| | |
|---|---|
| **Prüfer:** | Prof. Dr. rer. nat. Jochen Ludewig |
| **Betreuer:** | Dipl.-Inform. Tilmann Hampp |
| **begonnen am:** | 03. Juni 2002 |
| **beendet am:** | 02. Dezember 2002 |
| **CR-Klassifikation:** | D.2.7, D.2.8, E.1, F.2.2, I.6.1, I.6.7 |

Diplomarbeit Nr. 2018

# Performanceanalyse und -verbesserung des SESAM-Simulators

Jörg Franke

Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20 – 22
D – 70565 Stuttgart

## Zusammenfassung

Mit dem SESAM-Simulator (Software Engineering Simulation by Animated Models) wird das Ziel verfolgt, angehenden Projektleitern die Zusammenhänge in Software-Projekten zu vermitteln. Der Simulator führt dazu Modelle aus, die alle relevanten Aspekte der Projekte enthalten.
Beim Einsatz des SESAM-Simulators hat sich ein Laufzeitproblem herausgestellt, welches sich negativ für den Spieler auswirkt, der lange Wartezeiten in Kauf nehmen muss.

In dieser Diplomarbeit wurde die Basismaschine auf ihre Laufzeit untersucht. Die hierdurch gefundenen Schwachstellen waren Grundlage für die Entwicklung von mehreren Verbesserungsvorschlägen. Die im Rahmen dieser Diplomarbeit machbaren Vorschläge, wurden in Produktqualität implementiert und deren Auswirkungen untersucht.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1
# Einführung

*Ziel dieser Arbeit ist es, den SESAM-Simulator in Hinblick auf seine Effizienz zu untersuchen, die Schwachstellen herauszufinden und den, im Rahmen dieser Arbeit, möglichen Teil umzusetzen.*

## 1.1 Eine kurze Einführung in SESAM

SESAM steht für „Software Engineering Simulation by Animated Models“ und ist der Name eines Simulationssystems. SESAM wird hauptsächlich in der Lehre dazu eingesetzt, angehenden Projektleitern die Zusammenhänge innerhalb von Softwareprojekten näher zu bringen. Der Spieler übernimmt dabei die Rolle des Projektmanagers und führt am Simulator Projekte durch.

Die Simulation baut auf Modellen auf, die alle möglichen Projektzustände beschreiben, den Anfangszustand, sowie die Dynamik, die den Zustand ändert. Der Spieler kann durch Kommandos die Simulation beeinflussen und bekommt über Nachrichten Informationen über den aktuellen Zustand. Die Dynamik des Systems wird durch Regeln beschrieben. Durch den Bedingungsteil einer Regel wird festgelegt, ob diese zu einem bestimmten Zeitpunkt aufgrund des Zustands ausgeführt werden kann. Sind alle Bedingungen erfüllt, wird der Aktionsteil der Regel ausgeführt, der Änderungen an dem Zustand vornimmt. Die Simulation läuft in diskreten Zeitschritten ab, wobei in jedem dieser Schritte alle Regel betrachtet werden müssen und gegebenenfalls ausgeführt werden.

## 1.2 Aufgabenstellung

Die folgende Aufgabenstellung ist Grundlage für diese Diplomarbeit.

Ziel der Diplomarbeit ist es, Schwachstellen der Basismaschine bezüglich der Laufszeit festzustellen und Lösungsansätze zur Verbesserung zu entwickeln und umzusetzen. Die im Simulator umgesetzten Konzepte sollen untersucht werden, insbesondere die zentrale Datenstruktur für den Zustand, der Algorithmus zur Suche ausführbarer Regeln und die Ausführung der Regeln. In einem ersten Schritt soll der bisher verwendete Algorithmus analysiert werden, insbesondere die Abhängigkeit der Laufzeit vom Zustand und dem Bedingungsteil der Regeln. Mögliche Schwachstellen sollen durch Simulationsläufe, etwa mit vorhandenen Modellen erkannt und bewertet werden. Davon ausgehend sollen im nächsten Schritt Verbesserungsvorschläge für die Architektur des Simulators entwickelt, mögliche Auswirkungen diskutiert und ihre Machbarkeit untersucht werden. Die im Rahmen der Diplomarbeit umsetzbaren Verbesserungen sollen als Entwurf dokumentiert und in Produktqualität implementiert werden. Tests sollen die Verbesserung der Laufzeit belegen. Die Ergebnisse der Simulation sollen sich im Vergleich zur bisher eingesetzten Basismaschine möglichst nur durch den Einfluss des Zufallsgenerators ändern. Vorschläge, die nicht umgesetzt werden können, sollen dokumentiert werden.

## 1.3 Überblick über die Arbeit

Kapitel 1 bietet eine kurze Einführung in die Thematik der Arbeit und beinhaltet die zugrundeliegende Aufgabenstellung.

Kapitel 2 geht auf die Hintergründe von SESAM ein, dies beinhaltet auch, dass Begriffe genauer definiert werden. Außerdem werden die derzeit eingesetzten Modelle beschrieben und die Funktionsweise der Basismaschine wird aufgezeigt.

Kapitel 3 beschäftigt sich mit der Analyse der Basismaschine. Zuerst wird die Regelinstanzsuche einer Komplexitätsuntersuchung unterzogen, die durch Effizienzmessungen vervollständigt wird. Daraus werden dann die Schwachstellen der Basismaschine abgeleitet.

Kapitel 4 zählt die erarbeiteten Verbesserungsvorschläge auf, beschreibt und bewertet sie. Daraufhin werden die umzusetzenden Vorschläge ausgewählt.

Kapitel 5 enthält den Entwurf der zuvor ausgewählten Vorschläge.

Kapitel 6 beschreibt das Vorgehen während der Implementierung und die aufgetretenen Probleme. Außerdem werden die durchgeführten Unit- und Regressionstests beschrieben sowie deren Ergebnisse.

Kapitel 7 vergleicht die Ergebnisse der Effizienzmessungen vor und nach den Veränderungen und zeigt damit das Ergebnis der Diplomarbeit.

Kapitel 8 fasst den Projektverlauf zusammen und versucht eine abschließende Bewertung der Arbeit.

# Kapitel 2
# Sesam und die Basismaschine

*Dieses Kapitel dient dazu, Hintergrundinformationen zu SESAM und der zugrundeliegenden Theorie zu liefern. Außerdem wird der Aufbau der Modelle und die Eigenschaften von verschiedenen Modellen beschrieben. Am Ende wird genauer auf die Funktionsweise der Basismaschine eingegangen.*

## 2.1 Aufbau des SESAM-Systems

Das Hauptziel, das mit dem Einsatz von SESAM verfolgt wird, ist die Bereitstellung einer möglichst realistischen Simulation von Softwareprojekten, die angehenden Projektleitern als Ergänzung zu ihrer Ausbildung dient. Sie sollen spielerisch ihr gelerntes Wissen anwenden können und durch die somit erfahrenen Zusammenhänge vertiefen. Als weiteres Ziel soll es möglich sein, ungesicherte Hypothesen zu modellieren und durch Nachspielen der Simulation die Konsequenzen zu studieren.

Das SESAM-System ist aus verschiedenen Komponenten aufgebaut, die von unterschiedlichen Rollen benutzt werden:

- Modellbauer:
  Er erstellt die Modelle, die durch die Basismaschine simuliert werden. Durch die Modelle legt er fest, welche Objekte und Beziehungen eines Software-Projekts abgebildet werden und welche Effekte auftreten können. Er definiert somit die Welt, in der sich der Spieler bewegen kann. Die Modelle werden als Hochsprachen-Modelle erstellt und dann in Basissprachen-Modelle übersetzt.
- Spieler:
  Der Spieler übernimmt den, nicht modellierten, Part des Projektleiters. Er greift über Kommandos in den Verlauf der Simulation ein und erhält Nachrichten über den aktuellen Zustand des Systems.
- Tutor:
  Der Tutor führt Schulungen durch, an denen die Spieler teilnehmen. Dabei leitet er die Spieler nicht nur an, sondern hat auch die Möglichkeit, über Auswertungswerkzeuge, die gespielten Simulationen nachzuvollziehen, um mit den Spielern die erreichten Resultate zu diskutieren.

Das Folgende Schaubild zeigt die Rollen und ihre Interaktion mit den Komponenten des Systems.

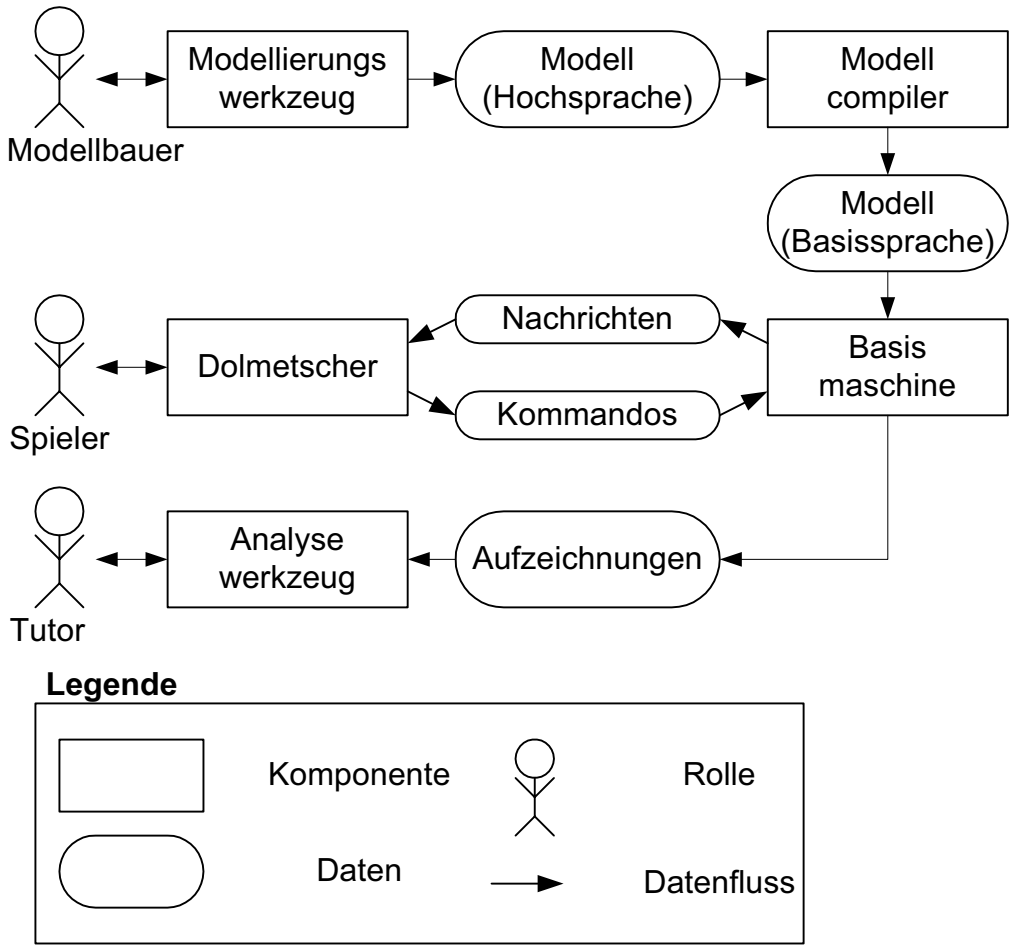



**Abbildung 1. Komponenten und Rollen des SESAM-Systems**

## 2.2 Konzepte

Systemsimulation stellt eine Methode zur Lösung von Problemen dar, bei der man die Änderungen eines dynamischen Systemmodells über der Zeit verfolgt [Gordon, 1969].

Bei der Durchführung von Systemsimulationen kann man grob drei Schritte voneinander abgrenzen:

- die Modellbildung oder auch Problemdefinition,
- Modell- oder Simulationsexperimente,
- die Ergebnisanalyse.

Ein rechnergestütztes Simulationssystem ist nach [Page, 1991] ein Softwaresystem, das die Bearbeitung der drei Aufgabenbereiche im Rahmen einer Simulationsstudie unterstützt.

### 2.2.1 Art der Simulation

Bei SESAM handelt es sich um eine diskrete Simulation. Hierbei erfolgen die Zustandsänderungen sprunghaft zu diskreten Zeitpunkten. Eine solche Zustandsänderung wird durch das Eintreten eines atomaren, das heißt keine Simulationszeit verbrauchenden Ereignisses verursacht. Zum besseren

Verständnis soll diese Simulationsmethode gegen die kontinuierliche abgegrenzt und zusätzliche Eigenschaften der Simulation von SESAM beschrieben werden.

Im Gegensatz zu der diskreten geht man bei der kontinuierlichen Simulation davon aus, dass sich der Zustand des Modells stetig mit der Zeit verändert. Das Systemverhalten wird durch eine Menge verkoppelter Gleichungen charakterisiert, deren freie Variable die Zeit ist. Ausgehend von einem Anfangszustand lässt sich so der Zustand zu einem späteren Zeitpunkt berechnen.

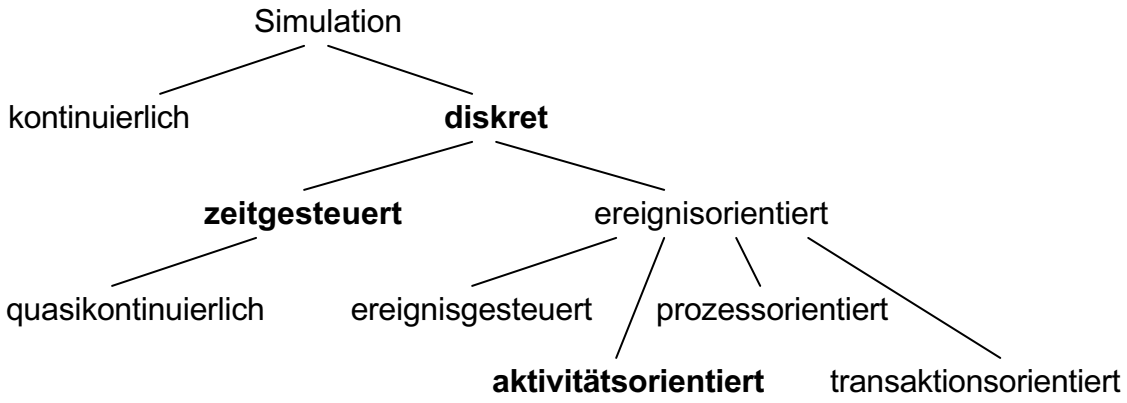



**Abbildung 2. Klassifikation von Simulationsmethoden**

Weitere Eigenschaften der diskreten Simulation von SESAM sind, dass die Simulation zum einen zeitgesteuert abläuft und zum anderen, dass sie aktivitätsorientiert ist. Zeitgesteuert deshalb, da die Modellzeit in konstante Zeitschritte unterteilt ist. Aktivitätsorientiert, weil nach Ablauf eines Zeitschritts geprüft wird, welche Aktivitäten ausgeführt werden können. Die Größe des Zeitschritts $\Delta t$ ist von entscheidender Bedeutung für Korrektheit, Genauigkeit und Effizienz der Simulation: sie sollte einerseits klein genug sein, damit Zustandsänderungen eines Objekts sich auf andere Objekte erst in einer der nächsten Zeitepochen auswirken, da sonst die Reihenfolge, in der die Zustandsänderungen der Objekte durchgeführt werden, Einfluss auf die Simulation haben kann. Andererseits sollte sie möglichst groß sein, damit wenige Zustandstransformationen durchgeführt werden müssen und die Simulation durch ein schnelles Voranschalten der Simulationsuhr effizient abläuft.

Zitiert nach [Page, 1991] und [Mattern, 1989].

### 2.2.2 Zeitbegriffe in SESAM

Bei der Simulation wird die reale Zeit (**Originalzeit**) eines Softwareprojekts auf die **Modellzeit** abgebildet, die möglichst nahe an der Realität liegen soll. Somit können dem Spieler zeitliche Abläufe verdeutlicht werden. Die **Ausführungszeit** ist die Zeit, die zur Ausführung der Simulation benötigt wird. Durch sie wird das Ziel verfolgt die Modellzeit zu stauchen, da es nicht praktikabel ist, die Simulation die reale Zeit eines Softwareprojektes dauern zu lassen.

Die Modellzeit ist in Zeitschritte unterteilt. Bei jeder Aktion, die der Spieler durchführt, wird Modellzeit verbraucht. Wenn seit dem letzten Zeitschritt die durch den Spieler verbrauchte Modellzeit größer als ein solcher Zeitschritt ist, wird der Regelauswertungszyklus gestartet. Der Spieler kann

aber auch ein oder mehrere Zeitschritte vorschalten, wobei in jedem Schritt der Regelauswertungszyklus ausgeführt wird.

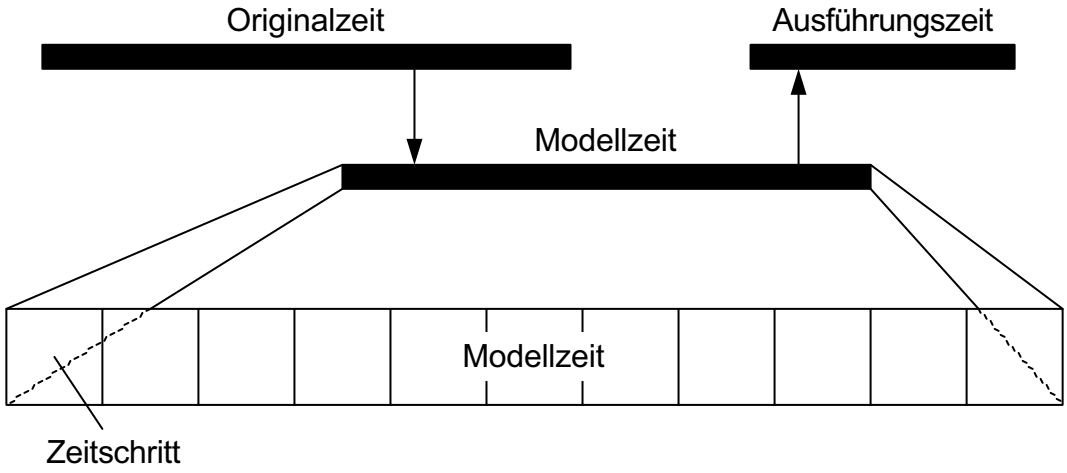



**Abbildung 3. Zeitbegriffe**

### 2.2.3 Aufbau der Modelle

Auf die Modelltheorie und die Merkmale von Modellen wird hier nicht näher eingegangen. Dies kann bei [Stachowiak, 1973] nachgelesen werden.

Bei SESAM-Modellen unterscheidet man zwischen den Hochsprachen-Modellen, die der Modellbauer erstellt und den Basissprachen-Modellen, die aus den Hochsprachen-Modellen durch den SEMOS-Compiler erzeugt werden. Hochsprachen-Modelle enhalten viele Sprachkonzepte, die es dem Modellbauer erleichtern, die Modelle zu entwickeln. Die Basissprachen-Modelle sind dagegen für die Ausführung durch den SESAM-Simulator optimiert.

Jedes Modell besteht aus Schema-, Regel- und Situations-Modell.

**Schema-Modell**

Zu Beginn werden im Schema-Modell die Typen festgelegt:

- alle an einem Projekt beteiligten Typen von Objekten und ihre Attribute (die Entitätstypen),
- alle möglichen Beziehungen (die Relationstypen) zwischen diesen Objekten,
- alle verwendeten Attribute, die die Eigenschaften der Entitäten und Relationen beschreiben.

Darüberhinaus werden Nachrichten definiert, die der Spieler während der Simulation erhält.

**Regel-Modell**

Das Regel-Modell stellt den Kern der Dynamik des Modells dar und legt alle Regeln und Benutzerkommandos fest.

Abhängig von den Regeln und dem Zustand wird, während jedes Simulationsschrittes, der Zustand des Systems neu berechnet.

Die Hauptbestandteile einer Regel sind der Bedingungsteil und der Aktionsteil. Der Bedingungsteil enthält die Vorbedingungen, die erfüllt sein müssen, damit eine Regel ausgeführt werden kann. Der

Aktionsteil enthält alle Aktionen, die durch die Regel ausgelöst werden können. Benutzerkommandos entsprechen in ihrem Aufbau einer Regel und werden hier nicht weiter besprochen.

Der Bedingungsteil einer Regel besteht aus drei Komponenten, die festlegen, was im Zustand enthalten sein muss, damit die Regel ausgeführt werden kann:

- Formale Entitäten:
  Diese bestehen aus einem Namen und einem Entitätstyp. Dadurch ist festgelegt, welche Entitäten im Zustand (State) des SESAM-Simulators enthalten sein müssen.
- Formale Relationen:
  Diese bestehen aus einem Namen und einem Relationstyp. Dadurch ist festgelegt, welche Relationen im Zustand (State) des SESAM-Simulators enthalten sein müssen.
- Attributsbedingungen:
  Diese definieren beliebige Beziehungen zwischen Attributen, die erfüllt sein müssen.

Der Aktionsteil enthält Ausdrücke, die Entitäten oder Relation erzeugen oder löschen, Nachrichten erzeugen, oder Veränderungen an Attributen vornehmen.

In Hochsprachen-Modellen unterscheidet man bei Regeln außerdem zwischen:

- Aktivierungs-/Deaktivierungsregeln (Aktivitäten):
  Sie ermöglichen die über einen längeren Zeitraum stattfindenden kontinuierlichen Änderungen des Zustands. Mit ihrer Hilfe kann der Modellbauer die Komponenten des Regelmodells (Aktivitäten, Regeln und Benutzerkommandos) hierarchisch strukturieren. Eine Aktivität kann aus weiteren Aktivitäten, Regeln und Benutzerkommandos bestehen.
- Feuerungsregeln:
  Diese sorgen dagegen für die zu einem bestimmten Zeitpunkt stattfindenden diskreten Änderungen am Zustand.

In der Basissprache fehlt das Konzept der Aktivitäten. Dies wird durch die Aufspaltung einer Aktivität in mehrere Regeln auf verschiedenen Prioritätsstufen erreicht. Es wird dabei eine Regel für den Aktivierungsteil (Aktivierungsregel), eine Regel für den Aktivteil (Aktivregel) und eine Regel für jede Anweisung des Deaktivierungsteils (Deaktivierungsregeln) der Aktivität erzeugt.

Es sind Prioritäten von 2000, der höchsten, bis zu 0, der niedrigsten, möglich. Dabei sind unterschiedliche Bereiche für Aktivitäten und Regeln reserviert. Die Prioritäten verteilen sich auf verschiedene Arten von Regeln und Aktivitäten wie folgt:

2000 - 1201: Initialisierung des Zeitschritts:

- 2000 - 1810: Aktionsteil einer Regel oder Aktivierungsteil einer Aktivität,
- 1800 - 1610: Aktivteil einer Aktivität,
- 1600 - 1201: Deaktivierungsregel.

1000 - 201: Aktivitäten und Regeln außerhalb der Initialisierung:

- 1000 - 810: Aktionsteil einer Regel oder Aktivierungsteil einer Aktivität,
- 800 - 610: Aktivteil einer Aktivität,
- 600 - 201: Deaktivierungsregel.

**Situations-Modell**

Die Startsituation ist die Abbildung der Anfangssituation eines realen Projektes.

- Es erzeugt konkrete Ausprägungen von im Schemamodell definierten Entitäts- und Relationstypen.
- Es enthält alle Informationen, die dem Spieler zu Beginn der Simulation über das Projekt mitgeteilt werden können.

Die Startsituation stellt somit alle Bedingungen auf, die zu Beginn der Simulation existieren.

Bei der Verwendung des Begriffs Situationsmodell müssen zwei Situationen unterschieden werden. Er kann einerseits für die Abbildung der Startsituation eines Projektes, aber auch zur Darstellung der momentanen Situation während einer Simulation verwendet werden.

Die folgende Abbildung zeigt den Zusammenhang zwischen Schema-, Regel- und Situations-Modell [Schneider, 1994].

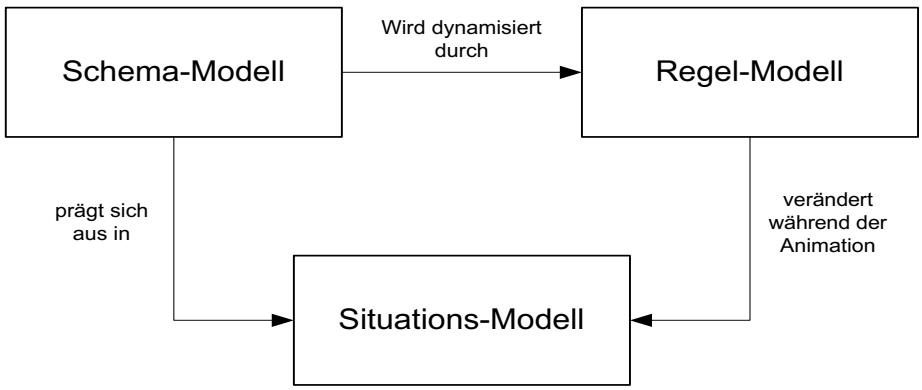



**Abbildung 4. Komponenten der Modelle**

## 2.3 SESAM-Modelle

Die Modelle, die für Messungen an der Basismaschine herangezogen wurden, werden im Folgenden kurz beschrieben. Es werden jeweils Kennzahlen der Modelle erhoben, wie zum Beispiel die Anzahl der Regeln, der Kommandos oder der Entitätstypen.

Die Kennzahlen dienen dazu, die Bedeutung der einzelnen Elemente der Komplexitätsuntersuchung abschätzen zu können.

### 2.3.1 Qualitätssicherungs-Modell

Das Qualitätssicherungs-Modell oder auch QS-Modell wurde von Drappa während ihrer Dissertation [Drappa, 2000] entwickelt. Das Hauptaugenmerk liegt bei den Maßnahmen zur Qualitätssicherung. Es erfordert von dem Spieler ein richtiges Vorgehen bei der Besetzung von Stellen sowie der Fortschritts und Qualitätskontrolle. Es müssen die Analyse, die Spezifikation, der Grob- und Feinentwurf, der Code, das Handbuch erstellt, sowie die Integration und die Auslieferung des Systems durchgeführt werden. Zusätzlich müssen alle Dokumente einem Review unterzogen, als auch Modul-, Integrations, System- und Abnahmentest durchgeführt werden. Der Spieler ist außerdem dafür verantwortlich, Mitarbeiter anzustellen, ihnen Aufgaben zuzuweisen und zu entlassen.

