date : Date &
            dr : DR &
            m : M &
            li : LI &
            !ti.(ti : date =>
                        rsvd(ti)(dr)+amount(ti) <= max-no(dr))
            ;(* wenn die zu reservierenden Plätze noch frei sind *)
        then
            Book-Dining-Room0 := R<U-Book(date,dr,m) ||
            Add(date,m,li) ||
            rsvd(date) := rsvd(date) <+
                 { dr|->(rsvd(date)(dr) + amount(date))}
            ;(* Die Anzahl der reservierten Plätze pro Zeit erhöht
             sich um die angegebene Zahl amount *)
        end;
    Cancel-Dining-Room0(date,dr,m,li) =
        pre
            date : Date &
            dr : DR &
            m : M &
            li : LI &
            !ti.(ti:date => dom(infoDR-M(ti)) = rng(ruDR-M(ti)) )
            ;(* für alle Tagungen im Speisesaal sind für
             die Zeiten ti Informationen gespeichert *)
        then
            R<U-Del-User(date,dr,m) ||
            Erase(date,m,li) ||
            rsvd(date):=rsvd(date) <+
             { dr|->rsvd(date)(dr) - amount(date)}
            ;(* Die Anzahl der reservierten Plätze pro Zeit
             verringert sich um die angegebene Zahl amount *)
        end;
end
```

H. Der Besucher-Pool V-P

```
context
    V-P-context
sets
    Date;   (* Date ist Menge von Zeitpunkten *)
    V;      (* V ist Menge von Besuchern *)
end
```

```
machine
    V-P
sees
    V-P-context
uses
    Pool (Date, V)
variables
;(* exists_V-P wird durch Parametrisierung zur Verfügung gestellt *)
operations
    Create-Visitor_0(date,v) =
        pre
            date : Date &
            v : V
        then
            Create(date,v)
        end;
    Destroy-Visitor_0(date,v) =
        pre
            date : Date &
            v : V
        then
            Destroy-Visitor(date,v)
        end;
end
```

IV. Das vollständige CAVIAR-System

A. Der Visitor-Teil V-SYS

```
machine
    V-SYS
uses
    V-P,
    HR-V,
    TR-V
variables
    date-of-time, date
invariant
    date-of-time inc Time --> Date &
    !date.(date:Date => rng(ru_HR-V(date)) inc exists_V-P(date)) &
    !t.(t:Time => rng(ru_TR-V(t)) inc exists_V-P(date-of-time(t))
    ;(* Besucher, die Hotel oder Transport gebucht haben,
     müssen existieren *)
end
```

B. Der Meeting-Teil M-SYS

```
machine
    M-SYS
uses
    M-V,
    CR-M,
    DR-M
variables
    date-of-session
invariant
    date-of-session inc Session --> Date &
    !s.(s:Session => rng(ru_CR-M(s)) inc exists_M-V(s)) &
    !date.(date:Date => rng(ru_DR-M(date)) inc
     union{s | s:Session & date-of-session(s) = d.exists_M-V(s)} )
    ;(* Tagungen, die einen Konferenzraum oder Speisesaal belegen,
     müssen existieren *)
end
```

C. Das Gesamtsystem CAVIAR

```
machine
    CAVIAR
uses
    V-SYS,
    M-SYS
invariant
    !s.(s:Session => rng(ru_M-V(s)) inc
         exists_M-V(date-of-session(s)))
    ;(* Alle Besucher, die eine Tagung besuchen,
     müssen dem System bekannt sein *)
end
```

Anhang B: Das Subsystem CR-M als Abstrakte Maschine mit Attributverwaltung

1. Die Abstrakte Maschine CR-attributes

```
context
    CR-attr-ctx
sees
    CAVIAR-sets
    CR-M-context
sets
    GEBAEUDE = N;
    STOCKWERK = N;
    PLATZANZAHL = N;
    EINRICHTUNG = CR-AUSSTATTUNG
end

machine
    CR-attributes
sees
    CR-attr-ctx
variables
    cr, gebaeude, stockwerk, platzanzahl, einrichtung
invariant
    cr inc CR;
    gebaeude : cr --> GEBAEUDE;
    stockwerk : cr --> STOCKWERK;
    platzanzahl : cr --> PLATZANZAHL;
    einrichtung : cr --> POW(EINRICHTUNG)
initialization
    cr, gebaeude, stockwerk, platzanzahl, einrichtung :=
{},{},{},{},{}
operations
    k <-- create_konf_raum(g,s,p,e) =
        pre
            g : GEBAEUDE &
            s : STOCKWERK &
            p : PLATZANZAHL &
            e : EINRICHTUNG
        then
            any r where
                r : CR - cr
            then
                cr := cr \/ {r} ||
                gebaeude(r) := g ||
                stockwerk(r) := s ||
                platzanzahl(r) := p ||
                einrichtung(r) := e ||
                k := r
            end;
        end;
```