### 2.3.2 Qualitätssicherungs-Modell erweitert um Verhaltensaspekte

Das QSVA-Modell baut direkt auf dem QS-Modell auf und enthält somit auch alle Effekte des QS-Modells. Es erweitert es jedoch um wichtige Verhaltensaspekte, wie die Motivation von Mitarbeitern. Der Spieler muss dabei besonders auf die Personalführung und die Repräsentation des Projektes nach außen achten. Erschwerend kommen noch zufällige Ereignisse hinzu, wie die Kündigung oder die Krankheit eines Mitarbeiters.

### 2.3.3 Feingranulare Variante des Qualitätssicherungs-Modells

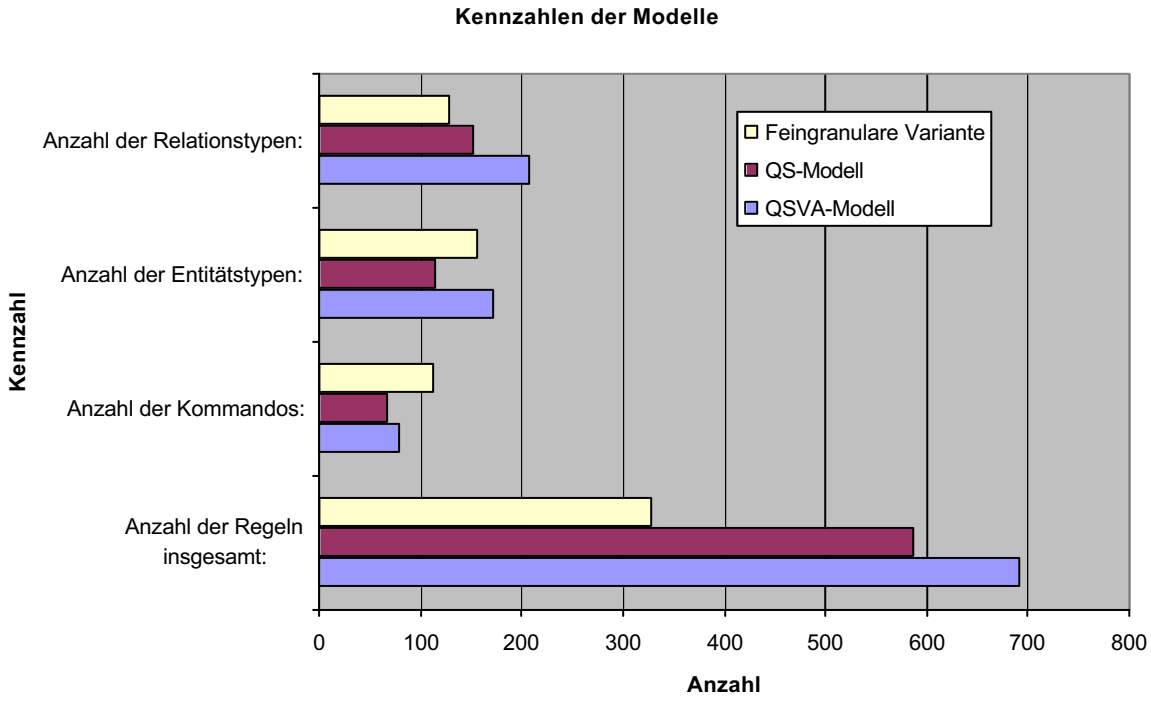
Die Feingranulare Variante des QS-Modells verringert die Schrittweite des Modells von einem Tag auf zwei Stunden. Somit wird es möglich, feingranulare Effekte, wie die Vorbereitung von Reviews zu modellieren und zu beobachten.

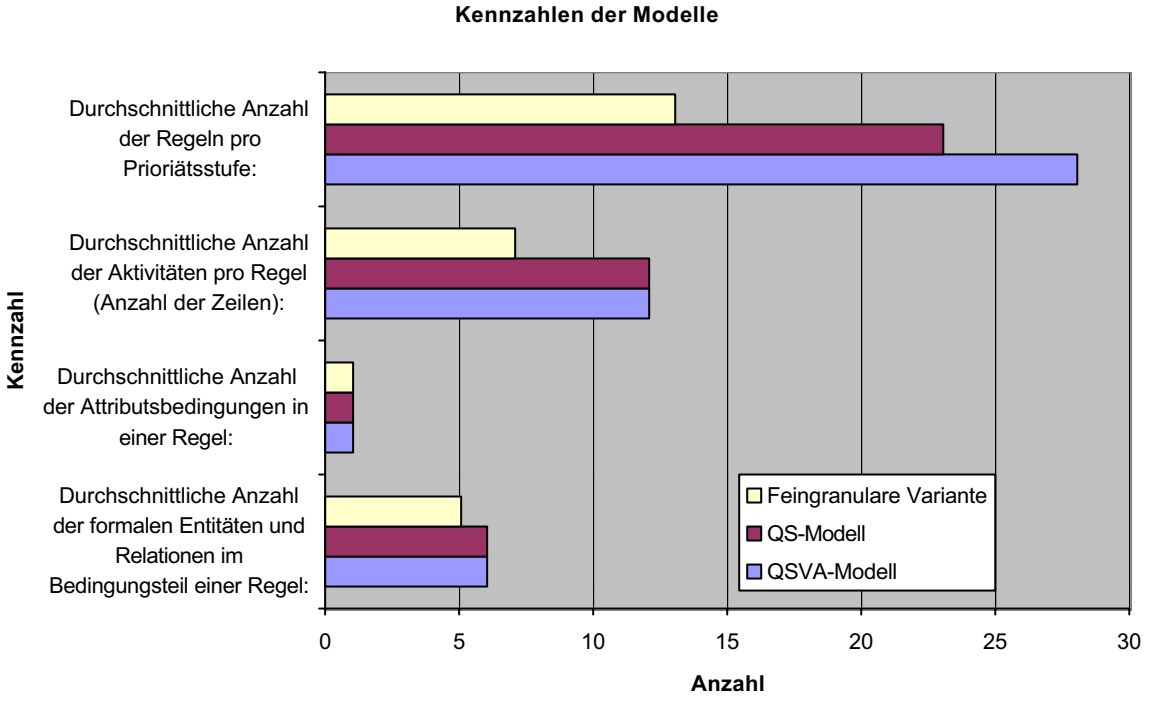
### 2.3.4 Kennzahlen der Modelle

Im Folgenden werden die Kennzahlen der Modelle in einer Tabelle dargestellt und zur besseren Übersicht in Schaubildern gegenübergestellt.

| **Name des Modells:** | QS-Modell | QSVA-Modell | Feingranulare Variante |
|---|---|---|---|
| **Anzahl der Regeln insgesamt:** | 586 | 692 | 327 |
| **Anzahl der Kommandos:** | 68 | 79 | 112 |
| **Anzahl der Entitätstypen:** | 114 | 172 | 156 |
| **Anzahl der Relationstypen:** | 153 | 207 | 128 |
| **Durchschnittliche Anzahl der formalen Entitäten und Relationen im Bedingungsteil einer Regel:** | 6 | 6 | 5 |
| **Durchschnittliche Anzahl der Attributsbedingungen in einer Regel:** | 1 | 1 | 1 |
| **Durchschnittliche Anzahl der Statements im Aktionsteil (Anzahl der Zeilen):** | 12 | 12 | 7 |
| **Durchschnittliche Anzahl der Regeln pro Prioriätsstufe:** | 23 | 28 | 13 |

**Tabelle 1. Kennzahlen der Modelle**


Kennzahlen der Modelle
Anzahl der Relationstypen:
Anzahl der Entitätstypen:
Anzahl der Kommandos:
Anzahl der Regeln insgesamt:
Kennzahl
Feingranulare Variante
QS-Modell
QSVA-Modell
0
100
200
300
400
500
600
700
800
Anzahl


Kennzahlen der Modelle
Durchschnittliche Anzahl der Regeln pro Prioriätsstufe:
Durchschnittliche Anzahl der Aktivitäten pro Regel (Anzahl der Zeilen):
Durchschnittliche Anzahl der Attributsbedingungen in einer Regel:
Durchschnittliche Anzahl der formalen Entitäten und Relationen im Bedingungsteil einer Regel:
Kennzahl
Feingranulare Variante
QS-Modell
QSVA-Modell
0
5
10
15
20
25
30
Anzahl

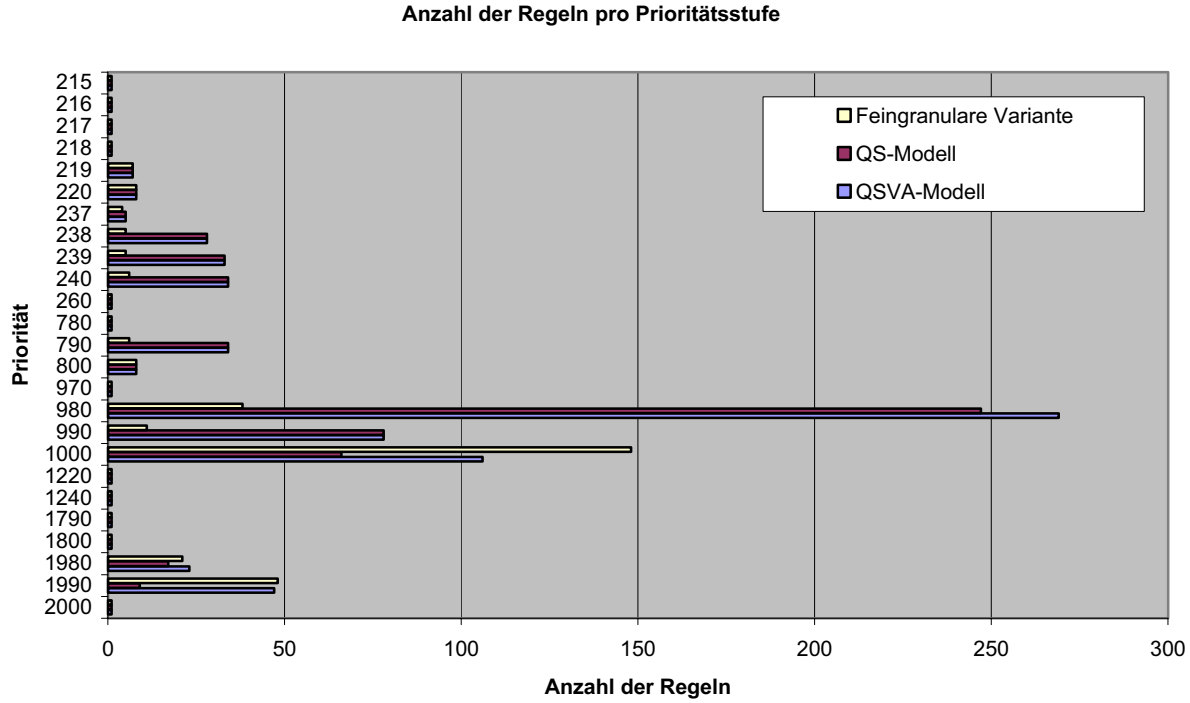



**Abbildung 5. Vergleich der Kennzahlen der einzelnen Modelle**

Bei dem QS-Modell erkennt man, im Vergleich zu den beiden anderen Modellen, eine gleichmäßigere Verteilung der Regeln auf die einzelnen Prioritätsstufen.

Das QSVA-Modell weist zum einen höhere Zahlen in der Stufe der Initialisierung, als auch bei den Aktivierungsregeln und den Aktionsteilen auf. Im Vergleich zum QS-Modell ändern sich die Anzahlen der Regel, auf den verbleibenden Stufen, nicht.

Da die Feingranulare Variante nur einen Ausschnitt des QS-Modells darstellt ist die Anzahl der Regel auf Prioritätsstufen unter 1000 geringer. Besonders fällt hier allerdings die stark erhöhte Anzahl der Regeln auf der globalen Stufen 1000 auf.

## 2.4 Überblick des SESAM-Simulators

Nachdem bereits im ersten Kapitel eine kurze Einführung über den SESAM-Simulator gegeben wurde, soll hier genauer auf den Aufbau und die Funktionsweise eingegangen werden. Insbesondere soll die Architektur der Basismaschine betrachtet werden.

### 2.4.1 Funktionsweise

Die Funktionsweise der Basismaschine soll auf zwei verschiedenen Abstraktionsebenen beschrieben werden. Zum einen wird ein Simulationslauf aus der Sicht des Spielers beschrieben und die Aktionen, die er durchführen kann. Zum anderen wird gezeigt, was in der Basismaschine geschieht und welche Teile betroffen sind. Zu Beginn wird die Architektur der Basismaschine dargestellt.

#### 2.4.1.1 Architektur der Basismaschine

Die Architektur folgt zum einen dem Model-View-Controller Prinzip, wodurch es möglich ist, die eigentliche Funktionalität unabhängig von der Oberfläche zu entwickeln. Zusätzlich gibt es eine Unterteilung in drei Schichten, die Teilbereiche voneinander abtrennen:

- Materialschicht: enthält die eigentlichen Datenstrukturen,
- Werkzeugschicht: alle komplexen Operationen auf den Daten werden definiert,
- Interaktionsschicht: dient der Kommunikation zwischen System und Systemumgebung.

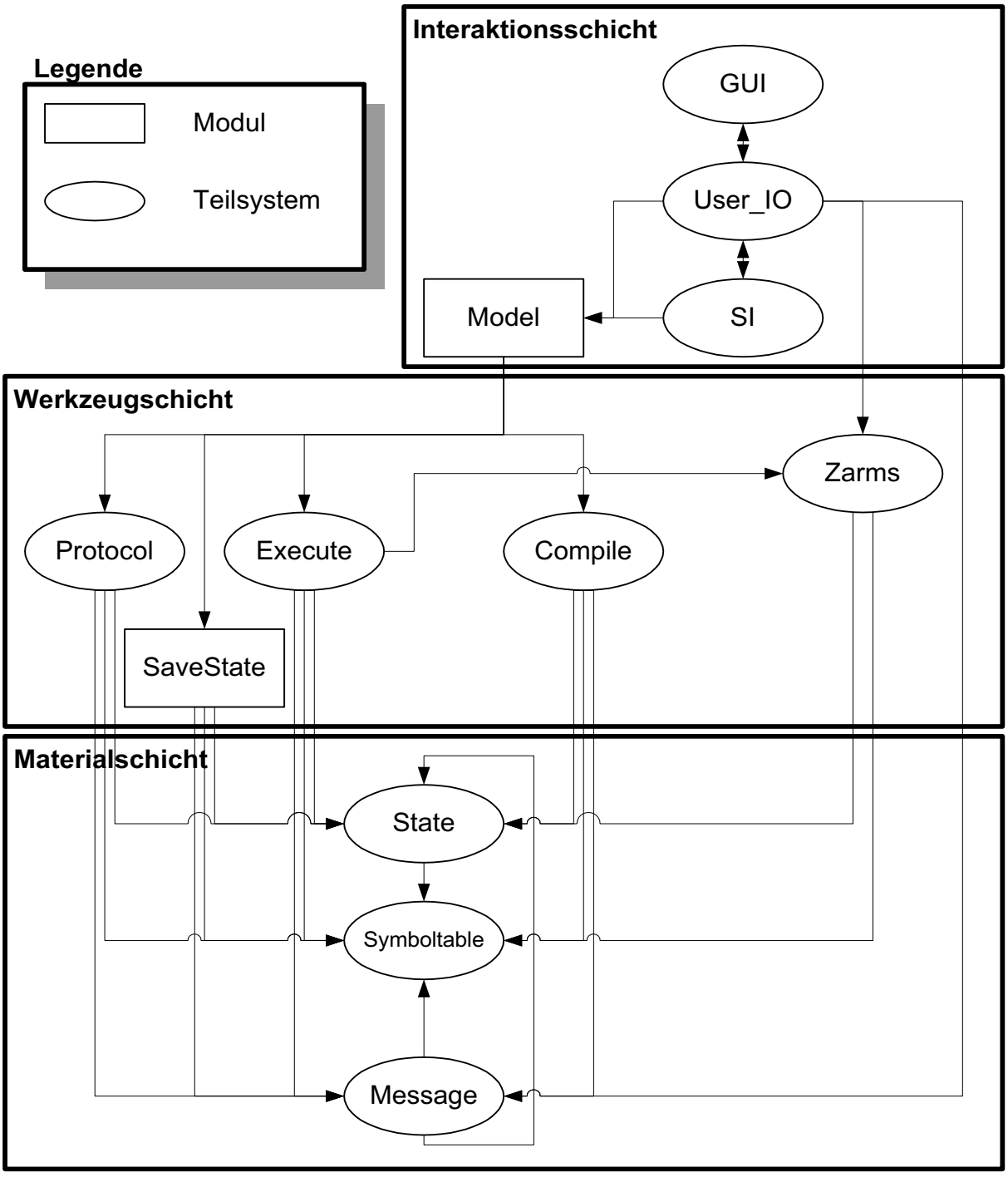



**Abbildung 6. Architektur der Basismaschine**

#### 2.4.1.2 Starten der Simulation

Der Spieler startet den SESAM-Simulator, indem er das Tcl/Tk-Skript “demo_gui” ausführt. Dieses lädt dann die als Bibliothek übersetzte Basismaschine. Die Kommunikation zwischen Tcl/Tk und Ada95 findet über das Teilsystem **User_IO** statt, welches die Schnittstelle definiert. Es werden alle Prozeduren definiert, die von dem Tcl/Tk-Skript aufgerufen werden können. Das Modul **Model** ist eine Sammlung aller Schnittstellenoperationen, die von **User_IO** verwendet werden. Das Teilsystem **SI** (SESAM Interpreter) ist zuständig für die Umformung einer quasi-natürlichsprachlichen Eingabe des Spielers in das interne Command-Format und umgekehrt mit Rückfragen und Paraphrasierungen von Messages in natürliche Sprache.

#### 2.4.1.3 Laden des Modells

Bevor der Spieler irgendeine Aktion durchführen kann, muss er zuerst ein Modell laden. **Compile** liest das Modell mit dem übergebenen Namen ein und baut implizit einen Syntaxbaum auf. Es überträg die folgenden statischen Informationen in die **Symboltable**:

- Attributstypen,
- Entitätstypen,
- Relationstypen,
- Funktionen,
- Prozeduren,
- Regeln,
- Benutzerkommandos.

Zusätzlich zu den statischen Informationen wird die Startsituation geladen und ausgeführt. Ausgeführt wird diese in dem Teilsystem **Execute,** was den eigentlichen Simulator realisiert. Hierbei werden Operationen ausgeführt, die Entitäten erzeugen oder löschen, Relationen erzeugen oder löschen und Attribute verändern.

Informationen zu Entitäten, Relationen und Attributen werden in **State** gespeichert, welches den veränderbaren Zustand des Simulators darstellt.

**Instance Creation Statements**

Jede Regel und jedes Benutzerkommando enthält einen Strukturteil auch Bedingungsteil genannt. In diesem Teil werden die Bedingungen festlegt, die durch den Zustand erfüllt sein müssen, damit eine Regel oder ein Benutzerkommando ausgeführt werden kann. Für jeden dieser Strukturteile wird für die Instanzsuche eine spezielle Anweisungsfolge aus den so genannten Instance Creation Statements oder auch IC_Statements erzeugt. Diese Anweisungsfolgen bestehen aus drei verschiedenen Statements:

- Binde eine formale Entität (Bind_Ent_Stmt):
  Für jede formale Entität, die nicht an einer formalen Relation in der Regel beteiligt ist, wird ein solches Statement erzeugt. Das Statement enthält den Entitätstyp der formalen Entität.
- Binde formale Entität (als Rollenbelegung) ausgehend von einer anderen formalen Entität (als Rollenbelegung) innerhalb derselben formalen Relation (Bind_Ent_Over_Rel_Stmt):
  Durch diese Statements werden formale Relationen aufgelöst. Ist noch keine der an der formalen Relation beteiligten formalen Entitäten durch ein Statement behandelt, wird ein Bind_Ent_Stmt für die erste formale Entität erzeugt. Für die restlichen formalen Entitäten werden Bind_Ent_Over_Rel_Stmts erzeugt, die jede formale Entität über die bereits gebundene formale Entität binden.

- Prüfe eine Attributsbedingung (Check_Stmt):
  Für jede Attributsbedingung wird ein Check_Stmt erzeugt. (Aus Konsistenzgründen wird zusätzlich je ein Check_Stmt für jede formale Relation mit mehr als drei Rollen erzeugt.)

#### 2.4.1.4 Weiterschalten der Simulation

Möchte der Spieler in der Simulation fortschreiten, ohne eine Aktion durchzuführen, so kann er den Simulator anweien, ein oder mehrere Zeitschritte weiterzuschalten. Für jeden dieser Zeitschritte wird ein Regelauswertungszyklus (**Execute**) ausgeführt. In diesem Zyklus werden alle Regeln untersucht, ob sie ausgeführt werden können. Die ausgeführten Regeln nehmen Veränderungen am Zustand vor, erzeugen Nachrichten und verbrauchen Modellzeit.

Ist der Regelauswertungszyklus komplett durchgeführt, werden je nach Einstellung nur die Benutzermessages oder zusätzlich die Tutor- und die Debugmessages ausgegeben. Die Modellzeit wird um einen Zeitschritt weitergeschaltet.

#### 2.4.1.5 Ausführen eines Benutzerkommandos

Um ein Benutzerkommando auszuführen, gibt der Spieler durch einen natürlichsprachigen Text an welches Kommando ausgeführt werden soll und welche Parameter übergeben werden sollen. Zum Beispiel “Lasse Review stattfinden” mit den beiden Entwicklern “Heinz” und “Karl”. Die Entitäten “Heinz” und “Karl” werden dann an das Kommando gebunden und damit versucht, Instanzen mit dem Instanzsuchalgorithmus (**Execute**) zu bilden. Falls diese gefunden wurden, werden diese Ausgeführt, die Nachrichten gesammelt und die verbrauchte Zeit zur Modellzeit addiert. Wurde dadurch seit dem letzten Zeitschritt mehr Zeit verbraucht als ein Zeitschritt dauert, wird ein Regelauswertungszyklus gestartet.

#### 2.4.1.6 Speichern und Laden des Spielstands

Zu jedem Zeitpunkt der Simulation kann der Spieler den aktuellen Stand der Simulation speichern und diese später fortsetzen. In dem Modul **Savestate** wird das Situationsmodell erstellt, das alle Anweisungen enthält, um die aktuelle Situation wiederherzustellen. Zusätzlich wird noch die eingegebene Anweisungsfolge von Benutzerkommandos und Proceedanweisungen gespeichert, was in dem Teilsystem **Protocol** stattfindet. Dies ist sinnvoll für den Tutor, der eine Simulation des Spielers nachvollziehen möchte.

#### 2.4.1.7 Weiteres

Zu Beginn der Simulation kann angegeben werden, ob das so genannte Stateprotocol (**Protocol**) ausgegeben werden soll. Es enthält Informationen über alle Typen des Modell, es werden auch alle Veränderungen des Zustands aufgenommen.

Das Teilsystem **ZARMS** ist eine Auswertungshilfe für den Tutor. Es ermöglicht den aktuellen Zustand mit allen Entitäten und Relationen sowie allen ausgeführten Regeln zu betrachten.

# Kapitel 3
# Analyse der Basismaschine

*Die Basismaschine, insbesondere der Algorithmus zur Suche nach ausführbaren Regeln, wird einer theoretischen Komplexitätsanalyse unterzogen. Diese Analyse wird durch Effizienzmessungen belegt und danach die Schwachstellen der Basismaschine aufgezeigt.*

## 3.1 Einführung

Um das Laufzeitverhalten der Regelauswertung zu untersuchen, werden Komplexität und Effizienz des Algorithmus untersucht.

### 3.1.1 Grundannahme

Bereits im Design des SESAM-Simulators wurde darauf hingewiesen [Kraus, 1999], dass die Hauptkomplexität in Modulen enthalten ist, die für die Suche nach ausführbaren Regeln zuständig sind. Im Entwurf wurden schon mehrere Algorithmen und ihre Vor- und Nachteile gegenübergestellt. Dieselbe Annahme liegt auch der Aufgabenstellung dieser Diplomarbeit zugrunde. Aufgrund der Annahme, dass nur in diesem Teil wesentliche Effizienzverbesserungen zu erreichen sind, wird hier der Algorithmus zur Suche von Regelinstanzen auf seine Komplexität hin untersucht.

### 3.1.2 Begriffsbildung

Bei der Analyse von Algorithmen kann man die Begriffe Effizienz und Komplexität gegeneinander abgrenzen. Beide dienen zur Bewertung von Algorithmen und sind nicht voneinander zu trennen.

Die Effizienz eines Algorithmus wird aufgrund der verbrauchten Betriebsmittel bewertet, wobei bei dieser Untersuchung hauptsächlich Wert auf die benötigte Laufzeit gelegt wird.
Die Komplexität eines Algorithmus wird hingegen durch die Anzahl der notwendigen Operationen in Abhängigkeit von der Größe des zu lösenden Problems beschrieben. Die Komplexität schätzt durch eine untere Schranke die Effizienz von Algorithmen "von unten her" ab. Daher beanspruchen alle eine Funktion realisierenden Algorithmen mindestens den durch die untere Schranke vorgegebenen Aufwand. Die Effizienz hingegen ist immer die obere Schranke für die Komplexität der durch den Algorithmus realisierten Funktion.

Mit der asymptotischen Komplexität, auch Ordnung einer Funktion genannt, kann die obere und untere Schranke eines Algorithmus angegeben werden. Man versteht darunter eine Funktion **f(n)**, die ab einem gewissen $\mathbf{n_0}$ eine obere oder untere Schranke des Algorithmus darstellt. Die Variable **n** stellt die Größe des zu lösenden Problems dar. Die konstanten Faktoren **M** und **L** können dabei vernachlässigt werden, da diese nicht von der Größe des Problems **n** abhängen.

Die obere Schranke wird nach [Knuth, 1997] durch die **O-Notation** beschrieben:

Sei $f: N \rightarrow R$ eine Funktion. Es existieren die beiden positiven Konstanten M und $n_0$, so dass für die Menge O(f) gilt: $|O(f)| \leq M\,|f(n)|$, für alle $n \geq n_0$.

Die untere Schranke wird nach [Knuth, 1997] durch die **Ω-Notation** beschrieben:

Sei $g: N \rightarrow R$ eine Funktion. Es existieren die beiden positiven Konstanten L und $n_0$, so dass für die Menge Ω(g) gilt: $|\Omega(g)| \geq L\,|g(n)|$, für alle $n \geq n_0$.

## 3.2 Theoretische Analyse des Suchalgorithmus

### 3.2.1 Vereinfachende Annahmen der theoretischen Analyse

Wird untersucht, ob eine Regel ausgeführt werden kann, müssen die benötigten Entitäten gesucht werden. Dies geschieht durch die beiden IC_Statements:

- "Bind_Ent_Stmt" welches Entitäten eines bestimmten Typs bindet,
- und "Bind_Ent_Over_Rel_Stmt", das Entitäten bindet, die zusätzlich an einer Relation beteiligt sein müssen.

Zur Vereinfachung der Analyse wird festgelegt, dass bei der Suche nach Entitäten, ausgehend von den IC_Statements "Bind_Ent_Stmt" und "Bind_Ent_Over_Rel_Stmt", jeweils die gleiche Anzahl an Entitäten gefunden wird.

In der Basissprache werden die Regeln in unterschiedliche Prioritätsstufen eingeteilt, um das Aktivitätenkonzept der Hochsprache zu realisieren. Für die theoretische Analyse wird eine durchschnittliche Anzahl von Regeln pro Prioritätsstufe angenommen.

### 3.2.2 Untersuchung des Suchalgorithmus

Zu Beginn der Untersuchung wird ein Callgraph erstellt, der die Abhängigkeit der Regelinstanzsuche im System aufzeigt. Es bestehen hierbei zwei "Kritische Pfade", von denen der eine den Regelauswertungszyklus darstellt. Der zweite "Kritische Pfad" ist die Ausführung eines Benutzerkommandos. Da ein Benutzerkommando und eine Regel dieselbe Struktur aufweisen, ist dieser Pfad ein Spezialfall des Regelauswertungszyklus, wobei nur Instanzen zu einer Regel gefunden werden müssen.

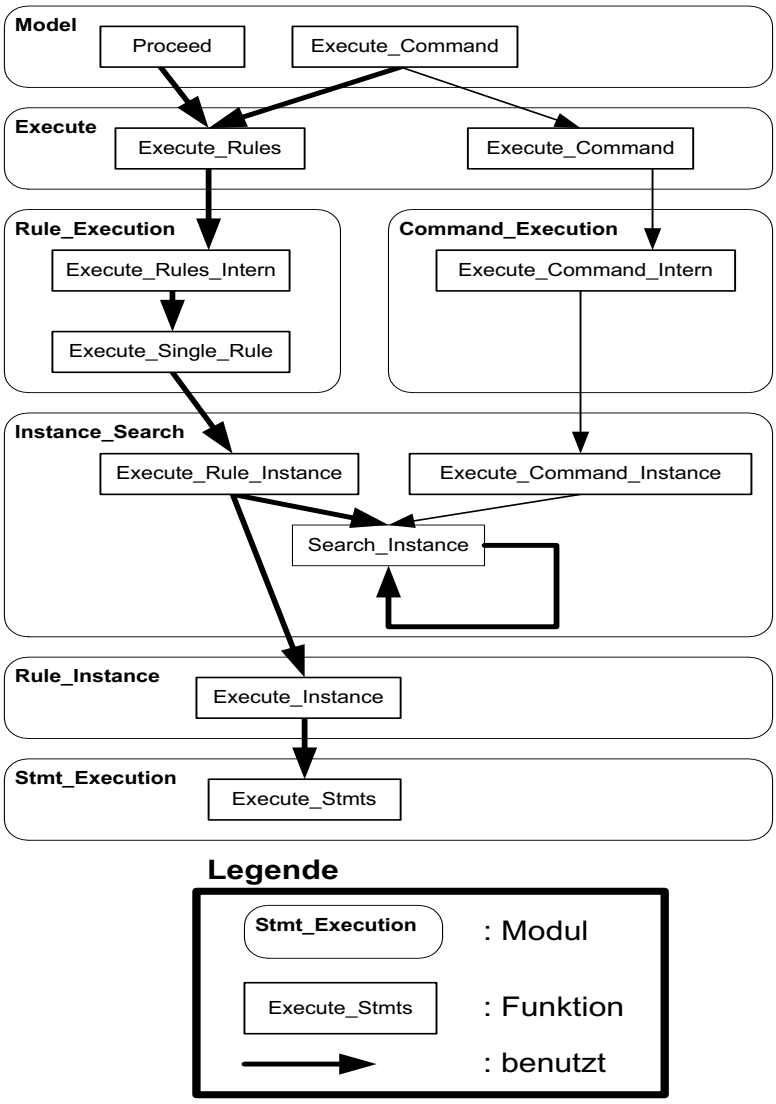



**Abbildung 7. Kritischer Pfad**

Die am “Kritischen Pfad” beteiligten Funktionen werden im Folgenden einzeln auf ihre Komplexität untersucht und die Ergebnisse danach zusammengefasst. Sie werden Top-Down in der Aufrufreihenfolge aufgeführt.

#### 3.2.2.1 Model.Proceed

Der Spieler gibt während der Simulation durch “proceeds” an, wieviele Zeitschritte der Simulator voranschreiten soll. Die Funktion Model.Proceed realisiert dieses Fortschreiten. Für jeden Schritt wird ein Regelauswertungszyklus ausgeführt, indem die Funktion Execute_Rules aufgerufen wird. Falls durch einen Regelauswertungszyklus mehr Zeit als ein Zeitschritt verbraucht wird, so wird nochmals ein Regelauswertungszyklus gestartet. Die auszugebenden Nachrichten werden gesammelt und an der Oberfläche an den Spieler ausgegeben.

| **Definition der Variablen:**<br><br>**t: Anzahl der Wiederholungen des Regelauswertungszyklus**<br>**s: Anzahl der Schritte um die weitergeschaltet werden soll** |
|---|
| **worst case:** $s9 = (s + t) \cdot$ Execute_Rules<br>Der "worst case" tritt nur ein, falls bei dem Regelauswertungszyklus mehr Modell-Zeit verbraucht wird als ein normaler Zeitschritt dauert. |
| **best (normal) case:** $s9 = s \cdot$ Execute_Rules |

#### 3.2.2.2 Model.Execute_Command

Diese Funktion dient der Ausführung eines Kommandos. Falls bei der Ausführung aller Kommandos so viel Zeit verbraucht wurde, dass ein Zeitschritt überschritten wird, so wird der Regelausführungszyklus gestartet. Dazu wird, wie bei Model.Proceed die Funktion Execute_Rules aufgerufen. Die auszugebenden Nachrichten werden gesammelt.

| **Definition der Variablen:**<br><br>**t: Anzahl der Wiederholungen des Regelauswertungszyklus** |
|---|
| **worst case:** $s8 = t \cdot$ Execute_Rules<br>Der "worst case" tritt nur ein, falls bei dem Regelauswertungszyklus mehr Modell-Zeit verbraucht wird, wie ein normaler Zeitschritt dauert. |
| **best (normal) case:** $s8 = 1 \cdot$ Execute_Rules |

#### 3.2.2.3 Execute.Execute_Rules

Diese Funktion reicht den Aufruf weiter an die interne Ausführung in Rule_Execution. Es werden interne Fehler abgefangen, sowie die auszugebenden Nachrichten weitergeleitet.

| **every case:** $s7 = 1 \cdot$ Execute_Rules_Intern |
|---|

#### 3.2.2.4 Rule_Execution.Execute_Rules_Intern

In dieser Funktion beginnt der Regelauswertungszyklus für einen Zeitschritt.

Wie bereits erwähnt, sind die Regeln in Prioritätsstufen eingeordnet. Diese Stufen werden von der höchsten bis zur niedersten durchlaufen und jeweils alle Regeln einer Stufe untersucht.

Es wird versucht, Instanzen von jeder Regel der aktuellen Stufe zu bilden und auszuführen. Dabei wird darauf geachtet, dass keine Regelinstanz während eines Zyklus doppelt ausgeführt wird. Da sich durch eine ausgeführte Regel der Zustand ändert, besteht die Möglichkeit, dass zusätzliche Instanzen von Regel gebildet werden können, die bereits untersucht wurden. Um die Reihenfolge der Prioritäten beizubehalten, werden nach einer ausgeführten Regel alle Regeln der aktuellen Prioritätsstufe nochmals untersucht, nicht jedoch die Regeln mit einer höheren Priorität. Erst wenn keine der Regeln einer Stufe mehr ausgeführt werden kann, wird zu der nächst tieferen Stufe gewechselt.

Der folgende Flussgraph zeigt nochmals den beschriebenen Ablauf.

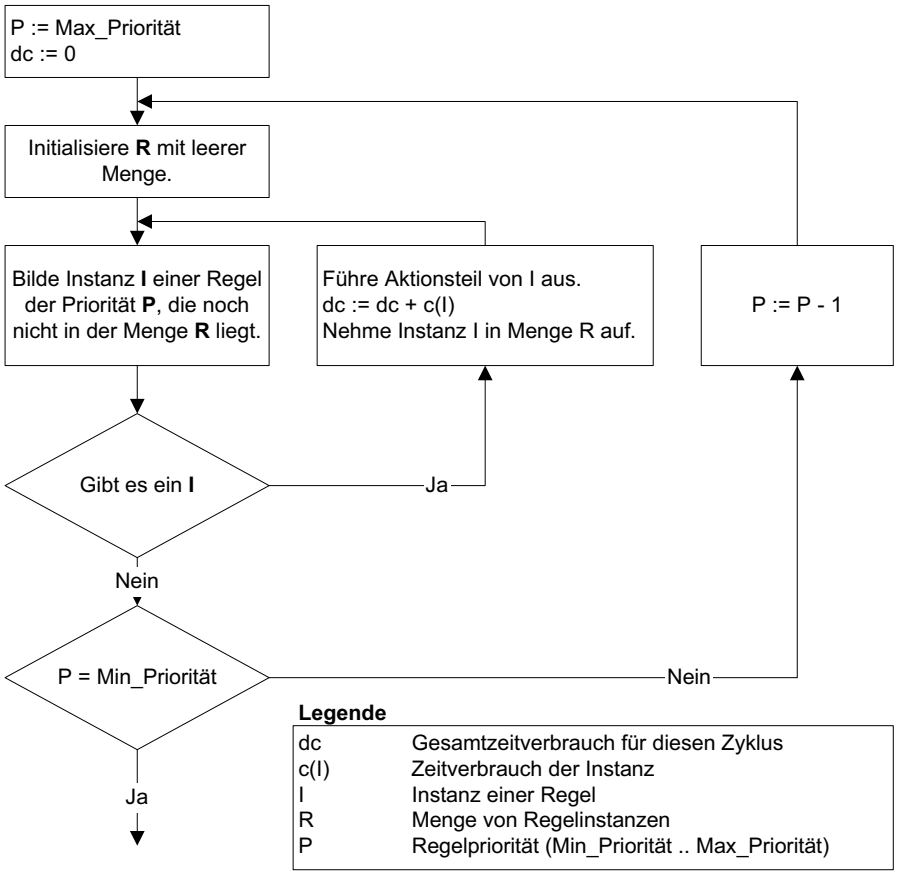



**Abbildung 8. Flussgraph des Regelauswertungszyklus**

Es wird vereinfachend angenommen, dass eine Regel, von der bereits Regelinstanzen gefunden wurden, in dem weiteren Verlauf des Regelauswertungszyklus nicht mehr betrachtet werden muss. Ohne diese vereinfachende Annahme müsste als Worst Case eine Endlosschleife angenommen werden, da es theoretisch möglich ist, durch die Ausführung einer Regel immer wieder eine neue Vorraussetzung zu schaffen, dass eine andere Regel dieser Prioritätsstufe ausführbar ist.

**Definition der Variablen:**

**x: Anzahl der Prioritätsstufen**
**y: Durchschnittliche Anzahl der Regeln pro Prioritätsstufe**

**worst case:** $s6(x,y) = x \cdot \sum_{i=1}^{y} i \cdot \text{Execute\_Single\_Rule} = \text{x} \cdot \frac{y \cdot (y+1)}{2} \cdot \text{Execute\_Single\_Rule}$

Der “worst case” kommt zustande, falls bei jeder Prioritätsstufe bei dem Durchlaufen aller Regeln dieser Stufe, nur eine Regel-Instanz gefunden werden kann. Es müssen dann alle Regeln dieser Stufe, bis auf die gefundene, nochmals betrachtet werden. Dies wird fortgesetzt, bis alle Regeln dieser Stufe einmal ausgeführt sind.

| |
|---|
| **average case:** $s6(x,y) = \frac{1}{2} \cdot x \cdot y \cdot \sum_{i=2}^{y-1} i \cdot \text{Execute\_Single\_Rule} = x \cdot \frac{y \cdot (y+3)}{4} \cdot \text{Execute\_Single\_Rule}$ |
| **best case:** $s6(x,y) = x \cdot y \cdot \text{Execute\_Single\_Rule}$<br>Der “best case” tritt ein, falls keine Regel-Instanz gefunden wird. Somit muss jede Regel nur einmal betrachtet werden. |

#### 3.2.2.5 Rule_Execution.Execute_Single_Rule

Es wird versucht, so lange Instanzen von einer Regel zu bilden, bis dies nicht mehr möglich ist. Es werden dabei alle möchlichen Kombinationen von Entitäten gebunden.

| |
|---|
| **Definition der Variablen:**<br><br>**q: Anzahl der Untersuchungen einer Regel während eines Regelauswertungs-zyklus. (q ≥ 1)** |
| **every case:** $s5(q) = q \cdot \text{Execute\_Rule\_Instance}$ |

#### 3.2.2.6 Instance_Search.Execute_Rule_Instance

Es werden zwei Funktionen verwendet. Durch Search_Instance wird versucht eine Instanz der Regel zu bilden. Diese wird dann durch die Funktion Execute_Instance ausgeführt.

Es wird der Zeitverbrauch der Regelausführung zurückgegeben.

| |
|---|
| **every case:** $s4 = 1 \cdot \text{Search\_Instance} + 1 \cdot \text{Execute\_Instance}$ |

#### 3.2.2.7 Instance_Search.Search_Instance

Die Funktion Search_Instance bekommt eine Regel als Parameter übergeben und liefert bei erfolgreicher Suche eine Regelinstanz zurück, sonst null. Die übergebene Regel kann bereits Entitäten gebunden haben, was bei Benutzerkommandos der Fall sein kann, welche in der Struktur einer Regel entsprechen. Die Instanz kann gebildet werden, wenn für alle formalen Entitäten und Relationen die benötigten Entitäten und Relationen im Zustand existieren und alle Attributsbedingungen erfüllt sind.

Jede Regel wird beim Laden eines Modells zu Beginn der Simulation durch den Compiler in eine interne Form übersetzt. Der Compiler erzeugt aus dem Bedingungsteil die “IC_Statements”. Diese werden durch den Compiler in einer Reihenfolge angeordnet, so dass die Suche möglichst früh abgebrochen werden kann, sollte eine benötigte Entität oder Relation fehlen oder eine Attributsbedinung nicht erfüllt sein. Zu Beginn der Suche wird die globale Liste der "IC_Statements" mit den "IC_Statements" der Regel gefüllt.

Bei der Suche nach einer Regel-Instanz werden alle “IC_Statements” dieser Liste nacheinander abgearbeit und es wird dabei zwischen drei Fällen unterschieden. Für jeden Typ eines “IC_Statement”[1] ein Fall:

- Für ein “Check_Stmt” wird geprüft, ob die Bedingung erfüllt ist.

1. Eine genauere Erklärung der “IC_Statements” erfolgt in Kapitel 2.

- Für ein “Bind_Ent_Stmt” wird nach allen Entitäten gesucht, die zu dem angegebenem Entitätstyp konform sind. Konform sind Entitätstypen dann, wenn diese voneinander abgeleitet wurden.
- Für ein “Bind_Ent_Over_Rel_Stmt” wird nach allen Entitäten gesucht, die zu dem angegebenem Entitätstyp konform sind und der Rolle entsprechen, die in der Relation angegeben ist.

Bei den “IC_Statements”, “Bind_Ent_Stmt” und “Bind_Ent_Over_Rel_Stmt” wird eine Liste mit Entitäten zurückgegeben, die nacheinander versuchsweise gebunden werden.

Wenn die Bedingung erfüllt ist oder eine Entität erfolgreich gebunden wurde, wird der Algorithmus erneut aufgerufen und das nächste Statement bearbeitet. War die Bedingung nicht erfüllt oder das Binden nicht erfolgreich, so wird zurückgesprungen auf das Statement davor und versucht, eine weitere Entität der bereits gesuchten zu binden. Durch die Rekursion wird somit implizit ein Suchbaum aufgebaut, der durchlaufen wird.

Wenn alle “IC_Statements” mit Erfolg ausgeführt sind, wird die Regelinstanz zurückgeliefert und diese in einer Liste gespeichert. Damit kann bei mehreren Suchen nach einer Regelinstanz entschieden werden, ob diese Regelinstanz schon ausgeführt wurde. Falls keine Entität eines Typs gebunden werden kann oder eine Attributsbedingung nicht erfüllt ist, wird null zurückgegeben.

Die Rekursion des Algorithmus lässt sich schlecht als Flussgraph darstellen. Das folgende Schaubild zeigt daher einen erzeugten Suchbaum. Es sind die IC_Statements zu sehen, die nacheinander behandelt werden. Der Bedinungsteil der dazugehörigen Regel würde drei formale Entitäten, eine formale Relation und eine Attributsbedingung enthalten. Für jedes der Bind_Ent_Stmts und Bind_Ent_Over_Rel_Stmts existieren je zwei passende Entitäten, die versuchsweise gebunden werden. Für jede gebundenen Entitäten müssen dann alle folgenden IC_Statements durchprobiert werden.

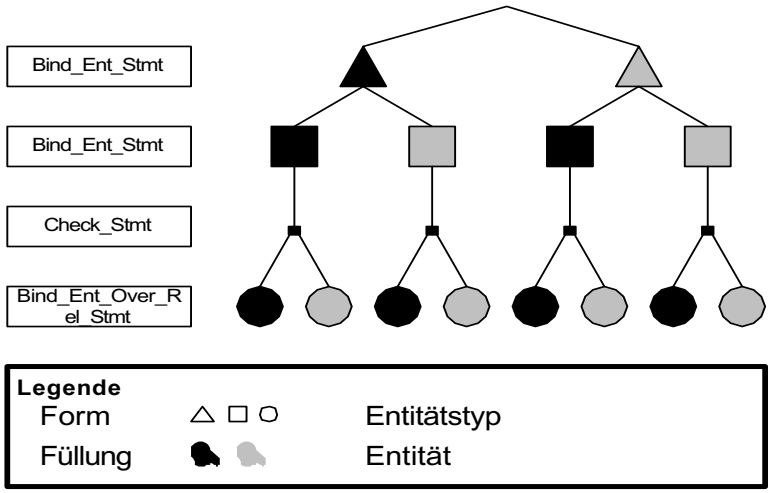



**Abbildung 9. Suchbaum für eine Regelinstanzsuche**

**Definition der Variablen:**

**m: Anzahl der Bind_Ent_Stmts + Anzahl der Bind_Ent_Over_Rel_Stmts (pro Regel)**
**n: Durchschnittliche Anzahl der Entitäten pro Entitätstyp**
**o: Anzahl der Check_Stmts pro Regel**
**p: Tiefe des Stmts in dem Suchbaum 1<=p<=m**

**worst case:** $s1(m,n,o,p) = \sum_{i=1}^{m} n^i + \sum_{i=1}^{o} n^{p-1} = \frac{n^{m+1}-1}{n-1} + \sum_{i=1}^{o} n^{p-1}$

Der “worst case” tritt ein, wenn für jedes Bind_Statement alle möglichen Entitäten versuchsweise gebunden werden müssen. Es stellt sich dabei jeweils erst bei dem letzten IC_Statement heraus, dass die Regel so nicht instanziierbar ist. Je nach Tiefe im Suchbaum muss also eine Entität “n^(p)”-mal versuchsweise gebunden werden. Somit muss für jede Entität eines bestimmten Typs das darauf folgende Check_Stmt “n^(p-1)” geprüft werden.

**average case:** $s1(m,n,o,p) = \frac{1}{2} \cdot \left( \sum_{i=1}^{m} n^i + \sum_{i=1}^{o} n^{p-1} + 1 \right) = \frac{1}{2} \cdot \left( \frac{n^{m+1}-1}{n-1} + \sum_{i=1}^{o} n^{p-1} + 1 \right)$

**normal case:** $s1(m,n,o,p) = m \cdot n + n \cdot o$

Der “normal case” bezieht Beobachtungen aus Regel-Auswertungszyklen mit realen Modellen mit ein. Der Fall, dass eine Entität mehrfach versuchsweise gebunden werden muss, tritt zum Beispiel auf, wenn mehrere Entwickler die selbe Aktivität an einem Dokument durchführen. Dies ist zwar während der Simulation kein seltener Fall, die Anzahl der Regeln, die dies betrifft ist allerdings sehr gering (circa 1/15 aller Regeln). Dieser Fall kann somit vernachlässigt werden.

**best case:** $s1(m,n,o,p) = 1$

Der “best case” tritt ein, falls nach dem ersten “IC_Statement“ festgestellt wird, dass die Regel nicht instanziierbar ist.

#### 3.2.2.8 Rule_Instance.Execute_Instance

Der Aktionsteil und die lokalen Variablen einer Regelinstanz oder der Instanz eines Benutzerkommandos wird ausgelesen und der Funktion Execute_Stmts zur Ausführung übergeben.

**every case:** $s2 = 1 \cdot \text{Execute\_Stmts}$

#### 3.2.2.9 Stmt_Execution.Execute_Stmts

Die Statements einer Regel oder eines Kommandos werden sequentiell abgearbeitet. Einzelne Statements werden durch die Prozedur Execute_Stmt behandelt.

| **Definition der Variablen:**<br><br>**z: #Execute_Stmts im Aktionsteil** |
|---|
| **every case:** $s3 = z$ |

### 3.2.3 Zusammenfassung

s9 *<br>oder<br>s8 *

s7 * s6 * s5 * s4 * ( s1 + s3 * s2 )

**Abbildung 10. Zusammenhang der Formeln**

Bei der Zusammenfassung der Formeln wird die Formel s8 ausgewählt, da sie die komplexere der beiden möglichen darstellt.

| **Definition der Variablen:**<br><br>**m: Anzahl der Bind_Ent_Stmts + Anzahl der Bind_Ent_Over_Rel_Stmts (pro Regel)**<br>**n: Durchschnittliche Anzahl der Entitäten (pro Entitäts-Typ)**<br>**o: Anzahl der Check_Stmts (pro Regel)**<br>**p: Tiefe des Stmts in dem Suchbaum**<br>**q: q: Anzahl der Untersuchungen einer Regel während eines Regelauswertungszyklus. (q ≥ 1)**<br>**t: Anzahl der Wiederholungen des Regelauswertungszyklus**<br>**s: Anzahl der Schritte um die weitergeschaltet werden soll**<br>**x: Anzahl der Prioritätsstufen**<br>**y: Durchschnittliche Anzahl der Regeln pro Prioritätsstufe**<br>**z: Anzahl der Execute_Stmts im Aktionsteil** |
|---|

#### 3.2.3.1 Unbewertete Zusammenfassung aller Regeln

| | |
|---|---|
| **worst case:** | $s(m, n, o, s, t, p, q, x, y, z) = s \cdot t \cdot \left( x \cdot \frac{y \cdot (y+1)}{2} \cdot \left( q \cdot \left( \left( \left( \frac{n^{m+1}-1}{n-1} + \sum_{i=1}^{o} n^{p-1} \right) + (z) \right) \right) \right) \right)$ |
| **normal case:** | $s(m, n, o, s, t, p, q, x, y, z) = s \cdot t \cdot \left( x \cdot y \cdot \left( \frac{y+3}{4} \right) \cdot (q \cdot ((m \cdot n + n \cdot o) + (z))) \right)$ |
| **average case:** | $s(m, n, o, s, t, p, q, x, y, z) = s \cdot t \cdot \left( x \cdot y \cdot \left( \frac{y+3}{4} \right) \cdot \left( q \cdot \left( \left( \left( \frac{1}{2} \cdot \left( \frac{n^{m+1}-1}{n-1} + \sum_{i=1}^{o} n^{p-1} + 1 \right) \right) + z \right) \right) \right) \right)$ |
| **best case:** | $s(m, n, o, s, t, p, q, x, y, z) = s \cdot t \cdot x \cdot y \cdot q \cdot z$ |

### 3.2.3.2 Bewertete Zusammenfassung aller Regeln

Im Folgenden werde weitere Vereinfachungen vorgenommen und begründet, um die Haupteinflussfaktoren zu ermitteln.

Es wird nur ein Zeitschritt und damit ein kompletter Regelauswertungszyklus betrachtet. Falls der Spieler mehrere Zeitschritte vorschalten möchte, so kann dies aus einem Zyklus hochgerechnet werden. Das Gleiche gilt auch für den Fall, dass während des Zykluses soviel Zeit verbraucht wird, dass sofort ein neuer Zyklus angestoßen wird. Die Variablen “s” und “t” können somit von der Betrachtung ausgeschlossen werden.

Nur von wenigen Regeln können mehrere Instanzen in einem Zeitschritt gebildet werden. Dies ist zum Beispiel der Fall bei Regeln, durch die Entwickler Veränderungen an einem Dokument vornehmen. Für jeden Entwickler, der dieser Aktivität zugeordnet ist, wird eine Instanz erzeugt. Da dies allerdings nur bei circa 1/15 aller Regeln der Fall ist, kann die Variable “q” somit vernachlässigt werden.

Bei der Erhebung der Kennzahlen von Modellen fällt auf, dass die Anzahl der “Check_Stmts” pro Regel mit einem Statement sehr gering ist. Dieser Einfluss kann somit vernachlässigt werden. Die Variablen “p“ und “o“ können aus der Betrachtung ausgeschlossen werden.