```
    destroy_konf_raum(k) =
        pre
            k : cr
        then
            cr := cr - {k} ||
            gebaeude := (cr - {k}) <| gebaeude ||
            stockwerk := (cr - {k}) <| stockwerk ||
            platzanzahl := (cr - {k}) <| platzanzahl ||
            einrichtung := (cr - {k}) <| einrichtung
        end;
    modify_platzanzahl(k,p) =
        pre
            k : cr &
            p : PLATZANZAHL
        then
            platzanzahl := platzanzahl <+ {k |-> p}
        end;
    modify_einrichtung(k,e) =
        pre
            k : cr &
            e : EINRICHTUNG
        then
            einrichtung := einrichtung <+ {k |-> e}
        end;
    v <-- gebaeude_(k) =
        pre
            k : cr
        then
            v := gebaeude(k)
        end;
    v <-- stockwerk_(k) =
        pre
            k : cr
        then
            v := stockwerk(k)
        end;
    v <-- platzanzahl_(k) =
        pre
            k : cr
        then
            v := platzanzahl(k)
        end;
    v <-- einrichtung_(k) =
        pre
            k : cr
        then
            v := einrichtung(k)
        end;
end
```

2. Die Abstrakte Maschine M-attributes

```
context
    M-attr-ctx
sees
    CAVIAR-sets,
    CR-M-context
sets
    THEMA = STRING;
    REFERENT = {Müller, Lehmann, Schulze};
    LEHRMITTEL = CR-AUSSTATTUNG;
    GRÖSSENORDNUNG = {klein, mittel, umfassend};
    SESSION = POW(DATUM x ZEITRAUM);
            ;(* Eine Tagung kann an mehreren Tagen stattfinden *)
end

machine
    M-attributes
sees
    M-attr-ctx
variables
    tagung, thema, referent, lehrmittel, grössenordnung
invariant
    tagung inc TAGUNG;
    thema : tagung --> THEMA;
    referent : tagung --> REFERENT;
    lehrmittel : tagung --> POW(LEHRMITTEL);
    grössenordnung : tagung --> GRÖSSENORDNUNG
initialization
    thema, referent, lehrmittel, grössenordnung := {},{},{},{};
operations
    m <-- create_tagung(t,r,l,g) =
        pre
            t : THEMA &
            r : REFERENT &
            l : POW(LEHRMITTEL) &
            g : GRÖSSENORDNUNG
        then
            any s where
                s : M - tagung
            then
                tagung := tagung \/ {s} ||
                thema(s) := t ||
                referent(s) := r ||
                lehrmittel(s) := l ||
                grössenordnung(s) := g ||
                m := s
            end;
        end;
    destroy_tagung(m) =
        pre
            m : tagung
        then
            tagung := tagung - {k} ||
            gebaeude := (cr - {k}) <| gebaeude ||
            stockwerk := (cr - {k}) <| stockwerk ||
            platzanzahl := (cr - {k}) <| platzanzahl ||
            einrichtung := (cr - {k}) <| einrichtung
        end;
```