Der Einfluss der Ausführung des Aktionsteils einer Regel darf nicht unterschätzt werden, da dieser aufgrund von Schleifen und If-Statements recht komplex werden kann. Hier kann allerdings nur nach Anzahl der Aktivitäten bewertet werden, da die Einflussfaktoren zu groß sind und die Regeln zu unterschiedlich. Eine genauere Untersuchung muss während der Effizienzmessung stattfinden.

| | |
|---|---|
| **worst case:** | $s(m,n,x,y,z) = x \cdot \frac{y \cdot (y+1)}{2} \cdot \left( \left( \frac{n^{m+1}-1}{n-1} \right) + z \right)$ |
| **normal case:** | $s(m,n,o,x,y,z) = x \cdot y \cdot \left( \frac{y+3}{4} \right) \cdot (m \cdot n + z)$ |
| **average case:** | $s(m,n,x,y,z) = x \cdot y \cdot \left( \frac{y+3}{4} \right) \cdot \left( \frac{1}{2} \cdot \left( \frac{n^{m+1}-1}{n-1} + 1 \right) + z \right)$ |
| **best case:** | $s(m,n,x,y,z) = x \cdot y \cdot z$ |

### 3.2.3.3 Asymptotische Abschätzung

$O(x,y,z,m,n) = x \cdot y^2 \cdot (n^m + z)$

Entscheidend für die obere Schranke ist die durchschnittliche Anzahl der Entitäten pro Entitätstyp **n** und die durchschnittliche Anzahl an IC_Statements pro Regel **m**, durch die diese Entitäten an eine Regel gebunden werden.

Es zeigt sich außerdem, dass die Anzahl der Regeln (durchschnittliche Anzahl der Regeln pro Prioritätsstufe **y**) und ihre Verteilung auf die Prioritätsstufen **x** die Ausführdauer beeinflussen.

$\Omega(x,y) = x \cdot y$ : für x*y = Anzahl aller Regeln

Um eine möglichst aussagekräftige untere Schranke zu erhalten, wurde der Minimalfall betrachtet, für den versucht wird, jede Regel nur einmal zu instanziieren. Bei jeder Regel wird nach dem ersten “IC_Statement“ festgestellt, dass diese nicht instanziierbar ist. Somit wird jede Regel, während eines Zeitschritts, nur einmal betrachtet.

## 3.3 Effizienzmessung

In der theoretischen Analyse wurde der Algorithmus aufgrund des Codes auf seine Komplexität hin untersucht. Es wurden Vereinfachungen angenommen, um die abhängigen Variablen möglichst gering zu halten.

Die Effizienzmessung des Algorithmus hat mehrere Ziele:

- In erster Linie soll untersucht werden, an welchen Stellen des SESAM-Systems die meiste Zeit verloren geht. Ausgehend von der Hypothese, dass die größte Komplexität in der Regelsuche liegt, konzentriert sich die Untersuchung auf diesen Teil. Sie ist jedoch nicht darauf beschränkt, um auch diese Hypothese falsifizieren zu können.
- Das zweite Ziel dieser Untersuchung ist, die gefundenen Zusammenhänge aus der theoretischen Analyse zu verfeinern, wo diese aufgrund der Vereinfachungen und nicht miteinbezogener Faktoren zu ungenau ist. Zum Beispiel wurde der Aktionsteil aufgrund der Hypothese nicht näher untersucht.
- Die theoretische Analyse kann nur Ober- und Untergrenzen für die Ausführdauer festlegen. Mit dieser Messung soll das wirkliche Verhalten des Simulators für die unterschiedlichen Modelle gezeigt werden.

### 3.3.1 Vorgehen

Die Effizienzmessung baut auf der theoretischen Analyse des Algorithmus auf und verwendet darin gewonnenes Wissen:

- Welche Stellen des Algorithmus sind genauer zu betrachten?
  Die theoretische Analyse zeigt, dass die Suche nach Instanzen einer Regel den größten Einfluss auf die Laufzeit der Basismaschine hat.
- Welche Variablen beeinflussen die Ausführdauer?
  Sowohl die Anzahl der Regeln als auch ihre Verteilung auf die Prioritätsstufen beeinflussen die Laufzeit.
  Daneben spielt die Anzahl der Entitäten und die Anzahl der IC_Statements eine große Rolle.

Zu Anfang der Untersuchung werden Messparameter festgelegt. Alle beeinflussenden Faktoren werden als Parameter definiert und die zu messenden Zeiten festgelegt.

In einem zweiten Schritt werden diese Messparameter den Teilen des Codes zugeordnet, in denen diese erhoben werden können. Die Veränderungen werden anschließend vorgenommen und dokumentiert. Diese werden mit Hilfe von besonders formatierten Kommentarzeilen auskommentiert, so dass sich bei normaler Übersetzung nichts an dem System ändert. Das Format der Kommentarzeilen ist: “-- -J-”. Der Parameter des Makefiles zum Erstellen der Version für die Effizienzmessung ist “debug-effizienz”. Die veränderte Version der Basismaschine wird dem CVS-Repository überstellt und alle weiteren Veränderungen an dieser Version durchgeführt. Dieses Vorgehen dient dazu die Vergleichsmessungen nach den Verbesserungen an dem System zu vereinfachen.

Nach den Veränderungen an dem Code des SESAM-Systems werden Simulationsläufe mit drei verschiedenen Modellen durchgeführt. Im Einzelnen sind dies:

- Das QS-Modell mit einer Schrittweite von einem Tag und dem Referenzspielverlauf aus [Kalajzic, 2001].
- Die Feingranulare Variante des QS-Modell mit einer Schrittweite von zwei Stunden und dem Referenzspielverlauf aus [Hampp, 2001].
- Das Original QSVA-Modell mit einer Schrittweite von einem Tag und dem Referenzspielverlauf aus [Kalajzic, 2001].

Die Spielverläufe für die Modelle QS-Modell und QSVA-Modell werden mit und ohne eingeschaltetem Stateprotocol durchgeführt. Die Ausgabe des Protokolls verändert die Ausführdauer von Statements aus dem Aktionsteil von Regeln. Bei den Messungen werden nur die davon betroffenen Zeiten aufgeführt - die Zeit für den Regelauswertungszyklus und die Zeit für die Ausführung des Aktionsteils.

Zuletzt werden die erhobenen Daten ausgewertet. Zum einen werden die erhobenen absoluten Zeiten umgerechnet, um die Anteile der untersuchten Funktionen des Systems an der Gesamtlaufzeit zu ermitteln. Zum anderen wird versucht, unter Zuhilfenahme der gemessenen Kennzahlen, die Zusammenhänge der theoretischen Analyse zu bestätigen.

## 3.3.2 Messparameter

### 3.3.2.1 Kennzahlen

Die folgenden Kennzahlen werden während der Messungen erhoben:

- Anzahl der besuchten Knoten pro Regelinstanzsuche.
  Dies entspricht der Anzahl an Knoten in dem implizit erzeugten Suchbaum.
- Die Anzahl der IC_Statements pro Regel.
- Die Anzahl der Entitäten pro Entitätstyp.
- Die Anzahl der durchgeführten Regelinstanzsuchen pro Regelauswertungszyklus.
- Die Anzahl der ausgeführten Regeln pro Regelauswertungszyklus.
- Die Anzahl an Execute_Stmts pro Regel.
- Die Anzahl der Instanzen einer Regel während eines Regelauswertungszyklus.

### 3.3.2.2 Zeiten

Das folgende Schaubild soll die zu messenden Zeiten verdeutlichen. Der wirkliche zeitliche Ablauf während der Ausführung des Systems wird vereinfacht und nur die für die Messung relevanten Zeiten aufgeführt. Zuerst wird das Modell geladen und zwei Benutzerkommandos ausgeführt. In dem anschließenden Regelauswertungszyklus werden mehrere Instanzen von Regeln erfolglos gesucht und einige Regelinstanzen ausgeführt.

Die Ausgaben der Zeiten und der Kennzahlen erfolgen nicht während der Regelinstanzsuche und auch nicht während der Regelausführung, so dass sich zwar die Zeit für die Regelauswertungszyklen verlängert, die einzelnen gemessenen Zeiten werden jedoch nicht verfälscht und bleiben somit vergleichbar.

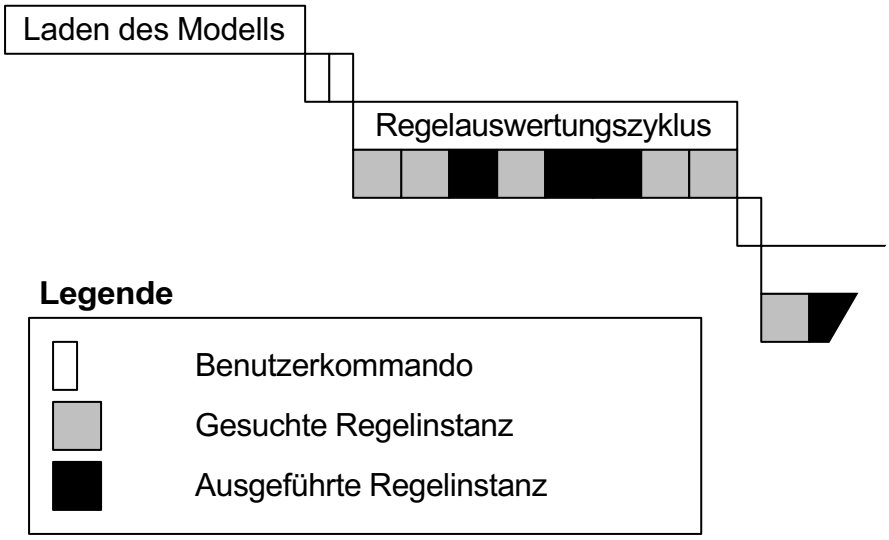



**Abbildung 11. Zeitlicher Ablauf**

Die folgenden Zeiten werden gemessen:

- Laden eines Modells.
- Ausführen eines Benutzerkommandos einschließlich der Suche nach der Kommandoinstanz.
- Ein kompletter Regelauswertungszyklus.
- Die Suche nach einer Regelinstanz.
- Die Ausführdauer des Aktionsteils einer Regel.

### 3.3.3 Messumgebung

Alle Messungen werden auf dem Abteilungsrechner “genf” der Abteilung Software-Engineering durchgeführt.

| | |
|---|---|
| Rechnersystem: | SUN Workstation Ultra-4 |
| Betriebssystem: | Solaris 2.7 |
| Entwicklungsumgebung: | Gnat 3.13p<br>ohne Compileroptimierung |

### 3.3.4 Ergebnisse

Im Folgenden werden die Ergebnisse dargestellt und erläutert. Die Zeiten werden nacheinander, wie in der Abbildung “Zeitlicher Ablauf” gezeigt, aufgeführt. Es beginnt mit dem Laden der Modelle, geht dann weiter mit der Gesamtzeit aller Regelauswertungszyklen und wie diese sich auf die Regelsuche und die Regelausführung verteilt. Zum Schluss werden noch die Benutzerkommandos behandelt.

Beim Laden der Modelle zeigt sich deutlich ein Zusammenhang zwischen der Größe der Modelle und der Zeit, die die Basismaschine benötigt, um diese zu laden.

| Modell | Zeit in ms | Größe in KB |
|---|---|---|
| **QS** | 47730 | 2.458 |
| **QSVA** | 60120 | 2.970 |
| **Feingranular** | 26740 | 1.158 |

**Tabelle 2. Laden der Modelle**

Die folgenden Zeiten für alle Regelauswertungszyklen werden durch die Ausgabe von Messdaten nach jeder Regelsuche und Regelausführung verfälscht, sie bleiben jedoch untereinander vergleichbar. Zum einen sieht man eine deutliche Steigerung der Ausführdauer bei eingeschaltetem Stateprotocol. Andererseits betrifft die Dauer der einzelnen Regelauswertungszyklen auch direkt den Spieler. Lässt dieser den Simulator mehrere Schritte voranschreiten, was recht häufig vorkommt, so kann die Wartezeit leicht im Minutenbereich liegen.

| Modell | State-protocol | Anzahl | Zeit in s | | | |
|---|---|---|---|---|---|---|
| | | | Min | Avg | Max | Gesamt |
| **QS** | Ja | 300 | 2,6 | 6,3 | 22,6 | 1904,3 s = 31,7 min |
| | Nein | 300 | 2,4 | 4,8 | 11,8 | 1453,9 s = 24,2 min |
| **QSVA** | Ja | 307 | 3,2 | 7,9 | 25,0 | 2417,5 s = 40,3 min |
| | Nein | 307 | 2,9 | 6,4 | 14,4 | 1961,6 s = 32,7 min |
| **Feingranular** | Nein | 1179 | 1,2 | 2,2 | 8,3 | 2580,8 s = 43,0 min |

**Tabelle 3. Zeit der Regelauswertungszyklen**

Die Gesamtzeit der Regelauswertungszyklen teilt sich auf in die Suche nach den Regelinstanzen und die Ausführung der gefundenen Regelinstanzen. Diese Zeiten werden nur minimal beeinflusst durch Zuweisungen an globale Variablen, die der Messwerterfassung dienen. Die folgende Tabelle zeigt die Gesamtzahl der gesuchten Regeln und die gesuchten Regeln je Zyklus.

| Modell | Gesuchte Regeln je Zyklus | | | Gesamt |
|---|---|---|---|---|
| | Min | Avg | Max | |
| **QS** | 692 | 969 | 1513 | 290735 |
| **QSVA** | 877 | 1407 | 2158 | 432027 |
| **Feingranular** | 559 | 590 | 1077 | 696701 |

**Tabelle 4. Anzahl der gesuchten Regeln**

Bei den Zeiten für die einzelnen Suchen nach Regelinstanzen zeigt sich, dass eine Suche sehr wenig Zeit verbraucht. Die Zeiten, die mit 0 angegeben sind, lagen unter der Messgrenze.

| Modell | Zeit in ms | | | |
|---|---|---|---|---|
| | Min | Avg | Max | Gesamt |
| **QS** | 0 | 0 | 20 | 153160 |
| **QSVA** | 0 | 0 | 70 | 213840 |
| **Feingranular** | 0 | 0 | 90 | 318700 |

**Tabelle 5. Zeit für die Regelinstanzsuche**

Die Anzahl der ausgeführten Regeln fällt um den Faktor 20 geringer aus als die gesuchten Regeln.

| Modell | Ausgeführte Regeln je Zyklus | | | | Gesamt |
|---|---|---|---|---|---|
| | Min | Median | Avg | Max | |
| **QS** | 27 | 43 | 44 | 81 | 13107 |
| **QSVA** | 32 | 62 | 63 | 111 | 19425 |
| **Feingranular** | 36 | 39 | 39 | 65 | 46504 |

**Tabelle 6. Anzahl der ausgeführten Regeln**

Die Zeiten für die Ausführung der Regeln liegen jedoch um Faktoren von zwei bis fünf über der Suche.

| Modell | State-protocol | Zeit in ms | | | |
|---|---|---|---|---|---|
| | | Min | Avg | Max | Gesamt |
| **QS** | Ja | 0 | 66 | 8930 | 868760 |
| | Nein | 0 | 55 | 4961 | 456000 |
| **QSVA** | Ja | 0 | 46 | 6960 | 899100 |
| | Nein | 0 | 34 | 3944 | 469470 |
| **Feingranular** | Nein | 0 | 4 | 4060 | 191070 |

**Tabelle 7. Zeit für die Regelausführung**

Der Vollständigkeit halber werden die Anzahl und die Ausführdauer der Benutzerkommandos mit aufgeführt. Sie spielen mit einer Gesamtdauer von maximal sieben Sekunden während einer Simulation keine Rolle für die Effizienz. Die gemessene Zeit schließt sowohl die Suche nach Instanzen der Kommandos, als auch die Ausführung der Kommandos ein.

| Modell | Anzahl | Zeit in ms | | | |
|---|---|---|---|---|---|
| | | Min | Avg | Max | Gesamt |
| **QS** | 83 | 0 | 76 | 940 | 6210 |
| **QSVA** | 83 | 0 | 89 | 940 | 7360 |
| **Feingranular** | 41 | 0 | 5 | 20 | 160 |

**Tabelle 8. Zeit und Anzahl der ausgeführten Kommandos**

#### 3.3.4.1 Auswertung

Entgegen der Grundannahme aus Kapitel 3.1.1 wird bei den Modellen mit einer Schrittweite von einem Tag mehr Zeit für die Ausführung der Regeln verbraucht als für deren Suche. Bei ähnlichen Messungen von Mitarbeitern der Abteilung Software Engineering mit älteren Modellen hat sich genau das Gegenteil ergeben. Durch die Weiterentwicklung der Modelle, das Einführen von neuen Konzepten und die bereits durchgeführten Performanceverbesserungen hat sich der Zeitanteil deutlich verschoben.

Mit eingeschaltetem Stateprotokoll übersteigt die Zeit für das Ausführen beim QSVA-Modell die Suche um den Faktor 4, beim QS-Modell sogar um 5. Ohne Stateprotokoll beträgt dieser Faktor immer noch 2 beim QSVA-Modell und 3 bei dem QS-Modell.

Bei dem feingranularen Modell übersteigt dagegen die Zeit für die Suche die der Ausführung um den Faktor 1,5. Dies hat mehrere Gründe:

- Die Schrittweite beträgt anstatt einem Tag nur noch zwei Stunden. Da der Aktionsteil abhängig von der verfügbaren Zeit in einem Zeitschritt ist, können weniger Aktionen pro Regel ausgeführt werden. Zum Beispiel kann ein Entwickler in zwei Stunden weniger Anforderungen in ein Dokument übertragen als in den acht Stunden, die er bei einer Schrittweite von einem Tag gearbeitet hat. Es werden aber insgesamt mehr Regeln ausgeführt.
- Die feingranulare Variante enthält mehr Regeln auf einer Prioritätsstufe. Dadurch werden Regeln dieser Prioritätsstufe häufiger doppelt betrachtet als sonst.
- Der Regelauswertungszyklus wird 12-mal am Tag ausgeführt und somit 12-mal so oft Regelinstanzen gesucht.

### 3.3.5 Vergleich mit der theoretischen Analyse

Im Folgenden wird versucht, durch weitere Untersuchungen an dem Modell QSVA die in der theoretischen Analyse aufgestellten Zusammenhänge zu zeigen.

#### 3.3.5.1 Suche von Regel-Instanzen

Die theoretische Analyse zeigt, dass die Anzahl der besuchten Knoten während der Suche nach einer Regelinstanz stark von der Anzahl der Entitäten pro Entitätstyp und von der Anzahl der IC_Statements abhängt.

Das folgende Schaubild stellt die gemessene Anzahl der besuchten Knoten während einer Regelinstanzsuche der mit dem “normal case” berechneten Anzahl der Knoten gegenüber. Es wird nach der berechneten Anzahl der Knoten sortiert.

Beim “normal case“ berechnet sich die Anzahl der Knoten nach der Formel:

Durchschnittliche Anzahl von Entitäten pro Entitätstyp · (Anzahl der Bind_Stmts + Anzahl der Check_Stmts)

Es wird hier nur das Modell QSVA betrachtet, da Untersuchungen an den anderen Modellen ähnliche Ergebnisse ergeben.

Es zeigt sich, dass die wirkliche Anzahl der Knoten sich um den “normal case” herum bewegt. Die Annahme die zu dem “normal case” geführt hat, es komme so gut wie nie vor, dass eine Entität eines bestimmten Typs mehrfach versuchsweise gebunden werden muss, kann somit als richtig angesehen werden kann.

Die Auswahl der Regeln erfolgt zufällig aus der Menge der gesuchten Regeln während eines Simulationslaufes.

Die Ausnahmen, die eine extrem erhöhte Anzahl an besuchten Knoten aufweisen, sind Regeln, die während der Initialisierung des Modells ausgeführt werden. Beispiele dafür sind die Regeln:

- NOTATION_WECHSELN,
- NOTATION_FUER_ANALYSENOTIZEN_VOREINSTELLEN,
- NOTATION_FUER_HANDBUCH_VOREINSTELLEN,
- NOTATION_FUER_MODULSPEZ_VOREINSTELLEN.

Bei diesen Regeln existieren sehr viele Entitäten, die für eine formale Entität gebunden werden können. Dadurch müssen, wie in der theoretischen Analyse beschrieben, viele Entitäten mehrfach versuchsweise gebunden werden.

Die Ausnahmen, die eine extrem geringe Anzahl an besuchten Knoten aufweisen, sind Regeln die mehrere Relationen enthalten, wobei an jeder Relation verhältnissmäßig viele Rollen beteiligt sind. Beispiele dafür sind die Regeln:

- ABGELEITETE_ATTRIBUTE_DES_CODES_BERECHNEN,
- ABGELEITETE_ATTRIBUTE_DER_MODULSPEZ_BERECHNEN.

Je mehr Rollen an einer Relation beteiligt sind, je mehr Entitäten können bei der Regelinstanzsuche ausgeschlossen werden.

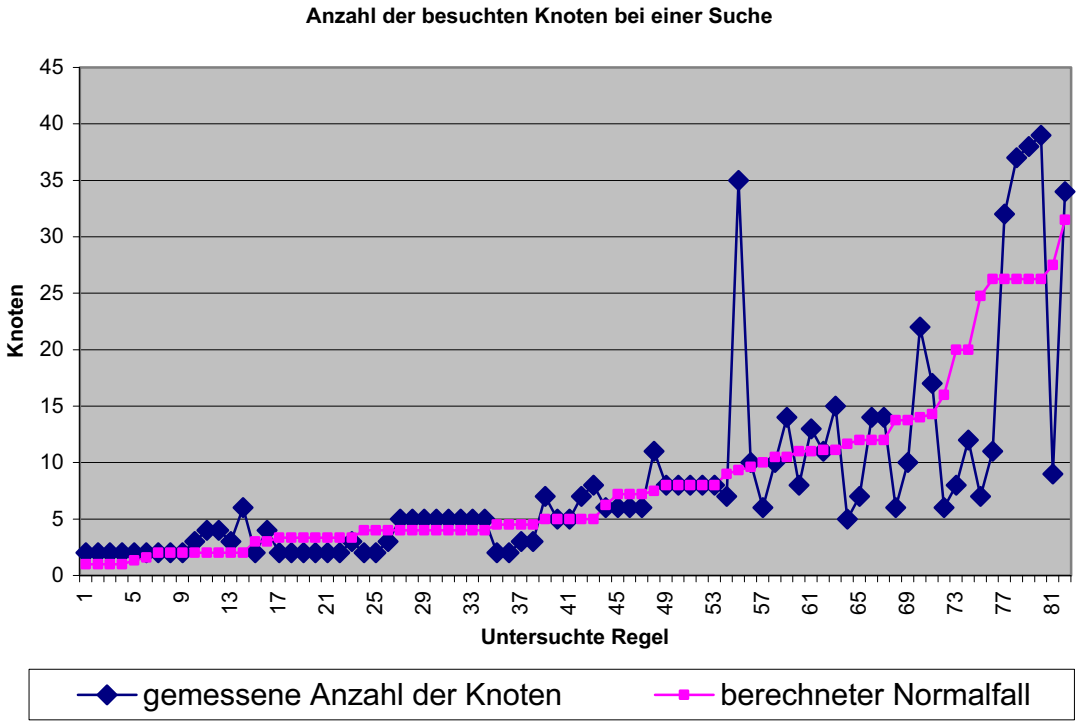



**Abbildung 12. Anzahl der Knoten bei einer Regelinstanzsuche**

### 3.3.5.2 Regelauswertungszyklus

Die folgenden Schaubilder zeigen die Anzahl der untersuchten Regeln während eines Regelauswertungszukluses. Die Messung wird mit den Fällen aus der theoretischen Analyse verglichen. Es wird der aus den Kennzahlen der Modelle berechneten Wert für die durchschnittliche Anzahl der Regeln pro Prioritätsstufe verwendet.

Die Schaubilder zeigen, dass sich die Werte zwar zwischen dem schlechtesten und dem besten Fall bewegen, dass der durchschnittliche Fall jedoch sehr stark vom Modell abhängig ist und nicht generell Gültigkeit besitzt. Die Anzahl hängt sehr stark davon ab, wieviele Regeln während eines Regelauswertungszyklus ausgeführt werden. Ausgeführte Regeln auf einer Prioritätsstufe führen dazu, dass weitere Regeln derselben Stufe erneut geprüft werden müssen.

Da bei der Feingranularen Variante des QS-Modells mehr Regeln auf einer Stufe existieren als bei den Modellen QS und QSVA, werden im Verhältnis mehr Regeln während eines Regelauswertungszyklus untersucht.

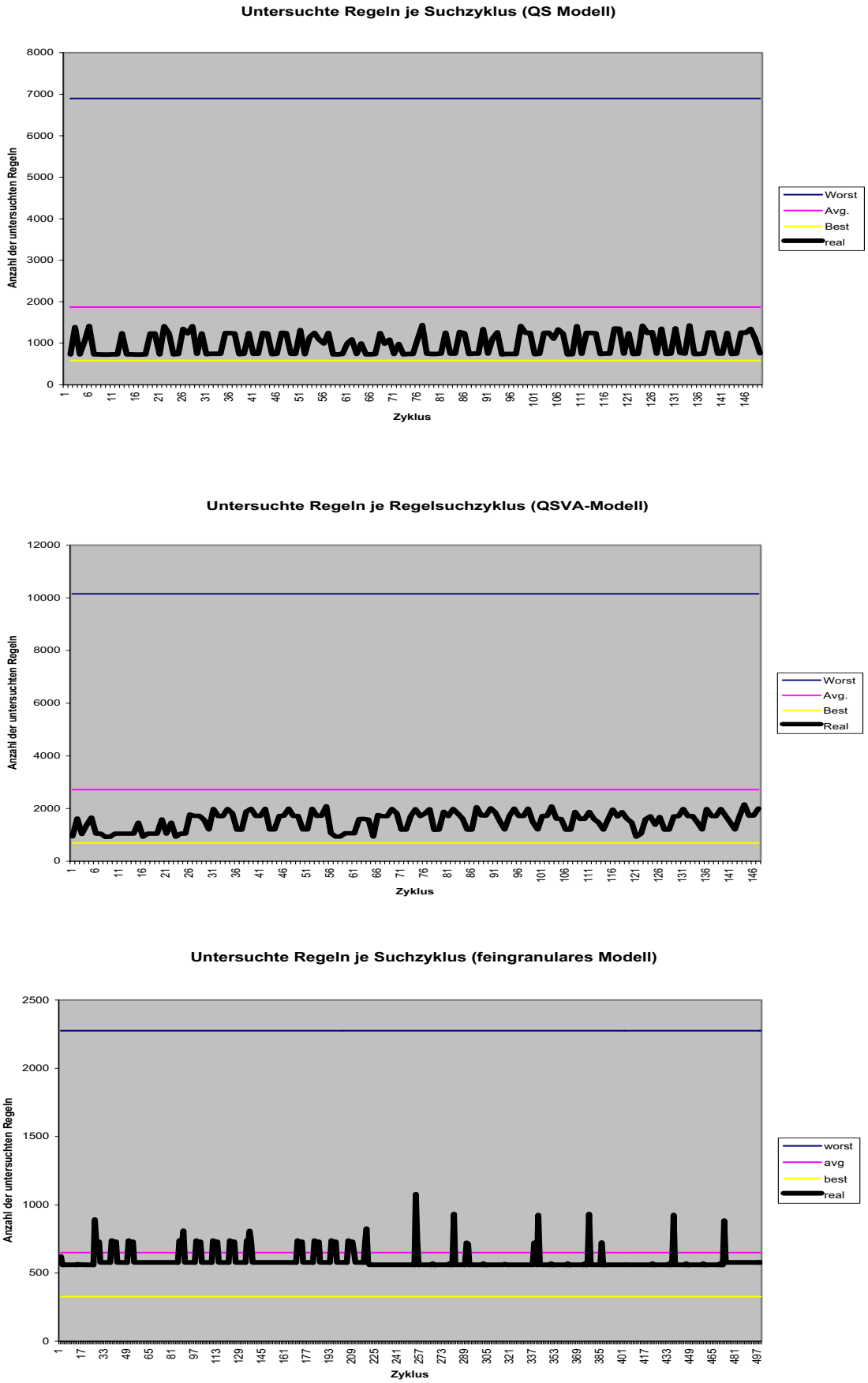



**Abbildung 13. Anzahl der untersuchten Regeln pro Regelauwertungzyklus je Modell**

## 3.4 Schwachstellen der Regelausführung

### 3.4.1 Regelinstanzsuche

Es hat sich gezeigt, dass die Zeit für die Suche nach einer einzelnen Regelinstanz sehr gering ist. Da aber die Anzahl der gesuchten Regelinstanzen die Zahl der ausgeführten Regelinstanzen um ein vielfaches übersteigt, ist und bleibt die Regelinstanzsuche ein bestimmender Faktor. Besonders, da sich die Weiterentwicklung der Modelle hauptsächlich auf die Anzahl der Regeln auswirkt. Die folgende Tabelle zeigt die Anzahl der nicht ausgeführten Regeln während eines Simulationslaufes im Verhältnis zu den ausgeführten Regeln.

| Modell | Anzahl der gesuchten Regelinstanzen | | nicht ausgeführte : ausgeführte |
|---|---|---|---|
| | nicht ausgeführte | ausgeführte | |
| **QS** | 277628 | 13107 | 21 : 1 |
| **QSVA** | 412602 | 19425 | 21 : 1 |
| **Feingranular** | 650197 | 46504 | 14 : 1 |

**Tabelle 9. Vergleich der gesuchten Regelinstanzen**

Da selbst im “average case” die Anzahl der Regeln einen quadratischen Einfluss auf die Laufzeit ausübt, werden zukünftige Erweiterungen der Modelle einen negativen Einfluss auf die Effizienz haben.

Eine Schwachstelle ist damit sicherlich die Summe der Regeln und die Anzahl der Suchen nach ihnen.

### 3.4.2 Regelinstanzausführung

Die folgende Tabelle setzt die Gesamtzeiten für die Suche nach Regelinstanzen in das Verhältnis zu den Ausführungszeiten der gefundenen Regeln, jeweils mit und ohne eingeschaltetem Stateprotokoll.

| Modell | State-protocol | Regelinstanzsuche gesamt in min | Regelausführung gesamt in min | Verhältnis |
|---|---|---|---|---|
| **QS** | Ja | 2,6 | 14,5 | 1 : 5,5 |
| | Nein | 2,6 | 7,6 | 1 : 3 |
| **QSVA** | Ja | 3,6 | 15,0 | 1 : 4 |
| | Nein | 3,6 | 7,8 | 1 : 2 |
| **Feingranular** | Nein | 5,3 | 3,2 | 1,5 : 1 |

**Tabelle 10. Vergleich der Zeiten für Suche und Ausführung von Regelinstanzen**

Die Regelausführung der Basismaschine wurde bei weiteren Simulationsläufen einer genaueren Untersuchung unterzogen. Es wird die Ausführung der einzelnen Statements sowie der Expressions betrachtet. Aus den Daten wird die durchschnittliche Dauer eines Statements berechnet und der prozentuale Anteil an allen ausgeführten Statements. Die Messungen werden mit eingeschaltetem Stateprotokoll durchgeführt.

Weiter werden die Ausführzeiten von einzelnen Regeln untersucht.

#### 3.4.2.1 Basismaschine

Bei der Messung der Zeiten für die einzelnen Statements und Expressions fällt auf, dass bei eingeschaltetem Stateprotocol manche Statements im Vergleich bis zu 30 mal so viel Zeit benötigen wie andere Statements. Es handelt sich dabei um Statements, die Entitäten und Relationen erzeugen oder löschen und um Expressions, die Veränderungen an Entitätsattributen vornehmen. Bei jeder Ausführung wird dabei ein Eintrag an die Stateprotocoldatei vorgenommen. Besonders fällt dabei die extrem erhöhte Dauer des Statements zur Erzeugung von Entitäten auf, da hierbei alle Attribute der Entität mit ausgegeben werden müssen.

Allgemein betrachtet kann auch sicherlich die Interpretation der Aktionsteile von Regeln und Kommandos als Schwachstelle der Basismaschine angesehen werden. Bei jeder Ausführung müssen die Ausdrücke zerlegt, Informationen aus der Symboltabelle ausgelesen und so Statement für Statement ausgeführt werden.

#### 3.4.2.2 Modell

Um die bei den Messungen aufgetretenen großen Schwankungen bei den Ausführzeiten von Regeln zu erklären, wurden die Ausführzeiten der einzelnen Regeln gemessen. Dabei zeigten sich zum Teil Ausführzeiten von mehreren Sekunden pro Regel.

Zusätzlich zu der Ausführdauer einer Regel wurde die Anzahl der ausgeführten Statements während der Ausführung gemessen. Besonders aufgefallen sind dabei die Regeln, die sich mit der Korrektur von allen Dokumenten nach einem Review oder Test beschäftigen. Hier wurden Zeiten von bis zu sieben Sekunden gemessen, bei einer Anzahl der ausgeführten Statements von 26629. Ein Beispiel für solch eine Regel ist:

- ACTIVE_ENTWICKLER_KORRIGIERT_ALLES_NACH_HANDBUCHREVIEW.

Weitere Regeln beschäftigen sich mit dem Bearbeiten von Dokumenten, dem Review und dem schnüren von Arbeitspaketen:

- ACTIVE_ENTWICKLER_ANALYSIERT,
- AKTIVE_SYSTEMENTWURF_REVIEWEN,
- MODULSPEZREVIEWAP_SCHNUEREN,
- ALLEKORREKTURAP_SCHNUEREN_NACH_SPEZREVIEW.

Zum Teil liegen die sehr hohen Ausführdauern an einer ineffizient modellseitig implementierten Suche nach Elementen in einem Datentyp “Bag”, der dazu verwendet wird, um die Anforderungen von Dokumenten zu speichern.

Als Beispiel sei hier die Funktion “Gib_Anforderung_zu_ID” aufgeführt. Diese wird von Aktionsteilen der hier aufgeführten Regeln sehr häufig aufgerufen.

In der Funktion wird so lange jedes Element in dem Bag untersucht, bis ein Element mit der übergebenen ID gefunden wird. Dies bedeutet, dass für **n** Anforderungen, wobei jede Anforderung ein Element des Bags ist, im durchschnittlichen Fall die Schleife alleine schon **n/2** mal ausgeführt wird. Umgerechnet auf die einzelnen Statements und Expressions sind das n/2 * (Assignment + 2 * Boolean_Literal + Function_Call) = ca. **2n** Statements und Expressions.

# Kapitel 4
# Verbesserungsvorschläge

*In diesem Kapitel werden Verbesserungsvorschläge vorgestellt und bewertet. Dazu wird jeder Vorschlag genau beschrieben, ebenfalls werden die Auswirkungen auf das Verhalten der Basismaschine diskutiert.*

## 4.1 Überblick

Dieses Kapitel enthält Vorschläge, durch die versucht werden soll, die gefundenen Schwachstellen der Basismaschine und der Modelle zu mindern oder zu beheben. Die Vorschläge wurden ohne Beachtung der zeitlichen Machbarkeit entwickelt und erst danach bewertet, um auch als Anregung für spätere Veränderungen zu dienen.

Die Vorschläge teilen sich in die drei Bereiche Regelinstanzsuche, Regelausführung und Modelle auf. Für jeden dieser Bereiche gibt es mehrere Vorschläge, die textuell beschrieben werden. Zur besseren Unterscheidung erhält jeder Vorschlag einen “sprechenden” Namen, der eine Haupteigenschaft des Vorschlags enthält. Zusätzlich findet sich bei jedem Vorschlag eine kurze Zusammenfassung, die die Kernidee beschreibt.

## 4.2 Bewertungskriterien

Nach Ausarbeitung der Vorschläge werden diese nach den folgenden Kriterien bewertet:

- Generelle Umsetzbarkeit:
  Es wird betrachtet, ob der Vorschlag, mit den zur Verfügung stehenden Mitteln, technisch machbar ist. Es wird hier nicht die zeitliche Machbarkeit untersucht.
- Aufwand und Komplexität der Veränderungen:
  Der Aufwand und die Komplexität werden in Schwierigkeitsstufen zusammengefasst. Es existieren drei Stufen, die im Folgenden beschrieben sind:

| **Schwierigkeitsstufe** | **Beschreibung** |
|---|---|
| Einfach: | • Der Umfang der Veränderungen an der Basismaschine ist als gering einzustufen.<br>• Die verwendeten Algorithmen sind allgemein bekannt und können ohne Anpassung für das Problem eingesetzt werden. |
| Mittel: | • Der Umfang der Veränderungen an der Basismaschine sind als mittel einzustufen.<br>• Die verwendeten Algorithmen sind bekannt, müssen jedoch für den Einsatzzweck angepasst werden. |
| Schwer: | • Der Umfang der Veränderungen an der Basismaschine sind als groß einzustufen.<br>• Die verwendeten Algorithmen sind nicht bekannt und müssen neu entwickelt werden. |

**Tabelle 11. Beschreibung der Schwierigkeitsstufen**

- Geschätzte Verbesserung der Laufzeit:
  Der Geschwindigkeitsvorteil wird ebenfalls in drei Stufen eingeteilt:
  - Groß
  - Mittel
  - Klein
- Auswirkungen auf die Basismaschine:
  Es muss möglichst genau beschrieben werden, welche Auswirkungen auf die Basismaschine zu erwarten sind. Insbesondere müssen die Anforderungen an die Basismaschine weiter erfüllt werden.

## 4.3 Regelinstanzsuche

In diesem Abschnitt werden Vorschläge beschrieben, die sich auf die Suche nach Regelinstanzen beziehen.

### 4.3.1 Vorschlag - “Zwei Caches”

#### 4.3.1.1 Kernidee

Zu jedem Zeitpunkt ist bekannt, welche Regelinstanzen existieren. Diese können, nach einer Prüfung weiterer Eigenschaften, während des Regelauswertungszyklus ausgeführt werden.

#### 4.3.1.2 Beschreibung des Vorschlags

Die Schwachstellenanalyse hat gezeigt, dass die Zahl der gesuchten Regelinstanzen einen großen Einfluss auf die Laufzeit der Basismaschine hat. Nur eine sehr geringe Anzahl der untersuchten Regeln kann dabei als Regelinstanzen ausgeführt werden.

Die Idee des Vorschlags ist, dass sich nur bei bestimmten Veränderungen am Zustand des Simulators die Menge der ausführbaren Regeln verändert. Somit erscheint es sinnvoll bei jeder dieser Veränderungen zu untersuchen, ob eine davon betroffenen Regel ausführbar ist oder nicht.

Nach dem Laden eines Modells enthält die Symboltabelle der Basismaschine Informationen über die existierenden Entitäts- und Relationstypen, sowie aller Regeln und Kommandos. In dem Strukturteil jeder Regel und jedes Kommandos sind Informationen enthalten, was für Entitäten und Relationen benötigt werden, um eine Instanz zu bilden. Es werden also Verbindungen zwischen den Entitäts- und Relationstypen und den betroffenen Regeln und Kommandos benötigt, so dass beim Erzeugen und Löschen von Entitäten und Relationen alle betroffenen Regeln und Kommandos betrachtet werden können.

Instanzen werden in Caches gespeichert. Für Regelinstanzen existieren zwei verschiedene Caches, wobei der erste “Regelcache 1” teilweise gebundene und der zweite “Regelcache 2” nur vollständig gebundene Regelinstanzen enthält. Im “Regelcache 2” befinden sich somit Regelinstanzen, die ausgeführt werden können.

In dem “Regelcache 1” wird je eine Kopie einer Regelinstanz für jede mögliche Kombination an gebundenen Entitäten und Relationen gespeichert. Erfordert eine Regel zum Beispiel eine Entität vom Typ “Entwickler” und existieren zwei Entwickler, so existiert für jeden Entwickler eine Regelinstanz in “Regelcache 1”. Hat eine dieser Regelinstanzen alle benötigten Entitäten und Relationen gebunden, wird sie in den “Regelcache 2” kopiert. Der zweite Cache wird nach der Priorität der Regel sortiert, um ein deterministisches Verhalten zu erreichen. Die Sortierung nach Namen, wie bei der

Regeltabelle der Symboltable, innerhalb einer Priorität entfällt, da hierdurch keine Performanzverbesserungen zu erwarten sind. Eine in den zweiten Cache kopierte Regelinstanz wird jeweils an das Ende der Regelinstanzen mit gleicher Priorität eingefügt, um den Regelauswertungszyklus zu vereinfachen.

“Regelcache 1” kann als Array implementiert werden, wobei jede Regel einem Index zugewiesen wird. Jeder Index enthält eine Liste aller Instanzen der Regel.

“Regelcache 2” kann ebenfalls als Array implementiert werden, mit einem Index pro Prioritätsstufe und ist nach Prioritätsstufen sortiert. Jeder Index dieses Arrays enthält eine Liste aller Regelinstanzen der Prioritätsstufe. Das folgende Schaubild zeigt den Aufbau der beiden Caches und stellt die Behandlung von Regelinstanzen dar.

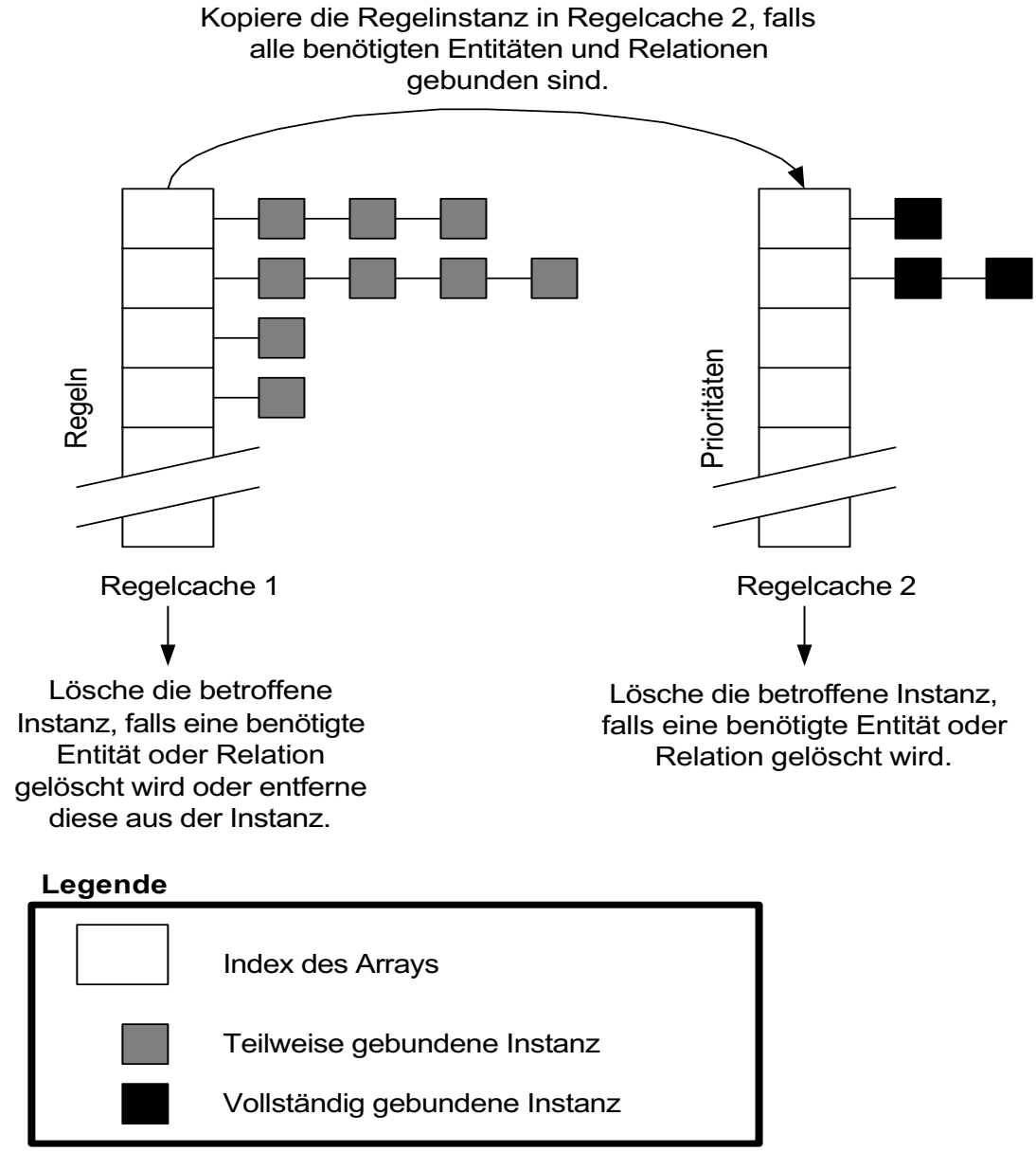



**Abbildung 14. Caches für Regelinstanzen**

Für Kommandoinstanzen werden noch weitere Entitäten benötigt, die erst dann übergeben werden, wenn der Spieler ein Kommando eingibt. Da somit in den meisten Fällen keine vollständig gebundenen Kommandoinstanzen existieren, reicht ein Cache für die Kommandoinstanzen aus. Dieser gleicht dem “Regelcache 1”, wobei jedem Kommando ein Index zugewiesen wird.

In jedem Regelauswertungszyklus werden die Regelinstanzen nacheinander aus dem “Regelcache 2” ausgelesen. Falls alle Attributsbedingungen erfüllt sind, wird die aktuelle Regelinstanz ausgeführt. Da sich durch die Ausführung einer Regel die Menge der ausführbaren Regeln verändern kann, werden diese direkt aus dem “Regelcache 2” ausgelesen, der automatisch immer auf dem neuesten Stand ist.

Zur Ausführung eines Kommandos werden die übergebenen Attribute an alle im Cache existierenden Instanzen des Kommandos gebunden und diese versucht auszuführen.

## 4.3.2 Vorschlag - “Zähler”

### 4.3.2.1 Kernidee

Zu jedem Zeitpunkt ist bekannt, für welche Regeln alle benötigten Entitäten und Relationen existieren. Nur von diesen werden beim Regelauswertungszyklus Regelinstanzen gesucht.

### 4.3.2.2 Beschreibung des Vorschlags

Dieser Vorschlag geht von ähnlichen Voraussetzungen aus wie der Vorschlag “Zwei Caches”. Es werden jedoch keine Caches für Teilregelinstanzen verwendet, sondern die Regeln und Kommandos um einen Zähler für jede formale Komponente erweitert. Dieser wird um eins erhöht, falls an die formale Komponente eine neu erzeugte Entität oder Relation gebunden werden kann. Attributsbedingungen werden nicht geprüft, da diese zum Teil von den gebundenen Entitäten abhängen und diese erst beim Bilden der Instanz bekannt sind.

Die Suche nach Regelinstanzen wird weiterhin verwendet, es müssen jetzt allerdings nur die Regeln betrachtet werden, von denen bekannt ist, dass alle nötigen Entitäten und Relationen existieren. Dieser Vorschlag schränkt damit die Anzahl der zu untersuchenden Regeln ein.

Die Behandlung von Benutzerkommandos bleibt unverändert.

## 4.3.3 Vorschlag - “Entitäteniterator”

### 4.3.3.1 Kernidee

Während des Regelauswertungszyklus wird nicht über die Regeln iteriert, sondern über alle existierenden Entitäten und ausgehend davon alle Regelinstanzen berechnet.

### 4.3.3.2 Beschreibung des Vorschlags

Dieser Vorschlag verwendet ebenfalls Informationen, welche Entitäts- und Relationstypen von den Regeln und Kommandos benötigt werden. Weiterhin wird ein Cache benötigt, der die teilweise gebundenen Instanzen der Regeln und Kommandos aufnimmt. Zu Beginn eines Regelauswertungszyklus werden alle Entitäten und Relationen durchlaufen und die Regelinstanzen berechnet. Erst wenn alle durchlaufen sind, darf mit der Ausführung begonnen werden, da erst zu diesem Zeitpunkt mit Sicherheit die ausführbare Instanz der Regel mit der höchsten Priorität feststeht. Bei jeder Veränderung des Zustands durch Erzeugen und Löschen von Entitäten und Relationen muss die Menge der Regelinstanzen entweder neu berechnet werden oder nur die Veränderung an der Menge.

Um ein Kommando auszuführen, ist es ebenfalls notwendig alle Entitäten und Relationen zu durchlaufen und so die Menge der Instanzen des auszuführenden Kommandos zu berechnen.

## 4.4 Regelausführung

In diesem Kapitel befinden sich Vorschläge, die sich auf die Ausführung der Aktionsteile von Regeln und Kommandos beziehen.

### 4.4.1 Vorschlag - “Shared Library”

#### 4.4.1.1 Kernidee

Die in den Modellen definierten Funktionen und die Aktionsteile von Benutzerkommandos und Regeln werden zu Ada95 Code compiliert und als Shared Library dynamisch an den Simulator gebunden.

#### 4.4.1.2 Beschreibung des Vorschlags

Jedes Modell wird, bevor es ausgeführt wird, nicht nur wie bisher in die Basissprache übersetzt, sondern zusätzlich in Ada95 Code übersetzt und zu einer Bibiliothek gebunden. Es gibt dabei zwei Möglichkeiten zur Umsetzung:

- Die Bibliothek wird dynamisch gebunden, so dass die Basismaschine einmal übersetzt werden muss, aber mit vielen verschiedenen Modellen verwendet werden kann.
- Die Bibliothek wird zusammen mit der Basismaschine zu einem Simulator zusammengebunden. Dies hat den Nachteil, dass bei Austausch des Modells oder bei Änderungen an dem Modell, der SESAM-Simulator neu gelinkt werden muss.

Die Bibliothek enthält in den Modellen definierte Funktionen und die Aktionsteile der Benutzerkommandos und Regeln. Jeder Aktionsteil wird dabei zu einer Prozedur, die alle von ihr benötigten Parameter übergeben bekommt, wie zum Beispiel an die Regel gebundene Entitäten.

Soll eine Regelinstanz ausgeführt werden, so wird die zugehörige Prozedur der Shared Library aufgerufen.

Da aus den Aktionsteilen auch auf den Zustand der Basismaschine zugegriffen werden kann, ist es nötig, ebenfalls Aufrufe aus der Bibliothek auf die Basismaschine zuzulassen.

### 4.4.2 Vorschlag - “Stateprotocolcache”

#### 4.4.2.1 Kernidee

Die Ausgabe des Stateprotocol wird gecached.

#### 4.4.2.2 Beschreibung des Vorschlags

Die folgenden Ausdrücke sind bei der Analyse mit eingeschaltetem Stateprotocol durch eine besonders lange Ausführdauer aufgefallen:

- Statements, mit denen Entitäten und Relationen erzeugt oder gelöscht werden können.
- Expressions für Zugriffe auf Entitätsattribute.

Der zeitliche Unterschied entsteht, da bei diesen Operationen ein Eintrag in die Stateprotocoldatei vorgenommen wird.

Bei jedem dieser Ausdrücke wird die Stateprotocoldatei geöffnet, ein Eintrag an die Datei angehängt und wieder geschlossen. Dies lässt vermuten, dass durch cachen der Schreibzugriffe Zeit eingespart werden kann. Die auszugebenden Daten werden so lange in einen Cache geschrieben, bis dieser voll ist. Dieser wird dann auf einmal an die Datei angehängt.

### 4.4.3 Vorschlag - “State”

#### 4.4.3.1 Kernidee
Effizienterer Zugriff auf die Datenstruktur im State_Manager.

#### 4.4.3.2 Beschreibung des Vorschlags
Der State_Manager verwaltet alle Entitäten und Relationen, die während einer Simulation existieren.

Die Datenstruktur des Zustands besteht aus Container-Listen. Ein Element einer Liste enthält jeweils einen Typ und alle dazugehörigen Entitäten oder Relationen. Sollen beispielsweise die Entitäten eines bestimmten Typs gefunden werden, so muss die Liste durchlaufen werden, bis das Element mit dem gesuchten Entitätstyp gefunden ist.

Die Zugriffe können beschleunigt werden, indem durch den Typ direkt auf das richtige Element der Liste zugegriffen werden kann. Eine Möglichkeit dafür ist, beim Laden des Modells jedem Entitäts- und Relationstyp eine eindeutige ID zuzuordnen, über die die Listenelemente indiziert werden können. Nach dem Laden werden dann die Listen des States mit den jeweiligen Elementen initialisiert.

## 4.5 Modelle

### 4.5.1 Vorschlag - “Array statt Bag”

#### 4.5.1.1 Kernidee
Verwendung von Arrays anstatt Bags in den Modellen.

#### 4.5.1.2 Beschreibung des Vorschlags
Die nähere Untersuchung der Regelausführung hat bei manchen Regeln eine extrem hohe Anzahl von ausgeführten Statements und Expressions ergeben, zum Teil mehr als 40000 bei einer Ausführung.

Als ein Problem hat sich dabei die Verwendung von Bags in den Modellen herausgestellt, da bei der Suche nach einem bestimmten Element des Bags durchschnittlich die Hälfte aller Elemente betrachtet werden muss.

Unter der Annahme, dass die einzelnen Elemente aufgrund einer eindeutigen ID gesucht werden, kann anstatt eines Bags ein Array verwendet werden, der die ID der Elemente als Index verwendet. Dadurch kann die Anzahl der auszuführenden Statements drastisch reduziert werden. Dies ist möglich, da Elemente des Bags einzelne Anforderungen darstellen und jede dieser Anforderungen durch eine eindeutige ID gekennzeichnet ist.