```
    modify_thema(m,t) =
        pre
            m : tagung &
            t : THEMA
        then
            thema := thema <+ {m |-> t}
        end;
    modify_referent(m, oldref, newref) =
        pre
            m : tagung &
            oldref : REFERENT &
            newref : REFERENT &
            !m_i.((m_i:m) => (referent(m_i) = oldref))
        then
            referent := referent <+ {m |-> newref}
        end;
    modify_lehrmittel(m, l) =
        pre
            m : tagung &
            l : POW(LEHRMITTEL)
        then
            lehrmittel := lehrmittel <+ {m |-> l}
        end;
    modify_grössenordnung(m, g) =
        pre
            m : tagung &
            g : GRÖSSENORDNUNG
        then
            grössenordnung := grössenordnung <+ {m |-> g}
        end;
    v <-- thema_(m) =
        pre
            m : tagung
        then
            v := thema(m)
        end;
    v <-- referent_(m) =
        pre
            m : tagung
        then
            v := referent_(m)
        end;
    v <-- lehrmittel_(m) =
        pre
            m : tagung
        then
            v := lehrmittel(m)
        end;
    v <-- grössenordnung_(m) =
        pre
            m : tagung
        then
            v := grössenordnung(m)
        end;
end
```

3. Die Abstrakte Maschine CR-M-attributes

```
context
    CR-M-attr-ctx
sees
    CR-attr-ctx,
    M-attr-ctx
sets
    SESSION = ZEITRAUM;
end

machine
    CR-M-attributes
sees
    CR-M-attr-ctx
uses
    CR-attributes,
    M-attributes,
    CR-M
promotes
    create_konf_raum, destroy_konf_raum,
    modify_platzanzahl, modify_einrichtung,
    gebaeude_, stockwerk_, platzanzahl_, einrichtung_,
    create_tagung, destroy_tagung,
    modify_thema, modify_referent, modify_lehrmittel,
    modify_grössenordnung,
    thema_, referent_, lehrmittel_, grössenordnung_
invariant
    dom(ran(ru_CR-M)) inc tagung;
    ran(ran(ru_CR-M)) inc cr;
    !(c,m). ((c,m) : ran(ru_CR-M) =>
            ((c inc cr) & (m inc tagung) &
            ((grössenordnung(m) = klein) =>
                        (platzanzahl(c) <= 20) &
             (grössenordnung(m) = mittel) =>
                        ((platzanzahl(c) > 20) &
                           (platzanzahl(c) < 80)) &
             (grössenordnung(m) = umfassend) =>
                        (platzanzahl(c) >= 80) ) )))
operations
    r <-- buche_cr_raum(m,s) =
        pre
            m : M &
            s : SESSION
        then
            any z where
                z : CR - cr &   ;(* z muß existieren *)
                !s_i.((s_i:s) =>
                 (!m_i.((m_i:M) => NOT({z|->m_i}: ru_CR-M(s_i))))) &
                                  ;(* z muß auch frei sein *)
                lehrmittel(m) inc einrichtung(z)
            then
                r := z ||
                Book-Conference-Room_0(s,z,m)
            end
        end;
```

```
    b <-- exist_cr(m) =
        pre
            m : tagung
        then
            any z where
                z : CR - cr &
            then
                if einrichtung(z) = lehrmittel(m)
                then
                    b := TRUE
                else
                    b := FALSE
                end;
            end;
        end
end
```

Literaturverzeichnis

[Abr89a] J. R. Abrial
Abstract Machines (Part 1: Basic Concepts)
August, 1989; Report

[Abr89b] J. R. Abrial
Introduction To Set Notations (Part 1 : Basic Concepts)
May, 1989; Report

[Abr89c] J. R. Abrial
A Formal Approach to Large Software Construction
in: Mathematics of Program Construction
Springer-Verlag
LNCS 375, 1989; Groningen; P. 1-20

[Abr90] J. R. Abrial
devB User Manual, Draft #2
1990; Report

[Abr91] J. R. Abrial
Abstract Machines, Part 1: Basic Concepts-Introduction
February, 1991; Ch. 17; Report

[Dij76] E. W. Dijkstra
A Discipline of Programming
Prentice-Hall, Inc.
1976; Englewood Cliffs, N.J.

[Dij90] E. W. Dijkstra
Formal Development of Programs and Proofs
Addison-Wesley
1990; Ch. 9, 10; J. M. Morris

[Fid91] K.-D. Schewe, I. Wetzel et al.
Abstract Machines revisited. FIDE Esprit BRA Project 3070
Technical Report Series
FIDE 91/11

[Hay87] Ian Hayes (ed.)
Specification Case Studies
Prentice-Hall International
1987

[Kin90] Steve King
Z and the Refinement Calculus
IBM UK Laboratories Ltd.
1990; Report

[Spi89] J. M. Spivey
The Z Notation, A Reference Manual
Prentice Hall International
1989