### 4.5.2 Vorschlag “Base2 Bag”

#### 4.5.2.1 Kernidee

Implementierung des abstrakten Datentyps Bag in der Basismaschine und Verwendung in den Modellen.

#### 4.5.2.2 Beschreibung des Vorschlags

Der Datentyp des Bags ist nur modellseitig implementiert und baut auf dem generischen Typ der Liste auf.

Falls die in Vorschlag “Liste statt Bag” getroffene Annahme, dass die IDs der Elemente eines Bags eindeutig sind, nicht zutrifft, so gibt es die Möglichkeit, den generischen Datentyp eines Bags in der Basismaschine zu implementieren. Als Zusatz wäre ein effizienter Zugriff auf die einzelnen Elemente durch eine Hashtable sinnvoll.

## 4.6 Bewertung

### 4.6.1 “Zwei Caches”

Während einer Simulation werden bei den Modellen QS und QSVA rund 5000 Entitäten und Relationen erzeugt und gelöscht. In jedem Spiel werden circa 300 Regelauswertungszyklen durchgeführt.

Dies macht durchschnittlich (5000 / 300) = 16 zu betrachtende Veränderungen am Zustand zwischen zwei aufeinanderfolgenden Regelauswertungszyklen.

Es existieren rund 700 Regeln in den Modellen. Jede Regel besitzt durchschnittlich 6 formale Komponenten in den Strukturteilen, insgesamt also 700 * 6 = 4200 formale Komponenten, die auf circa 300 Entitätstypen und Relationstypen verteilt sind. Unter der Annahme, dass die Typen gleichmäßig verteilt sind, müssen pro erzeugter oder gelöschter Entität oder Relation (4200 / 300) = 14 formale Komponenten und damit 14 / 6 = 2 Regeln betrachtet werden.

Diese Zahl muss noch mit der Anzahl der im Cache existierenden Regelinstanzen je Regel multipliziert werden, da jede dieser Regelinstanzen betrachtet werden muss. Diese Zahl kann erst nach Durchführung der Veränderung gemessen werden, da ihre Berechnung nicht ohne genaue Kenntnis des Zustands möglich ist.

Zwischen zwei Regelauswertungszyklen ergeben sich somit 16 Veränderungen * (14 formale Komponenten pro Veränderung) * Faktor = 224 * Faktor, die durchschnittlich zwischen zwei Regelauswertungszyklen betrachtet werden müssen. Dazu kommen die zu prüfenden Attributsbedingungen, die von der Anzahl an Regelinstanzen in “Regelcache 2” abhängen.

Im Vergleich dazu werden während eines Regelauswertungszyklus mit der derzeitigen Regelinstanzsuche durchschnittlich 5323 IC_Statements betrachtet.

Selbst bei großzügiger Abschätzung der Regelinstanzen pro Regel werden deutlich weniger Funktionsaufrufe benötigt als bei der bisherigen Regelinstanzsuche. Der Geschwindigkeitsvorteil kann als groß bezeichnet werden.

Es wird ein neues Paket benötigt, das die beiden Caches verwaltet und das Konzept der Instanzsuche muss verändert werden. Das Verwalten der Regelinstanzen ist bei genauerer Betrachtung keine triviale Aufgabe, die als sehr zeitkritisch einzustufen ist.

Der Vorschlag wird als schwer eingestuft, da die Konzepte neu sind und die Veränderungen an der Basismaschine sehr groß sind.

Dieser Vorschlag beeinflusst das Verhalten der Basismaschine folgendermaßen: Die Reihenfolge der ausgeführten Regeln kann sich im Vergleich zur jetzigen Regelinstanzsuche innerhalb der Prioritätsstufen ändern. Dies liegt zum einen an der nicht mehr vorhandenen Sortierung der Regeln nach Namen und zum anderen daran, dass nicht zugesichert werden kann, dass die Reihenfolge von mehreren Instanzen einer Regel gleich bleibt.

### 4.6.2 “Zähler”

Während eines Regelauswertungszyklus werden derzeit 1000 bis 1500 Regeln untersucht und nur durchschnittlich 60 davon ausgeführt. Bei rund 700 Regeln in den Modellen bedeutet das, dass manche Regeln mehrmals erfolglos untersucht werden. Dies kommt daher, da sich beim Ausführen des Aktionsteils einer Regel die Menge der ausführbaren Regeln verändern kann und somit alle Regeln der gleichen Prioritätsstufe erneut untersucht werden müssen.

In diesem Vorschlag können Regeln ausgeschlossen werden, denen eine benötigte Entität oder Relation fehlt, bevor von diesen Regelinstanzen gesucht werden. Es ist schwer abzuschätzen, wie viele Regeln wirklich vor der eigentlichen Regelinstanzsuche ausgeschlossen werden können, da dazu eine genaue Untersuchung der Verteilung von Entitätstypen und Relationstypen auf die Regeln nötig ist. Zusätzlich müsste noch die genaue Anzahl an Entitäten und Relationen eines bestimmten Typs während der Simulation betrachtet werden.

Während bisherigen Simulationen existieren zu jedem Zeitpunkt jeweils zwischen 80 und 100 Entitäten und 30 bis 50 Relationen. Bei 300 verschiedenen Entitäts- und Relationstypen kann davon ausgegangen werden, dass bis zur Hälfte der Regeln ausgeschlossen werden kann, da eine benötigte Entität oder Relation fehlt. Allerdings nur, wenn wirklich alle Typen als formale Komponenten in dem Bedingungsteil von Regeln vorkommen. Der Geschwindigkeitsvorteil muss als mittel eingestuft werden, da zur bisherigen Suche nach Regelinstanzen noch die Berechnung der Zähler bei Veränderungen am Zustand hinzukommt. Nach Überlegungen aus dem Vorschlag “Zwei Caches” kommt man auf circa 16 Veränderungen * (14 formale Komponenten pro Veränderung) = 224 Veränderungen an den Zählern der formalen Komponenten.

Der Schwierigkeitsgrad ist geringer als bei dem Vorschlag “Zwei Caches”. Er ist jedoch immer noch als mittel bis schwer einzustufen aufgrund der vielfältigen Veränderungen an der Basismaschine.

Es sind keine Veränderungen im Verhalten der Basismaschine zu erwarten.

### 4.6.3 “Entitäteniterator”

Während jedem Auswertungszyklus müssen alle Entitäten und alle Relationen betrachtet werden. Durchschnittlich existieren 100 Entitäten und 50 Relationen im Zustand. Mit den Zahlen aus Vorschlag “Zwei Caches” kommt man damit auf 150 * 6 formale Komponenten pro Regel = 900 formale Komponenten, die zu Beginn eines Regelauswertungszyklus geprüft werden müssen. Dazu kommen noch die Check_Stmts, die vor der Ausführung einer Regel geprüft werden müssen. Diese Zahl ist jedoch schwer abzuschätzen, da sie von der Anzahl der Teilregelinstanzen in Cache 2 und der Anzahl an ausgeführten Regeln abhängt. Wird der Aktionsteil einer Regel ausgeführt, so müssen entweder alle Regelinstanzen neu berechnet werden, oder wie bei Vorschlag “Zwei Caches” nur die Veränderungen an der Menge der Regelinstanzen berechnet werden.

Bei Fall 1 müssen diese 900 formale Komponenten nach jeder Ausführung einer Regel überprüft werden, was bei derzeit durchschnittlich 60 ausgeführten Regeln circa 54000 formale Komponenten während eines Regelauswertungszyklus macht.

Bei Fall 2 kommen zu den 900 formale Komponenten noch circa 224 hinzu, da für jede erzeugte oder gelöschte Entität und Relation alle betroffenen Regeln geprüft werden müssen. Die Zahl stammt aus Annahmen des Vorschlags “Zwei Caches”.

Alle Zahlen müssen noch mit einem Faktor, der die Anzahl an Teilregelinstanzen bezeichnet, multipliziert werden, da eine formale Komponente für jede existierende Teilregelinstanz einer Regel geprüft werden muss.

Der Schwierigkeitsgrad ist wie bei Vorschlag “Zwei Caches” als hoch einzustufen, da hier ebenfalls ein neues Paket erstellt werden muss und große Veränderungen an den Datenstrukturen und dem Regelauswertungszyklus nötig sind.

Der Geschwindigkeitsvorteil ist als gering einzustufen, da bei Fall 1 die Anzahl der zu untersuchenden formalen Komponenten deutlich höher wie bei der bisherigen Regelinstanzsuche ist. Bei Fall 2 darf die durchschnittliche Anzahl an Teilregelinstanzen die Zahl 18 nicht übersteigen, da sonst dieser Fall ebenfalls schlechter als bisher abschneidet.

Im Vergleich dazu werden während eines Regelauswertungszyklus mit der derzeitigen Regelinstanzsuche durchschnittlich 5323 IC_Statements betrachtet.

### 4.6.4 “Shared Library”

Bisher muss die Basismaschine die Aktionsteile der Regeln und Kommandos und die modellseitig definierten Funktionen Statement für Statement auswerten und dies jedes mal, sollen diese ausgeführt werden. Als Bibliothek an die Basismaschine gebunden, könnten diese sehr schnell als nativer Code ausgeführt werde. Der Geschwindigkeitsvorteil ist als groß einzuschätzen.

Da zum einen sehr große Veränderungen an der Basismaschine notwendig werden und zum anderen ein neuer Compiler erzeugt werden muss, der die Modelle in Ada-Code übersetzt, muss dieser Vorschlag als schwer eingestuft werden.

### 4.6.5 “Stateprotocolcache”

Die Schwierigkeit dieses Vorschlags kann als niedrig eingestuft werden, da die Anzahl der Veränderungen gering ist und das Prinzip des gecachten Schreibens in eine Datei allgemein bekannt ist.

Die folgenden Statements enthalten Aufrufe, die Einträge an das Stateprotocol anhängen:

| **Ort des Aufrufs** | **Größenordnung der Anzahl der Aufrufe** |
|---|---|
| Zuweisung an ein Entitätsattribut | 100000 + |
| Erzeugen und löschen einer Entität | 1000+ |
| Erzeugen und löschen einer Relation | 1000+ |
| Laden des Base2 Modells | 1 |
| Laden des Situations Modells | 1 + |
| Abspeichern des Zustands | 1 + |

**Tabelle 12. Anzahl der Stateprotocolaufrufe**

Da bei jedem Aufruf die Stateprotocoldatei geöffnet wird, der Protokolleintrag an diese angehängt und wieder geschlossen wird, hat dies besonders bei häufigen Aufrufen einen negativen Einfluss auf die Laufzeit. Durch eine Beschleunigung dieser Aufrufe könnten die Statements und Expressions profitieren, die Zuweisungen an Entitätsattribute vornehmen und Entitäten oder Relationen erzeugen

oder löschen. Das Erzeugen von Entitäten zieht besonders Nutzen daraus, da hier mehrere Zuweisungen an Entitätsattribute vorkommen.

Je größer der Cache desto weniger einzelne Schreibeoperationen auf die Stateprotocoldatei sind notwendig. Es muss ein gutes Verhältniss zwischen verbrauchtem Speicherplatz und eingesparten Schreibeoperationen gefunden werden. Der Geschwindigkeitsvorteil kann als mittel eingeschätzt werden, da nur bestimmte Statements und Expressions von dieser Beschleunigung profitieren.

Es sind keine Veränderungen im Verhalten der Basismaschine zu erwarten.

Das Mitschreiben des Stateprotocols stellt nicht den Normalfall dar und wird nur selten verwendet.

### 4.6.6 “State”

Bei den derzeitigen Modellen existieren circa 150 verschiedene Relationstypen und 150 verschiedene Entitätstypen. Durch die Veränderung kann die Anzahl der besuchten Listenelemente stark reduziert werden. Pro gesuchtem Typ wird nur eine Operation notwendig, anstatt der bisher durchschnittlichen n/2 Vergleiche.

Hauptsächlich von der Veränderung betroffen sind das Erzeugen und Löschen von Entitäten und Relationen, sowie vordefinierte Funktionen, die aus den Modellen aufgerufen werden können. Der Geschwindigkeitsvorteil kann somit als mittel bezeichnet werden.

Die Schwierigkeit des Vorschlags kann mit mittel angegeben werden, da die Veränderungen nur auf wenige Bereiche der Basismaschine beschränkt sind.

Veränderungen am spezifizierten Verhalten der Basismaschine sind nicht zu erwarten.

### 4.6.7 “Array statt Bag”

In den Modellen gibt es nur wenig verschiedene Variablen vom Typ Bag, welche jedoch recht häufig in den Aktionsteilen der Regeln verwendet werden. In Frage kommen die Regeln, welche Dokumente bearbeiten, da der Inhalt der Dokumente durch Bags mit den Anforderungen als Elemente implementiert ist. Da der Zugriff auf Elemente der Bags durch einmalig definierte Funktionen geschieht, kann die Schwierigkeit des Vorschlags als einfach eingestuft werden.

Wird eine Anforderung in einem Bag aufgrund der ID gesucht, so müssen im Mittel die Hälfte aller Elemente betrachtet werden. Durch die Verwendung von Arrays kann der Zugriff auf ein Element mit einer bestimmten ID auf 1 gesenkt werden.

Der Geschwindigkeitsvorteil kann als groß bezeichnet werden, besonders da dieses Konstrukt sehr häufig in Aktionsteilen von Regeln verwendet wird.

Veränderungen an dem Verhalten der Basismaschine sind nicht zu erwarten.

### 4.6.8 “Base2 Bag”

Die Implementierung des Bags als generischen Datentyp hat mehrere Vorteile:

- Da die an den Bags beteiligten Funktionen in den Modellen definiert sind, müssen diese Statement für Statement durch die Basismaschine interpretiert werden. Durch die Implementierung in der Basismaschine werden die Funktionen ohne Umweg als Ada Code ausgeführt.
- Durch den möglichen Zugriff auf einzelne Elemente mit einer Hashtable sind zum einen mehrere Elemente mit gleichen IDs erlaubt, zum anderen wird der Zugriff auf die Elemente effizienter.

Durch diesen Vorschlag werden gleichermaßen Veränderungen an der Basismaschine, dem Hochsprachenübersetzer, wie an den Modellen notwendig. Da Veränderungen an den Modellen nicht Teil dieser Diplomarbeit sind, kann nur die Grundlage gelegt werden, indem der Datentyp eines generischen Bags implementiert wird.

Bei der Implementierung kann auf bereits bestehende generische Datentypen, wie einem dynamischen Array und einer Liste aufgebaut werden. Die Algorithmen für Bags und Hashtables sind allgemein bekannt. Es sind jedoch größere Erweiterungen bei dem Compiler, der Interpretation der Statements und Expressions notwendig, sowie bei der Verwaltung der Werte im Zustand der Basismaschine. Somit muss die Schwierigkeit dieses Vorschlags als schwer eingestuft werden.

Der Geschwindigkeitsvorteil kann als hoch eingestuft werden, da auch hier nur wenige Statements notwendig sind, um auf ein bestimmtes Element in einem Bag zuzugreifen.

Veränderungen am spezifizierten Verhalten der Basismaschine sind nicht zu erwarten.

## 4.7 Auswahl

Die Vorschläge werden aufgrund von mehreren Kriterien ausgewählt.

- Wie passt der Vorschlag zu der Aufgabenstellung der Diplomarbeit (1 sehr gut, 2 mittel, 3 kaum),
- der zu erwartende Geschwindigkeitsvorteil (1 groß, 2 mittel, 3 klein),
- und die Schwierigkeit der Umsetzung (1 gering, 2 mittel, 3 groß).

Bei der Auswahl der Vorschläge müssen noch folgende Bedingungen beachtet werden:

- Die Vorschläge “Zwei Caches”, “Zähler“ und “Entitäteniterator” schließen sich gegenseitig aus. Es kann nur einer der drei ausgewählt werden.
- Die Vorschläge “Liste statt Bag” und “Base2 Bag” beziehen sich zum Großteil auf Veränderungen an Modellen und können somit nur als Vorlage für zukünftige Veränderungen dienen.

Da das Mitschreiben des Stateprotocols nur in Ausnahmefällen geschieht, sind ähnlich gute Vorschläge diesem Vorzuziehen.

| **Vorschlag** | Zwei Caches | Liste statt Bag | Zähler | Stateprotocol-cache | State | Entitäte-niterator | Base2 Bag | Shared Library |
|---|---|---|---|---|---|---|---|---|
| **Passt zur Aufgabenstellung** | 1 | 3 | 1 | 2 | 2 | 1 | 3 | 3 |
| **Veränderungen an der Basismaschine** | Siehe [1] | - | Siehe [1] | - | - | Siehe [1] | - | - |
| **Geschwindigkeits-vorteil** | 1 | 1 | 2 | 2 | 2 | 3 | 1 | 1 |
| **Schwierigkeit** | 3 | 1 | 3 | 1 | 2 | 3 | 3 | 3 |
| **Machbar** | Ja | Ja | Ja | Ja | Ja | Ja | Ja | Ja |

**Tabelle 13. Zusammenfassung**

1> Die Reihenfolge der ausgeführten Regeln innerhalb einer Prioriätsstufe ändert sich.

Aufgrund der berechneten Priorität und den zusätzlichen Bedingungen werden die folgenden Vorschläge ausgewählt. Die Vorschläge werden in der Reihenfolge bearbeitet, wie diese hier aufgeführt sind. Während des Entwurfs wird entschieden wie viele davon umgesetzt werden:

- “Zwei Caches”,
- “State”,
- “Stateprotocolcache”.

# Kapitel 5
# Entwurf

*Dieses Kapitel enthält den Entwurf der ausgewählten Vorschläge.*

## 5.1 Einführung

### 5.1.1 Entwurfsprinzipien
Die Basismaschine stellt ein großes System mit mehreren Teilsystemen dar. Diese sind in Schichten angeordnet und haben wohldefinierte Schnittstellen. In diesem Entwurf wird darauf geachtet, die Schnittstellen zwischen den einzelnen Schichten nicht zu verändern.

Innerhalb einer Schicht wird versucht die Veränderungen der Schnittstellen zwischen den Modulen auf ein Minimum zu beschränken. Außerdem soll der bestehende Code so wenig wie möglich geändert werden und Veränderungen möglichst in eigenen Modulen gekappselt werden.

## 5.2 “Zwei Caches” und “State”
Die beiden Vorschläge “Zwei Caches” und “State” haben Überschneidungen bei nötigen Veränderungen an der Basismaschine. Deshalb werden sie in einem Entwurf zusammengefasst.

### 5.2.1 Verweis auf betroffene Entwurfsdokumente
Die folgenden Dokumente besitzen durch diesen Entwurf nur noch eingeschränkt Gültigkeit:

- Entwurf der Basismaschine, Design Rationale und Grobentwurf, Version 9.11.1998,
- Feinentwurf des Teilsystems Compile, Version 1.0,
- Feinentwurf des Teilsystem Execute, Version 1.0,
- Feinentwurf des Teilsystems Symboltabelle, Version 1.0,
- Feinentwurf des Teilsystems State, Version 1.1.

### 5.2.2 Datenstrukturen
Bestimmte Datenstrukturen der Basismaschine müssen erweitert werden und es kommen neue hinzu. In diesem Abschnitt werden die jeweils betroffenen Datenstruktur beschrieben und die Veränderungen kenntlich gemacht. Neue Attribute bestehender Datentypen werden fett gedruckt, fallen Attribute weg, werden sie durchgestrichen.

#### 5.2.2.1 Command
Dieser Datentyp ist in dem Paket “command_table” des Teilsystems “symboltable” definiert. Er stellt die Kommandos der Modelle dar. Er enthält einen eindeutigen Namen, die Zeit, die durch das Kommando verbraucht wird, alle formalen Parameter, formale Komponenten, lokale Variablen, den Deklarationsteil, den Bedingungsteil, den Aktionsteil und die IC_Statements des Kommandos. Der Datentyp wird um eine eindeutige ID erweitert.

Die Liste der IC_Statements kann entfallen, da die Instanzsuche nicht mehr benötigt wird.

```
type Command_Element is
  record
    Name:              Identifier;
    Time:              B2_Integer;
    Formal_Parameters: Declaration_List;
    Formal_Components: Declaration_List;
    Local_Variables:   Declaration_List;
    Declaration_Part:  Declaration_List;
    Condition_Part:    Expression_List;
    Action_Part:       Statement_List;
    IC_Stmts:          Instance_Creation_Stmt_List;
    ID:                Natural;
  end record;
```

#### 5.2.2.2 Entitätstyp

Dieser Datentyp ist in dem Paket “entity_table” des Teilsystem “symboltable” definiert und beschreibt, die im Modell definierten Entitätstypen. Er enthält den eindeutigen Namen des Entitätstyps, eine Liste aller konformen Entitätstypen und eine Liste aller Attribute des Typs. Er wird um eine eindeutige ID erweitert, die es ermöglicht, effizient auf neue Datenstrukturen zuzugreifen. Zusätzlich wird der Datentyp um eine Liste aller von dem Entitätstyp abgeleiteten Typen erweitert. Diese Liste enthält auch den Entitätstyp selbst.

```
type Entity_Type_Element is
  record
    Name:                 Identifier;
    Conform_Entity_Types: Entity_Type_List;
    Derived_Entity_Types: Entity_Type_List;
    Attributes:           Entity_Attribute_List;
    ID:                   Natural;
  end record;
```

#### 5.2.2.3 Entity-Container und Relation-Container

Diese Datentypen stellen die Verbindung zwischen den Entitäts- / Relationstypen und den betroffenen Regeln und Kommandos her. Hierdurch werden die, beim Erzeugen oder Löschen einer Entität oder Relation, betroffenen Regel- und Kommandoinstanzen gefunden.

Ein Container besteht aus einem Array, dessen Elemente aus einer Liste mit Rule-Declarations-Pairs und einer Liste von Command-Declarations-Pairs bestehen. Auf die einzelnen Elemente des Arrays kann über die ID des Entitäts- oder des Relationstyps als Index zugegriffen werden. Die Rule/ Command-Declarations-Pairs enthalten die jeweilige Regel / Kommando und die Declarations, an deren Position eine Entität oder Relation von dem bestimmten Typ in der Instanz gebunden werden kann. Desweiteren ist enthalten, ob ein Konflikt zwischen formalen Relationen in einer Regel oder einem Kommando besteht.

Das folgende Schaubild soll den Aufbau des Datentyps verdeutlichen. Es sind zwei Indizes eines Arrays zu sehen, die jeweils eine Liste von Rule-Declarations-Pairs und eine Liste von Command-Declarations-Pairs enthalten.

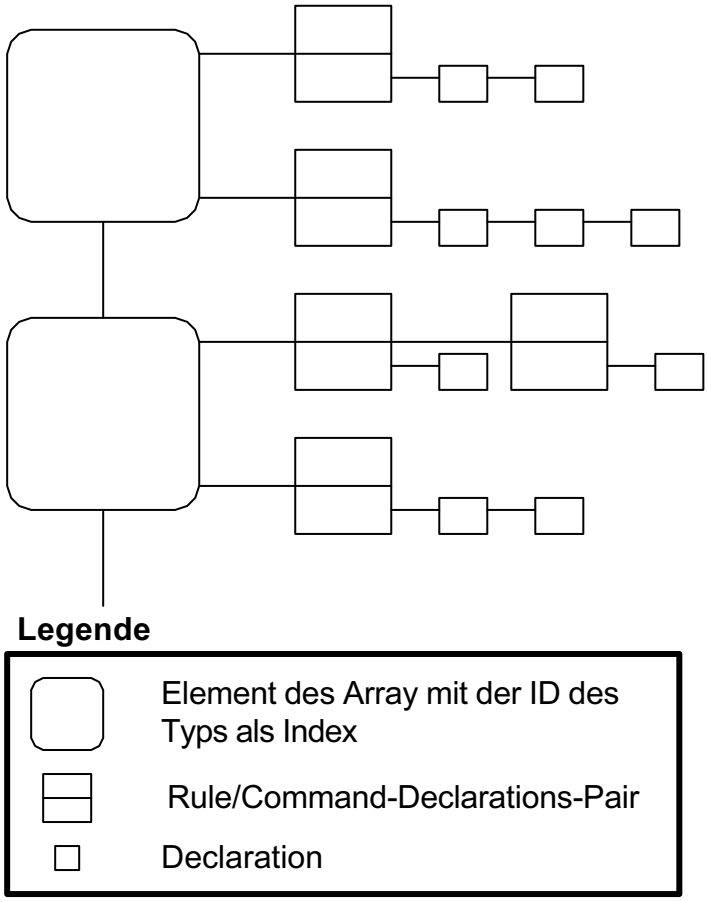



**Abbildung 15. Entity-/ Relation-Container**

#### 5.2.2.4 Instance

Der Datentyp Instance ist in dem Paket “rule_instance” des Teilsystem “Execute” definiert und kann entweder die Instanz einer Regel darstellen oder die eines Kommandos. Er enthält den Aktivierungsblock, in dem alle gebundenen Entitäten, Relationen und lokalen Variablen gespeichert sind und die Regel oder das Kommando, von dem dies eine Instanz ist.

Der Datentyp wird um drei Felder erweitert:

- Is_Executed zeigt an, ob diese Instanz schon ausgeführt wurde. Dies wird während des Regelauswertungszyklus benötigt.
- Count_Needed enthält die Anzahl an benötigten Entitäten, die an die Instanz gebunden sein müssen, damit diese ausgeführt werden kann.
- Copied_Declaration enthält die formale Entität oder Relation, weshalb die Instanz kopiert wurde. Dies wird benötigt, um entscheiden zu können, ob diese Instanz beim Löschen einer Entität oder Relation gelöscht werden kann.

```
type Instance_Record_Type (Instance_Kind: Instance_Kind_Type) is
  record
    Instance_Vars: AB_Type;
```

```
    Is_Executed  : Boolean;
    count_needed : Natural;
    Copied_Declaration: Declaration;
    case Instance_Kind is
      when IK_Rule =>
        Instance_Rule: Rule;
      when IK_Command =>
        Instance_Command: Command;
      end case;
  end record;
```

#### 5.2.2.5 Regelcache eins

Der Regelcache eins besteht aus einem Array, dessen Elemente aus einer Liste von teilweise gebundenen Regelinstanzen bestehen. Jedes Element enthält dabei Instanzen von jeweils einer Regel. Auf die einzelnen Elemente des Arrays kann über die ID der Regel als Index zugegriffen werden. Der Cache für die Kommandoinstanzen ist genauso aufgebaut. Es wird ebenfalls über die ID des Kommandos auf den Index zugegriffen.

#### 5.2.2.6 Regelcache zwei

Der Regelcache zwei besteht aus einem Array, dessen Elemente aus einer Liste von komplett gebundenen Regelinstanzen bestehen. Jedes Element enthält dabei Regelinstanzen von Regeln einer Prioritätsstufe. Die Elemente des Arrays werden über die Prioritätsstufe indiziert. Die Stufen sind in dem Bereich von 2000 bis 0.

#### 5.2.2.7 Relationstyp

Dieser Datentyp ist in dem Paket “relation_table” des Teilsystems “symboltable” definiert und beschreibt, die im Modell definierten Relationstypen. Er enthält einen eindeutigen Namen, eine Liste aller konformen Relationstypen und eine Liste aller beteiligten Rollen. Er wird um eine eindeutige ID erweitert, die es ermöglicht, effizient auf neue Datenstrukturen zuzugreifen. Zusätzlich wird der Datentyp um eine Liste aller von dem Relationstyp abgeleiteten Typen erweitert. Diese Liste enthält auch den Relationstyp selbst.

```
type Relation_Type_Element Is
record
  Name:                   Identifier;
  Conform_Relation_Types: Relation_Type_List;
  Derived_Relation_Types: Relation_Type_List;
  Roles:                  Relation_Role_List;
  ID:                     Natural;
end record;
```

#### 5.2.2.8 Rule

Dieser Datentyp ist in dem Paket “rule_table” des Teilsystems “symboltable” definiert und enthält alle Daten einer Regel. Er enthält einen eindeutigen Namen, die Priorität der Regel, die Zeit, die durch die Regel verbraucht wird, alle formalen Komponenten, lokale Variablen, den Deklarationsteil, den Bedingungsteil, den Aktionsteil und die IC_Statements der Regel.

Der Datentyp wird um eine eindeutige ID erweitert, die es ermöglicht, effizient auf den Regelcache eins zuzugreifen.

Die Liste der IC_Statements kann entfallen.

```
type Rule_Element is
  record
    Name:               Identifier;
    Priority:           B2_Integer;
    Time:               B2_Integer;
    Formal_Components: Declaration_List;
    Local_Variables:    Declaration_List;
    Declaration_Part:   Declaration_List;
    Condition_Part:     Expression_List;
    Action_Part:        Statement_List;
    IC_Stmts:           Instance_Creation_Stmt_List;
    ID:                 Natural;
  end record;
```

#### 5.2.2.9 State

Der Datentyp “State” enthält die Simulationszeit und Listen der Container für Entitätstypen und Relationstypen. Jeder Container enthält einen Entitäts- oder Relationstyp und alle Entitäten oder Relationen des jeweiligen Typs. Er ist im Modul “State_Manager” des Teilsystems “State” definiert. Da durch die Veränderung mit einem bestimmten Typ bekannt ist, welcher Container des States gemeint ist - durch die eindeutige ID - und die Menge der Container sich während einer Simulation nicht ändert, ist es sinnvoll anstatt einer Liste ein Array zu verwenden. Hierdurch kann die Suche nach einem Container für einen bestimmten Typ von n/2 auf 1 reduziert werden.

```
type State is
record
  Simulation_Time: B2_Date;
  -- Beschreibt DS fuer Entitaeten
  All_Entities:    Entity_Type_Entry_Array;
  -- Beschreibt DS fuer Relationen
  All_Relations:   Relation_Type_Entry_Array;
end record;
```

### 5.2.3 Aufteilung in Pakete

Die Veränderungen an dem bisherigen System sollen so gut wie möglich in eigenen Paketen gekapselt werden, um die Anpassung von dem bestehenden Code so gering als möglich zu halten. Das folgende Schaubild zeigt die neuen Module (mit vollständiger Methodenliste) und ihre Verknüpfung mit bereits bestehenden Modulen (ohne Methoden). Das Folgenden Schaubild der Module ist in UML-Notation.

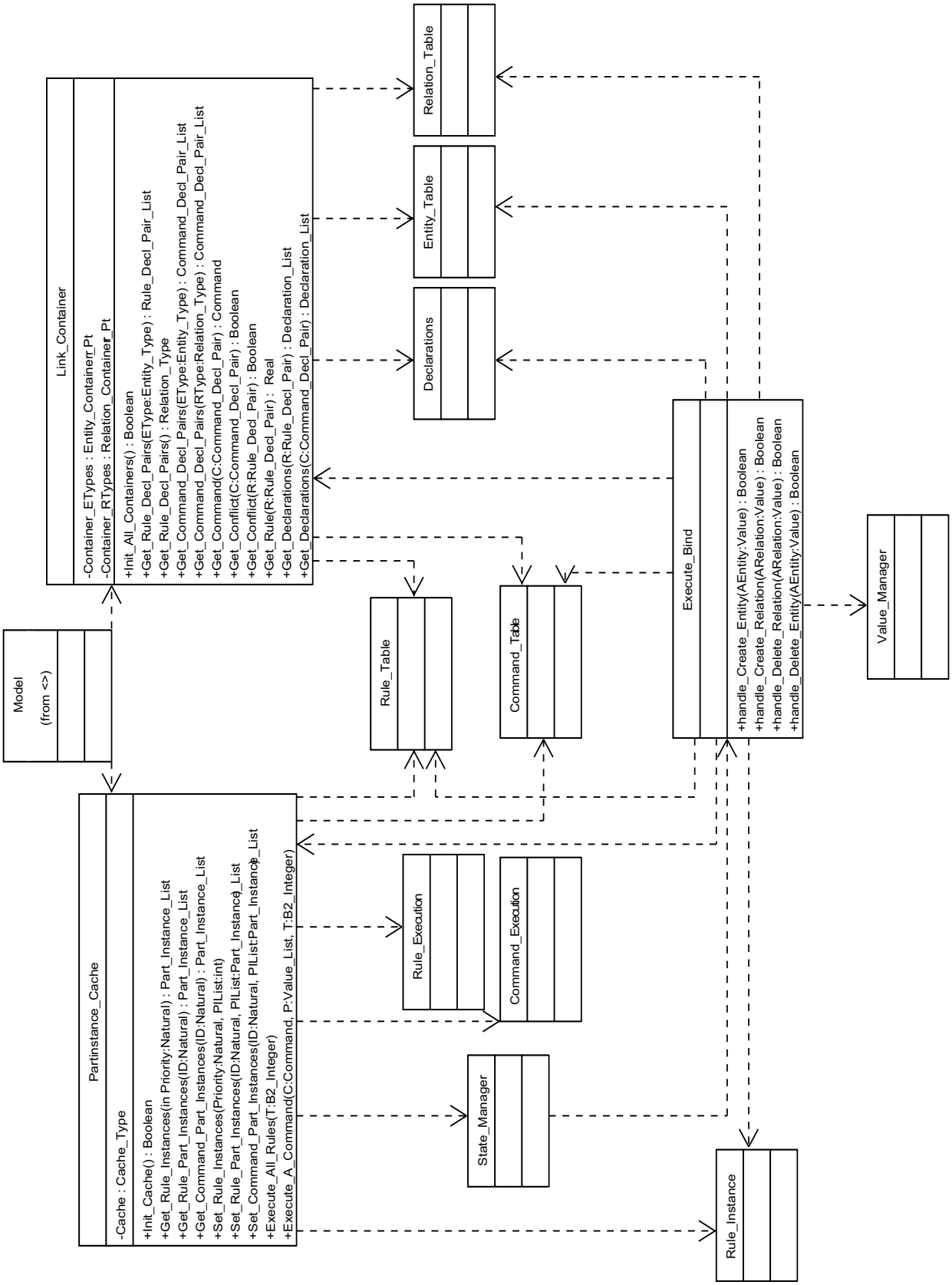



**Abbildung 16. Aufteilung in Pakete**

#### 5.2.3.1 command_execution
Das Modul command_execution ist für die Ausführung einer Kommandoinstanz zuständig. Die übergebenen Attribute des Kommandos werden in die Instanz eingetragen und dann alle Attributsbedingungen geprüft. Falls alle dieser Bedingungen erfüllt sind, wird die Instanz ausgeführt. Es wird die verbrauchte Zeit zurückgegeben.

#### 5.2.3.2 declarations
Es werden Methoden zum Erzeugen und Verwalten von lokalen Variablen und formalen Parametern, der Regeln und Kommandos, zur Verfügung gestellt. Die declarations werden in diesem Design zum korrekten Binden der Entitäten und Relationen an die Instanzen verwendet.

#### 5.2.3.3 entity_table
Dieses Modul verwaltet alle im Zusammenhang mit Entitätstypen abzuspeichernden Informationen, sowie Anfragen auf diese Information. Hierin ist der zu erweiternde Datentyp “Entity_Type” enthalten.

#### 5.2.3.4 execute_bind
Dieses Modul kapselt die Algorithmen, die benötigt werden, um die Regelinstanzen zu berechnen, falls Entitäten oder Relationen erzeugt oder gelöscht werden. Es müssen nur die jeweilige Funktionsaufrufe in die Module model und state_manager eingefügt werden. Die Algorithmen werden genauer im Abschnitt “Behandlung von Entitäten und Relation“ beschrieben.

#### 5.2.3.5 link_container
Dieses Modul kapselt den Datentyp, der die Verbindung zwischen den Entitäts- und Relationstypen und den Regeln und Kommandos herstellt. Für jeden Typ existiert eine Liste mit Rule-Declarations-Pairs und Command_Declarations_Pairs. Diese zeigen, an welcher Stelle eine Entität oder Relation von bestimmtem Typ in welcher Instanz einer bestimmten Regel oder eines Kommandos gebunden werden kann. Hier sind auch die Algorithmen enthalten, die diese Verbindungen berechnen. Die genauen Algorithmen sind im Abschnitt 5.2.4 enthalten.

#### 5.2.3.6 model
Dieses Modul ist eine Sammlung aller Schnittstellenoperationen aus den darunterliegenden Teilsystemen. Hierin werden Funktionen aufgerufen, die den Ablauf während einer Simulation steuern, wie das Laden eines Modells mit anschließender Initialisierung der Symboltabelle. Das Modul muss um Aufrufe an die Funktionen in den Paketen partinstance_cache und link_container erweitert werden, die die Initialisierung vornehmen.

#### 5.2.3.7 partinstance_cache
Das Modul partinstance_cache enthält drei Caches für Regel- und Kommandoinstanzen. Die Caches werden durch Arrays realisiert, deren Elemente Listen mit Regelinstanzen enthalten. Es gibt zwei Caches, die für jede einzelne Regel und jedes Benutzerkommando ein Element enthalten. Hierin sind Regelinstanzen gespeichert, die noch nicht alle benötigten Entitäten gebunden haben. Ein weiterer Cache ist nach der Prioritätsstufe der Regeln sortiert und enthält vollständig gebundene Regelinstanzen, von denen allerdings noch nicht bekannt ist, ob diese auch ausgeführt werden können. Es sind auch alle Zugriffe, die auf die Caches möglich sind, enthalten. Zusätzlich sind hier die Funktionen definiert, die zum Ausführen eines kompletten Regelauswertungszyklus und zum Ausführen eines Kommandos benötigt werden.

#### 5.2.3.8 relation_table
Dieses Modul verwaltet alle im Zusammenhang mit Relationstypen abzuspeichernden Informationen, sowie Abfragen auf dieser Information. Hierin ist der zu erweiternde Datentyp “Relation_Type” enthalten.

#### 5.2.3.9 rule_execution
Das Modul rule_execution ist für die Ausführung einer Regelinstanz zuständig. Es prüft alle Attributsbedingungen und führt die Instanz aus, falls alle dieser Bedingungen erfüllt sind. Es wird die verbrauchte Zeit zurückgegeben.

#### 5.2.3.10 rule_instance
Der abstrakte Datentyp dient der Aufnahme einer Instanz. Er kapselt dabei die zu verändernde Datenstruktur “instance”. Lokale Variablen, temporäre Variablen und formale Parameter werden ebenfalls in dieser Datenstruktur abgelegt. Der ADT kapselt aber auch alle Operationen, die auf Instanzen möglich sind.

#### 5.2.3.11 rule_table
Dieses Modul verwaltet alle im Zusammenhang mit Regeln abzuspeichernden Informationen, sowie Abfragen auf dieser Information. Hierin ist der zu erweiternde Datentyp “rule” enthalten.

#### 5.2.3.12 state_manager
Der state_manager verwaltet im Wesentlichen die Entitäten und Relationen des aktuellen Spielstandes. Dafür werden Entitäten und Relation (in den aufrufenden Modulen) erzeugt und in den state eingefügt. Es werden auch Operationen zum Löschen von Entitäten und Relationen angeboten, die die Konsistenz des States wahren. Wird zum Beispiel eine Entität gelöscht, werden auch alle Relationen gelöscht, an der die Entität beteiligt ist. Der state_manager wird zum einen um Funktionsaufrufe auf die Funktionen in dem Modul execute_bind erweitert. Zum anderen werden alle nötigen Veränderungen für den Vorschlag “State” vorgenommen. Die Datenstruktur wird verändert und muss initialisiert werden. Zusätzlich müssen die Zugriffsfunktionen angepasst werden.

### 5.2.4 Initialisierung
Die Verbindungen der Entitäts- und Relationstypen zu den Regeln und Kommandos, die diese in ihren Instanzen binden, ändern sich nicht während eines Simulationslaufes. Es genügt, diese während oder direkt nach dem Laden eines Modells zu erzeugen. Danach müssen sie nicht mehr überprüft werden. Im Folgenden werden die Bedingungen aufgestellt, die die Verbindungen erfüllen müssen und wann diese erstellt werden. Gleiches gilt für die ID’s der veränderten Datentypen und die Container des Moduls “state”. Das Schaubild “Initialisierung” verdeutlicht die zeitliche Abfolge der Initialisierung.

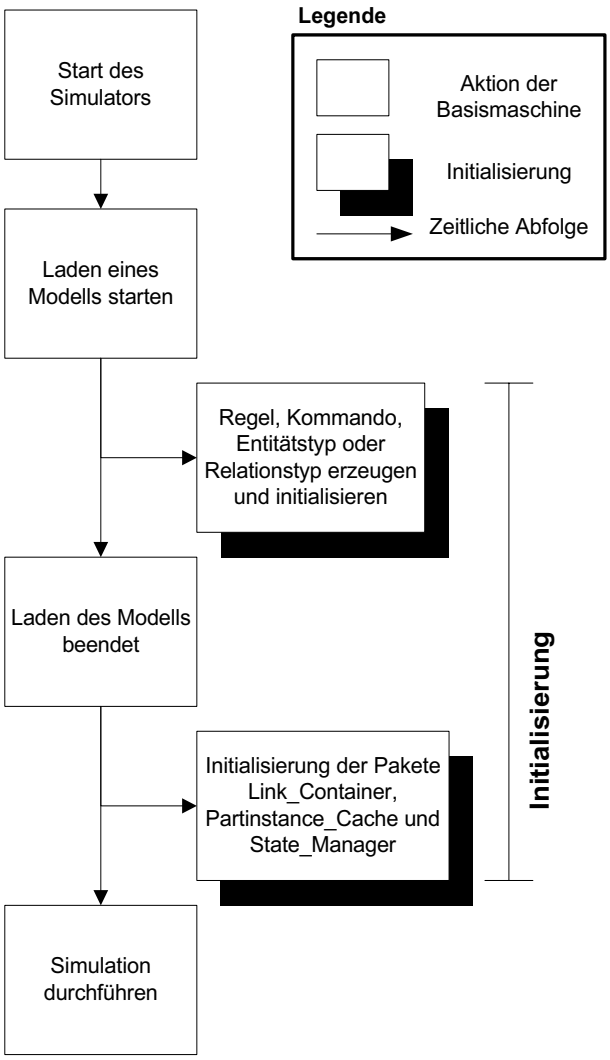



**Abbildung 17. Zeitlich Abfolge der Initialisierung**

### 5.2.4.1 Initialisierung der eindeutigen ID’s

Regeln, Kommandos, Entitätstypen, sowie Relationstypen muss eine eindeutige ID zugewiesen werden. Dies geschieht am einfachsten durch einen Zähler, der mit eins initialisiert wird. Es existiert je ein Zähler für Regeln, Kommandos, Entitäts- und Relationstypen. Diese werden jeder neu erzeugten Regel, Kommando, Entitäts-, Relationstyp als ID zugewiesen und danach um eins erhöht.

### 5.2.4.2 Initialisierung der Link_Container

Um die Verbindung zwischen den Entitäts-/ Relationstypen und den betroffenen Regeln und Kommandos herzustellen, wird über alle Regeln und Kommandos iteriert. Jede Regel und jedes Kommando besitzt einen Strukturteil, der Informationen darüber enthält, welche Typen von Entitäten und Relationen von deren Instanzen benötigt werden.

Zuerst wird geprüft, ob zwischen formalen Relationen in einer Regel oder eines Kommandos ein Konflikt besteht. Ein Konflikt bedeutet, dass zwei oder mehr formale Relationen auf dieselbe formale Entität verweisen. Ist dies der Fall, so wird für jedes Rule/Command-Declarations-Pair der Regel oder des Kommandos das Flag Conflict auf true gesetzt. Da bei jeder erzeugten Relation alle beteiligten Entitäten an die formalen Entitäten gebunden werden, kann es vorkommen, dass eine Entität eine

bereits korrekt gebundene Entität überschreibt und somit für die ursprünglich korrekt gebundene Relation eine Entität falsch gebunden ist.

Danach werden alle formale Relationen eines Strukturteils behandelt. Es wird der Typ der formalen Relation festgestellt. Da alle von diesem Typ abgeleiteten Relationstypen zu dieser formalen Relation passen, wird für jeden der abgeleiteten Typen entweder ein Paar bestehend aus der gerade betrachteten Regel oder des Kommandos und einer Liste von formalen Relationen erstellt, oder die formale Relation in eine bestehende Liste eingetragen. Die formalen Entitäten, die an der Relation beteiligt sind, werden als bereits behandelt markiert.

Nachdem alle formalen Relationen betrachtet sind, werden die nicht an einer formalen Relation beteiligten Entitäten behandelt. Auch hier wird für jede formale Entität der Typ festgestellt und entweder ein neues Paar erstellt oder die formale Entität an eine bestehende Liste angehängt.

Nach der vollständigen Initialisierung ist bekannt, welche Entität oder Relation an welcher Position in welcher Instanz gebunden werden kann. Das folgende Beispiel verdeutlicht nochmals die Zusammenhänge anhand des Strukturteils einer Regel und wie die formalen Entitäten und Relationen in den Link_Container eingetragen werden.

Das Folgende Diagram stellt zuerst dar, welche Entitäts- und Relationstypen existieren und in welcher Beziehung sie zueinander stehen. Der Strukturteil einer Regel enthält drei formale Entitäten und eine formale Relation. Die Linien von der Darstellung des Linkcontainers zu den formalen Komponenenten deuten den Inhalt des Linkcontainers an. Zum Beispiel existiert von den Containern der Entitätstypen Person, Kunde und Mitarbeiter je eine Verbindung zu der formalen Entität mit dem Typ Person, da diese Typen von Person abgeleitet sind.

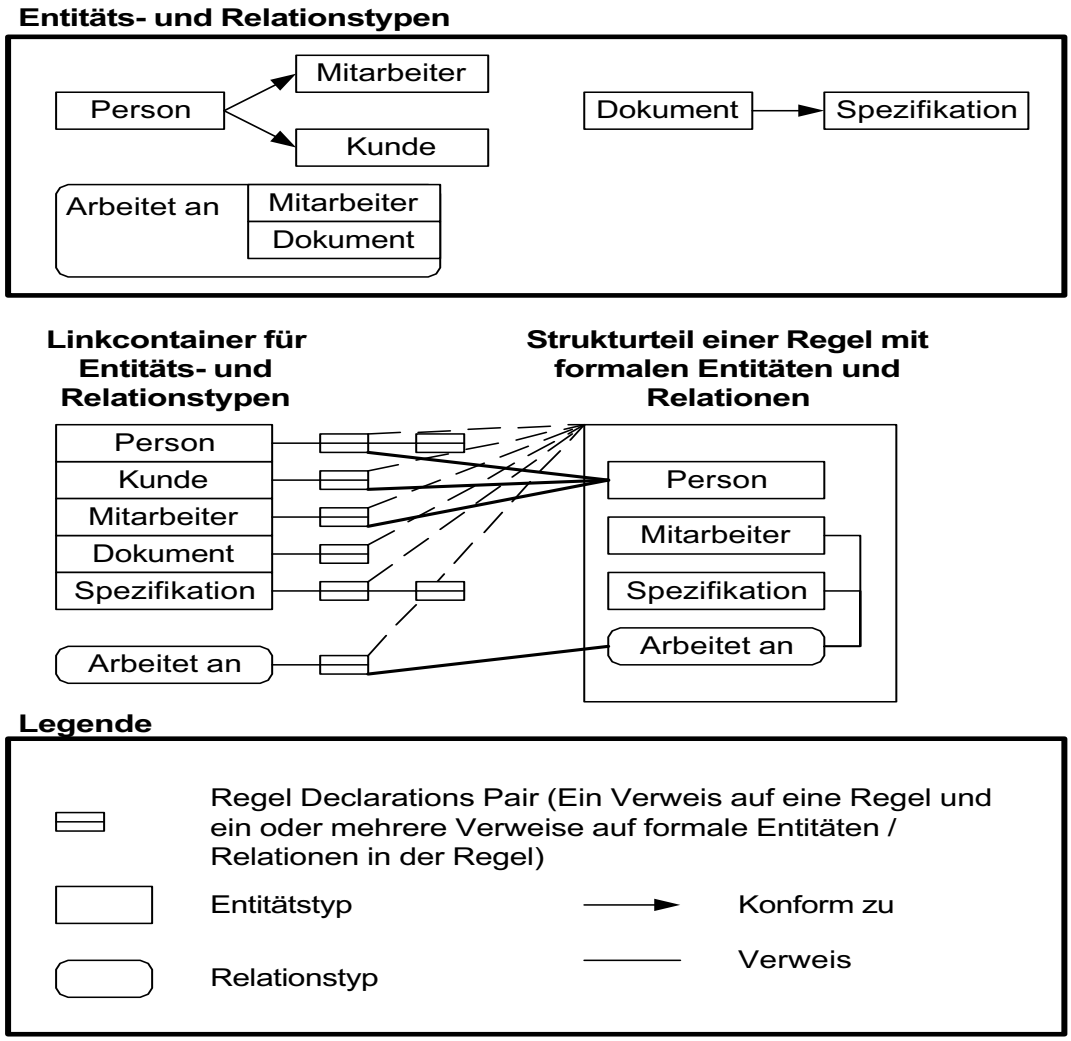



**Abbildung 18. Zusammenhang zwischen Link_Container und Strukturteil einer Regel**

### 5.2.4.3 Initialisierung des Partinstance_Cache

Nachdem das Modell komplett geladen ist, kann der Partinstance_Cache initialisiert werden. Der Regelcache eins und der Kommandocache werden mit je einer Instanz von jeder Regel und jedem Kommando initialisiert. Dabei wird für jede Instanz die Anzahl der benötigten Entitäten berechnet und eingetragen. Diese Zahl ist wichtig, um schnell entscheiden zu können, ob eine Instanz ausgeführt werden kann oder nicht.

### 5.2.4.4 Initialisierung für “State”

Die Initialisierung findet in zwei Stufen statt.

Die erste Stufe findet während dem Laden des Modells statt. Wird ein Entitäts- oder Relationstyp in der Symboltabelle erzeugt und ein konformer Typ in die Conformingliste eingetragen, so werden auch in allen betroffenen Typen die Derivedlisten auf den neuesten Stand gebracht. Die Conformingliste enthält dabei alle Typen, für die der aktuelle Typ eingesetzt werden kann, die Derivedliste enthält alle Typen die für den aktuellen Typ eingesetzt werden können, also auch den

aktuellen Typ. Das Schaubild “Vererbungsbaum” beschreibt die Zusammenhänge nochmals genauer. Es zeigt einen Vererbungsbaum von Typen und den Inhalt der entsprechenden Listen.

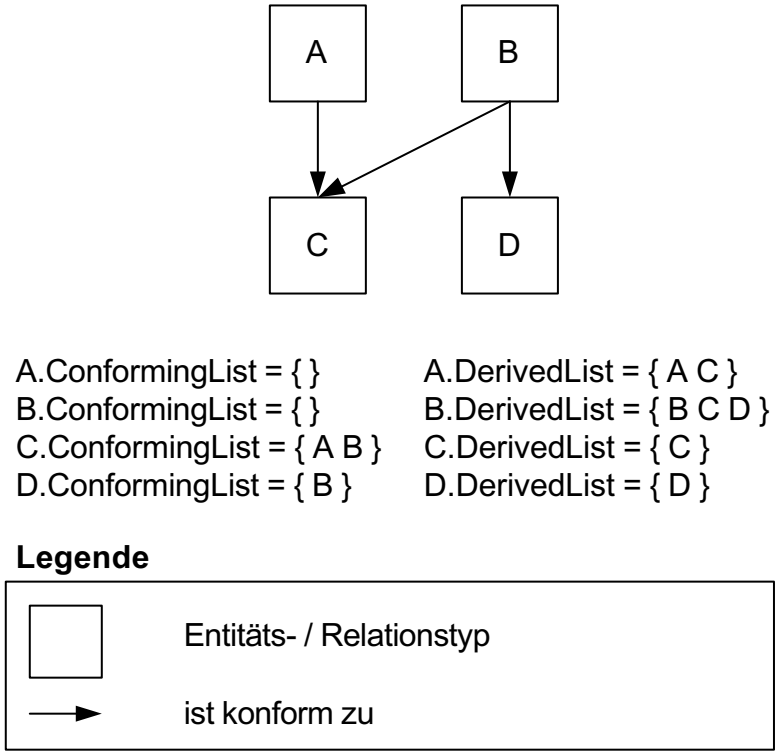



**Abbildung 19. Vererbungsbaum**

Die zweite Stufe findet statt, nachdem das Modell komplett geladen ist und die Startsituation initialisiert wurde. Bisher werden zu Beginn keine Container im State erzeugt, die die Relations- und Entitätstypen aufnehmen. Erst wenn Entitäten oder Relationen eines bestimmten Typs erzeugt werden, wird auch ein Containerelement für den Typ erzeugt.

Jetzt werden während der Initialisierung zwei Arrays erzeugt, die die Container für Relations- und Entitätstypen aufnehmen. Es werden die Listen für Entitäten und Relationen initialisiert.

## 5.2.5 Behandlung von Entitäten und Relationen

Wird eine Entität oder Relation erzeugt oder gelöscht, müssen alle Instanzen der betroffenen Regeln und Kommandos betrachtet werden. Durch die Paare in dem Link_Container ist festgelegt, an welche Instanz und an welche formale Komponente in der Instanz die Entität oder Relation gebunden werden kann. Stellvertretend für Regeln und Benutzerkommandos werden nur Regeln betrachtet.

### 5.2.5.1 Erzeugen einer Entität

Wird eine Entität erzeugt, so wird versucht, diese an alle Regelinstanzen der betroffenen Regeln zu binden. Dazu wird der Typ der Entität ermittelt und alle Rule-Declarations-Pairs des Typs aus dem Link_Container ausgelesen. Für jedes dieser Paare wird die Regel ermittelt und damit auf den Index im Regelcache eins zugegriffen. Für jede dieser Regelinstanzen wird die Liste der Declarations durchlaufen, an deren Stelle die Entität gebunden werden kann. Hier müssen zwei Fälle unterschieden werden:

- Die Regelinstanz hat für die Declaration noch keine Entität gebunden. Die Regelinstanz wird kopiert und die Entität an die Kopie gebunden. Die Anzahl der benötigten Entitäten wird um

eins heruntergesetzt. Die Originalinstanz kann gelöscht werden, da die Kopie alle bisherigen Bindungen enthält und somit die Kopie das Original ersetzt.

- Die Regelinstanz hat für die Declaration bereits eine Entität gebunden. Die Regelinstanz wird kopiert und die aktuelle Entität für die bereits gebundene eingesetzt. Das Attribut das anzeigt, aufgrund welcher Declaration die Instanz kopiert wurde, wird auf die aktuelle Declaration gesetzt.

Ist die Anzahl der benötigten Entitäten einer der kopierten Instanzen gleich null, so kann eine Kopie der Instanz im Cache zwei angelegt werden.

Der genaue Ablauf wird nochmal in dem folgenden Pseudocode dargestellt: (RI bezeichnet die Regelinstanz)

```
Free_Declaration := false;
Rule_Decl_Pair_List := Get_Rule_Pairs(Entity.Type);
for all Rule_Decl_Pair_List loop
  for all RIs in rulecache_1(Rule_Decl_Pair.Rule) loop
    for all Declarations in Rule_Decl_Pair.Declaration_List loop
      Copy_RI := copy(current_RI);
      if get_Variable(Declaration, RI) = null then
        bind_entity(Entity, Copy_RI, Declaration);
        Free_Declaration := true;
      else
        bind_entity(Entity, Copy_RI, Declaration);
        Copy_RI.copied_declaration := Declaration;
      end if;
    end loop;
    if Free_Declaration then
      Delete_Instance(TRI);
    end if;
  end loop;
end loop;
```

#### 5.2.5.2 Löschen einer Entität

Wird eine Entität gelöscht, so muss ermittelt werden, ob durch die Entität in der Instanz eine andere Entität ersetzt wurde, also, ob zwei Regelinstanzen mit unterschiedlichen Entitäten für die entsprechende formale Entität existieren. Dies kann durch das Attribut Copied_Declaration der Instanz festgestellt werden. Ist die Declaration, an deren Position die Entität gebunden ist gleich der Copied_Declaration, so kann die Instanz gelöscht werden. Ist die Declaration ungleich der Copied_Declaration, so darf die Entität nur aus der Instanz gelöst werden, da sonst Varianten, der Regelinstanzen gelöscht würden. Wird eine Entität aus einer Instanz herausgelöst, muss die Anzahl der benötigten Entitäten um eins erhöht werden.

Regelinstanzen in Regelcache zwei müssen auf jeden Fall gelöscht werden, falls sie die zu löschende Entität enthalten.

Das folgende Schaubild soll verdeutlichen, warum nur Regelinstanzen in Regelcache eins gelöscht werden dürfen, wenn die Declaration, an deren Stelle die Entität gebunden ist, mit der Copied_Declaration übereinstimmt. Es zeigt, welche Regelinstanzen kopiert werden, wenn neue Entitäten angelegt werden. Die Pfeile sind dabei nicht Teil der Datenstruktur für die Regelinstanzen, sondern zeigen nur an, wie kopiert wurde. Aufgrund des Schaubildes wird es einsichtig, dass beim Löschen der Entität vom Typ “Dreieck” keine Regelinstanz gelöscht werden darf, da sonst

Regelinstanzen verloren gehen würden. Beim Löschen einer Entität vom Typ “Kreis” dagegen können die drei betroffenen Regelinstanzen gelöscht werden.

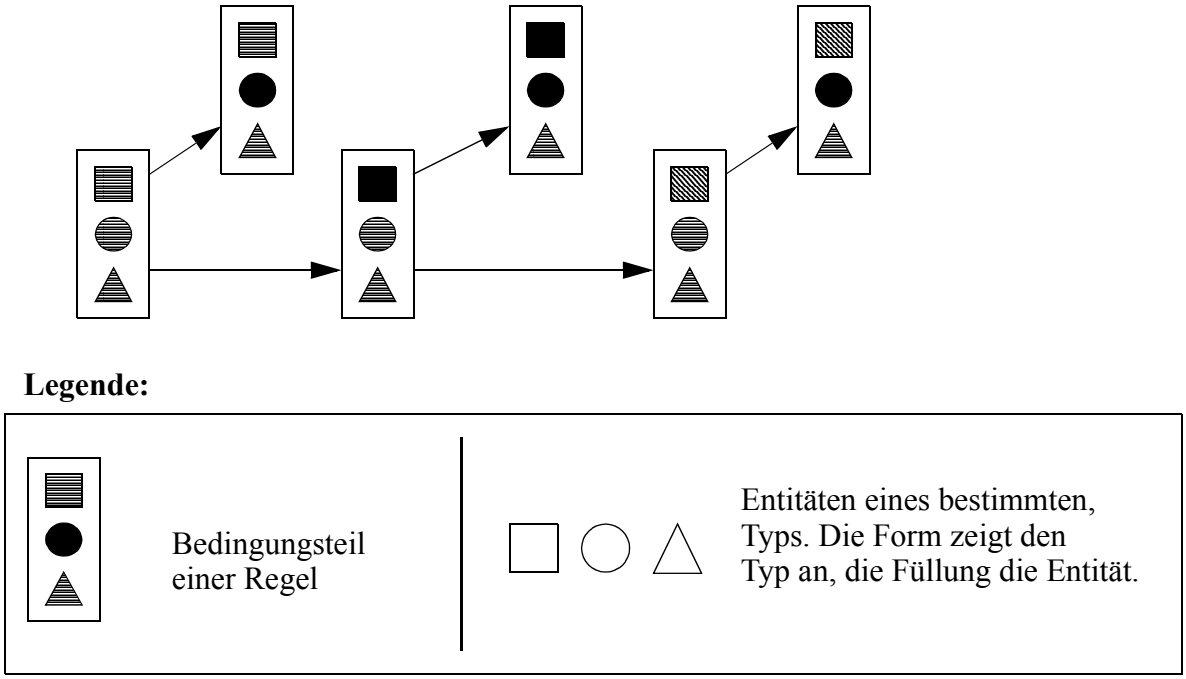



**Abbildung 20. Ableitung von Regelinstanzen**

#### 5.2.5.3 Erzeugen einer Relation

Wird eine Relation erzeugt, so wird versucht, diese an alle Instanzen der betroffenen Regel zu binden. Dazu wird der Typ der Relation ermittelt und alle Rule-Declarations-Pairs des Typs aus dem Link_Container ausgelesen. Für jedes dieser Paare wird die Regel ermittelt und damit auf den Index im Regelcache eins zugegriffen. Für jede dieser Regelinstanzen wird die Liste der Declarations durchlaufen, an deren Stelle die Relation gebunden werden kann.

Zuerst muss überprüft werden, ob der Typ der formalen Entitäten der Regel konform zu den Typen der an der Relation beteiligten Entitäten ist. Der Fall, dass dies nicht so ist, kann eintreten wenn die Entitätstypen, die an einem Relationstyp beteiligt sind, mehrere Subtypen haben. Der Relationstyp “Arbeitet” ist zum Beispiel mit den Entitätstypen “Person” und “Dokument” als Rollen definiert. Der Entitätstyp “Dokument” hat die beiden Subtypen “Spezifikation” und “Code”. Wird nun eine Relation von diesem Typ mit einer “Person” und einer “Spezifikation” erzeugt, so kann diese nur gebunden werden, falls der erwartete Entitätstyp für die zweite Rolle entweder “Dokument” oder “Spezifikation” lautet. Lautet er jedoch “Code”, darf sie nicht gebunden werden.

Sind alle formalen Entitäten zu den Typen der an der Relation beteiligten Entitäten konform, muss zwischen zwei Fällen unterschieden werden:

- Die Regelinstanz hat für die Declaration noch keine Relation gebunden. Die Regelinstanz wird kopiert und die Relation an die Kopie gebunden. Dabei wird jede an der Relation beteiligte Entität an die Instanz gebunden. Die Anzahl der benötigten Entitäten wird für jede gebundene Entität um eins heruntergesetzt. Wie beim Erzeugen einer Entität kann hier die Originalinstanz gelöscht werden.

- Die Regelinstanz hat für die Declaration bereits eine andere Relation gebunden. Die Regelinstanz wird kopiert und die aktuelle Relation für die bereits gebundene eingesetzt. Dabei wird jede an der Relation beteiligte Entität an die Instanz gebunden. War vorher an der Stelle noch keine Entität gebunden, so wird die Anzahl der benötigten Entitäten um eins heruntergesetzt. Das Attribut, das anzeigt, aufgrund welcher Declaration die Instanz kopiert wurde, wird auf die aktuelle Declaration gesetzt.

Aus Effizienzgründen wird in Kauf genommen, dass nicht korrekt gebundene Instanzen in Regelcache eins erzeugt werden können. Dies kann dann eintreten, wenn ein Konflikt zwischen formalen Relationen innerhalb einer Regel besteht, diese also auf die gleiche formale Entität verweisen.

Ist die Anzahl der benötigten Entitäten einer der kopierten Instanzen null, so muss, falls ein Konflikt besteht, noch geprüft werden, ob alle gebundenen Entitäten auch mit den gebundenen Relationen zusammenpassen. Passen sie zusammen oder besteht kein Konflikt, so kann eine Kopie der Instanz im Regelcache zwei angelegt werden.

#### 5.2.5.4 Löschen einer Relation

Das Löschen einer Relation verhält sich wie das Löschen einer Entität mit zwei Unterschieden:

- Soll die Relation aus der Instanz herausgelöst werden, müssen alle beteiligten Entitäten herausgelöst werden und für jede dieser Entitäten die Anzahl der benötigten Entitäten um eins erhöht werden.
- Besteht ein Konflikt zwischen formalen Relationen in einer Regel, so müssen nach dem Herauslösen einer Relation die verbleibenden Relationen repariert werden, da durch die Überschneidung zu viele Entitäten herausgelöst wurden.

### 5.2.6 Verwaltung der Caches

Die Verwaltung des Regelcache eins und des Kommandocache ist gleich. Deshalb wird hier nur auf den Regelcache näher eingegangen.

Regelcache eins enthält teilweise gebundene Regelinstanzen und vollständig gebundene Regelinstanzen. Sobald eine Regelinstanz alle benötigten Entitäten gebunden hat, wird diese in Regelcache zwei kopiert. Die Kopie der Regelinstanz wird an die Liste der Regelinstanzen mit gleicher Priorität angehängt. Wird bei einer vollständig gebundenen Regelinstanz eine Entität entfernt, so wird diese auf jeden Fall aus Regelcache zwei gelöscht, egal ob diese in Regelcache eins gelöscht oder nur die Entität entfernt wurde.

### 5.2.7 Regelauswertungszyklus

Während des Regelauswertungszyklus werden alle Regelinstanzen im Regelcache zwei betrachtet. Es wird von der höchsten Prioritätsstufe bis zur niedrigsten durchgegangen und versucht die Regelinstanzen einer Prioritätsstufe auszuführen. Da bereits alle benötigten Entitäten und Relationen gebunden sind, müssen nur noch die Attributsbedingungen überprüft werden. Sind alle Attributsbedingungen erfüllt, so kann der Aktionsteil der Regel ausgeführt werden. Wird eine Regel der Prioritätsstufe ausgeführt, so müssen alle Regelinstanzen der Prioritätsstufe nochmals betrachtet werden, da sich die Ergebnisse der Attributsbedingungen verändert haben könnten. Jede Regelinstanz, deren Aktionsteil ausgeführt wurde, wird als ausgeführt markiert, damit diese bei einem weiteren Durchgang ausgeschlossen werden kann. Die verbrauchte Zeit der ausgeführten Regelinstanzen wird addiert. Der folgende Pseudocode stellt den Ablauf nochmals dar (RI bezeichnet die RuleInstance):

```
T := 0;
for priority in priority_Max..priority_Min loop
  loop
    found_one := false;
    for RI in all RuleInstances(priority) loop
      if not executed(RI) then
        if all_constraints_ok(RI) then
          found_one := true;
          execute(RI);
          T := T + Get_Time(RI);
          RI.executed := true;
        end if;
      end if;
    end loop;
  until not found_one;
end loop;
```

### 5.2.8 Ausführen eines Benutzerkommandos

Wird vom Spieler initiiert, dass ein Benutzerkommando ausgeführt werden soll, so wird versucht, die bestehenden teilweise gebundenen Kommandoinstanzen auszuführen. Dazu werden alle Instanzen des betroffenen Kommandos betrachtet und versucht, mit den übergebenen Entitäten eine komplette Kommandoinstanz zu erzeugen. Bevor diese ausgeführt werden kann, müssen noch die Attributsbedingungen geprüft werden. Sind diese erfüllt, so kann das Kommando ausgeführt werden.

### 5.2.9 Deterministisches Verhalten

Es muss auch weiterhin gelten, dass bei gleichem Zustand das gleiche Resultat erzeugt wird. Solange der Aufbau der Modelle unverändert bleibt, wird dies durch den stets gleichen Aufbau des Zustands sichergestellt. Die Reihenfolge der Container gleicht der Einlesereihenfolge des Compilers, so dass auch die Liste der abgeleiteten Typen bei identischen Modellen identisch ist. Die Entitäten und Relationen eines Typs werden weiterhin alphabetisch nach dem Namen sortiert.

### 5.2.10 Änderungen der Schnittstellen

Da alle veränderten Datentypen Teil von Datenkapseln sind, müssen die Interfaces der betroffenen Module um Zugriffsoperationen erweitert werden. Alle Module sind bereits in dem Kapitel “Datenstrukturen” aufgeführt. Für jedes Attribut, das die Datenstruktur erweitert, wird eine Zugriffsoperation hinzugefügt. Die Veränderungen beschränken sich aber auf Module innerhalb eines Teilsystems.

## 5.3 “Stateprotocolcache”

### 5.3.1 Überblick

Die Daten, die an die Stateprotocoldatei angehängt werden sollen, werden nicht direkt in die Datei geschrieben, sondern zuerst intern in einem Cache gespeichert. Erst wenn dieser voll ist, wird dessen Inhalt an die Datei angehängt. Wenn die Simulation beendet wird, sei es durch den Benutzer oder durch ein Fehler im Program, wird der Inhalt des Caches ebenfalls in die Datei geschrieben.

## 5.3.2 Verweis auf betroffene Entwurfsdokumente

Es sind keine bereits bestehenden Entwurfsdokumente durch diesen Entwurf betroffen.

## 5.3.3 Datenstrukturen

### 5.3.3.1 Cache

Bei den Ausgaben, die in die Stateprotocoldatei geschrieben werden, handelt es sich ausschließlich um Strings. Daher bietet es sich an, als Cache ebenfalls einen String zu verwenden, der dann als Ganzes sehr effizient in die Datei geschrieben werden kann. Die Stateprotocoldatei eines normalen Simulationslaufes ist mehrere Megabyte groß und es werden mehr als 100.000 Schreiboperationen ausgeführt. Pro Schreiboperantion werden durchschnittlich nur knapp 100 Byte in die Datei geschrieben.

Die Anzahl der Schreiboperationen in die Datei sollte so gering wie möglich gehalten werden. Als Anhaltspunkt wird hier die durchschnittliche Anzahl der Regelauswertungszyklen während einer Simulation gewählt, da hier sehr viele Veränderungen vorgenommen werden. Nimmt man dieses an, so sollte circa 500 mal während einer Simulation der Cache geschrieben werden. Bei einer Größe der Protocoldatei von 20 MB macht das eine Größe des Caches von: 20MB / 500 = **circa 40 KB.**

Es ist sinnvoll, die Größe des Caches flexibel zu halten und es muss bei Messungen entschieden werden, welche Größe als sinnvoll erscheint. Deshalb ist es notwendig diese Größe sehr einfach veränderbar zu halten. Dies geschieht durch die Verwendung einer Konstante, die die Größe des Caches beschreibt.

## 5.3.4 Verwaltung des Caches

### 5.3.4.1 Schreiboperationen

Bei jeder Schreiboperation, die bisher direkt in die Datei geschrieben wird, findet jetzt eine Überprüfung statt, ob der Cache die neuen Daten noch aufnehmen kann. Falls ja, werden die Daten in den Cache geschrieben. Falls nicht, wird der komplette Inhalt des Caches in die Stateprotocoldatei geschrieben, der Cache geleert und die Daten danach in den Cache eingefügt.

Der folgende Pseudocode stellt den Ablauf nochmals dar:

```
if Cache.Size > (Cache.Occupied + Data.Size) then
  Cache.Append(Data);
else
  Append_Protocol(Cache, File);
  Empty_Cache(Cache);
  Cache.Append(Data);
end if;
```

### 5.3.4.2 Ende der Simulation

Die Simulation wird entweder von dem Spieler, über die Oberfläche, beendet oder durch eine Exception vom SESAM-Simulator abgebrochen.

Bei beiden Fällen muss vor dem Beenden des Simulators der Inhalt des Caches in die Stateprotocoldatei geschrieben werden. Dadurch wird es notwendig, die Funktion, die den Inhalt des Caches schreibt, mit in das Interface des Moduls “Stateprotocol” aufzunehmen.

Um bei einem unerwarteten Abbruch des Simulators die Möglichkeit zu haben, den Cache noch zu speichern, werden die Exceptions an der Schnittstelle zwischen der Tcl/Tk-Oberfläche und dem Adateil behandelt, da hier alle an die Basismaschine möglichen Aufrufe enthalten sind.

### 5.3.5 Änderungen des Interface

Das Interface des Moduls “Stateprotocol” muss um eine Funktion erweitert werden, die es erlaubt, das Schreiben des Caches in die Stateprotocoldatei von außerhalb anzustoßen.

# Kapitel 6
# Umsetzung der Vorschläge

*Dieses Kapitel beschreibt die Implementierung in Produktqualität, sowie den Test der Veränderungen an der Basismaschine. Es wird genauer auf Probleme eingegangen, die bei der Implementierung aufgetreten sind und welche Auswirkungen diese auf den Entwurf hatten. Zusätzlich ist der Regressionstest enthalten.*

## 6.1 Implementierung

### 6.1.1 Vorgehen

Um mögliche Überschneidungen durch Wartung während der Diplomarbeit an der Basismaschine zu vermeiden, wurde im CVS-Repository ein neuer Branch angelegt. Somit besteht entweder die Möglichkeit nach Ende der Diplomarbeit die unterschiedlichen Entwicklungszweige zusammenzuführen oder als Varianten beizubehalten.

Die Entwicklungsumgebung war ein Windows-PC mit den Programmen:

- Das Werkzeug Understand for Ada (Version 1.4) wurde zur Analyse der Basismaschine eingesetzt. Es werden Informationen zu allen Entitäten des Codes angeboten, wie Definitionsstelle von Variablen, Callgraph von Funktionen und mehr.
- Der Texteditor Textpad (Version 4.5.0) wurde zur Entwicklung des Codes eingesetzt.
- Der eingesetzte Compiler unter Windows ist: Gnat for Windows (Version 3.14p).

Die Veränderungen wurden hauptsächlich unter der Windowsumgebung vorgenommen und erst nach genauer Prüfung des Codes auf dem Abteilungsrechner “Genf” zur Ausführung gebracht. Dieses Vorgehen schließt Trial and Error schon aus Zeitgründen aus.

Um Schwierigkeiten bei der Implementierung oder Fehler des Entwurfs möglichst früh zu entdecken, wurden die Veränderungen am Code grob von den schwierigsten zu den leichtesten geordnet und in dieser Reihenfolge durchgeführt.

### 6.1.2 Probleme

#### 6.1.2.1 Fehler des Entwurfs

In dem Entwurf für den Vorschlag “Zwei Caches” war ursprünglich geplant, das Konzept der IC_Statements beizubehalten, um die Veränderungen am Code möglichst gering zu halten. Für jeden Entitäts- und Relationstyp ist bekannt, welche IC_Statements in welcher Regel oder welchem Kommando betroffen sind. Dadurch kann dann bei Veränderungen am Zustand die Menge der Instanzen berechnet werden. In einem Spezialfall tritt jedoch das Problem auf, dass zu wenige Instanzen berechnet werden. Dies liegt an der Umsetzung der Relationen durch die IC_Statements. Eine Relation, mit n Rollen, wird aufgeteilt in n-1 “Bind_Ent_Over_Rel_Stmts” und optional ein “Bind_Ent_Stmt”. Um eine Relation binden zu können, muss eine der, an der Relation beteiligten, Entitäten bereits an die Regel gebunden sein. Die bereits gebundene Entität kann entweder durch ein “Bind_Ent_Stmt” ohne weitere Vorraussetzungen bereits gebunden sein oder durch ein “Bind_Ent_Over_Rel_Stmt” als Teil einer Relation. Bei Regeln, in denen sich zwei formale Relationen überlappen, sie verweisen beide auf die gleiche formale Entität, kann dies zu Problemen führen. Es soll zum Beispiel die Relation R1 gebunden werden, die eine Entität benötigt, die durch

ein “Bind_Ent_Over_Rel_Stmt” der Relation R2 gebunden wird. Die Relation R2 existiert allerdings noch nicht. Da nur beim Erzeugen der Relation die, durch sie verursachten, Veränderungen an der Menge der Instanzen berechnet werden, wird die Relation auch später nicht mehr gebunden.

Bisher hat dies kein Problem dargestellt, da während jedem Zyklus versucht wurde alle Relationen zu binden.

Die beiden Module, die für die Verbindungen und die Berechnung der Instanzen zuständig sind, mussten somit komplett neu durchdacht und auch neu implementiert werden. Dies hat zu zeitlichen Problemen geführt, welche allerdings durch vermehrten Arbeitseinsatz und die eingeplanten Puffer aufgefangen wurden. Die restlichen, bereits implementierten Module, wie die Caches, konnten ohne größere Veränderungen übernommen werden.

In dieser Diplomarbeit ist der angepasste Entwurf enthalten.

### 6.1.3 Bewertung

Die Implementierung der Veränderungen an der Basismaschine kann als erfolgreich bezeichnet werden. Es wurden alle geplanten Veränderungen unter Einhaltung der Anforderungen durchgeführt. Die Unit- und Regressionstests zeigen keine Fehler.

Bei der Implementierung wurde darauf geachtet, die von der Abteilung vorgegebenen Programmierrichtlinien [Drappa, 1997] einzuhalten, um späteren Entwicklern die Einarbeitung nicht unnötig zu erschweren. Als weitere Hilfestellung wurden zum einen viele Kommentare in den Code eingefügt, die die Funktionsweise von komplexen Codeabschnitten erläutern und zum anderen wurden zusätzliche Funktionen erstellt, die die Ausgabe von internen Informationen erlauben. Einen Überblick über die Metriken des Codes gibt Kapitel 8.1.2.

## 6.2 Unit-Test

### 6.2.1 Testplan

Der Test ist eine Mischung aus Unit- und Integrations-Test. Es werden keine Stubs verwendet, sondern die veränderten und die neuen Module zusammengefügt und diese nacheinander getestet. Es wird jedoch nur der veränderte Ausschnitt des Systems betrachtet. Das komplette System wird während des Regressions-Test untersucht.

Die Unit-Tests beschränken sich auf Code, der komplett neu entwickelt wurde oder sehr starke Veränderungen erfahren hat. Bereits bestehende Module, die nur geringen internen Veränderungen unterworfen sind, werden durch bereits bestehende Tests im Regressions-Test behandelt.

Die Tests sind nach Modulen gegliedert aufgelistet und nicht nach Veränderungen für die einzelnen Vorschläge, da einzelne Veränderungen am Code für mehrere Vorschläge gültig sind. Soweit nicht anders gekennzeichnet, bauen die Tests aufeinander auf und verwenden die erzeugten Ergebnisse des vorangegangenen Tests.

Für alle Testfälle wurden Sollresultate definiert, die durch den Testtreiber automatisch geprüft werden. Die Ergebnisse werden an der Standardausgabe ausgegeben.

Die folgende Tabelle enthält die Tests der einzelnen Module und deren Beschreibung.

| Test | Beschreibung |
|---|---|
| Anlegen von Entitätstypen | Es werden fünf Entitätstypen angelegt, die in einer Vererbungsbeziehung zueinander stehen.<br>Die folgenden Eigenschaften werden geprüft:<br>• Sind die eindeutigen ID's der Entitätstypen korrekt erzeugt worden? Der erste erzeugte Typ muss die ID = 1 besitzen, alle darauffolgenden eine ID, die um eins erhöht ist.<br>• Werden die Listen der abgeleiteten Typen korrekt erzeugt? |
| Anlegen von Relationstypen | Es werden fünf Relationstypen angelegt, die in einer Vererbungsbeziehung zueinander stehen.<br>Die folgenden Eigenschaften werden geprüft:<br>• Sind die eindeutigen ID's der Relationstypen korrekt erzeugt worden? Der erste erzeugte Typ muss die ID = 1 besitzen, alle darauffolgenden eine ID, die um eins erhöht ist.<br>• Werden die Listen der abgeleiteten Typen korrekt erzeugt? |
| Anlegen von Regeln | Es werden drei Regeln mit formalen Entitäten und Relationen in den Strukturteilen, den angegebenen Prioritäten und Zeiten erzeugt.<br>Es werden die folgenden Eigenschaften geprüft:<br>• Sind die eindeutigen ID's der Regeln korrekt erzeugt worden? Die erste erzeugte Regel muss die ID = 1 besitzen, alle darauffolgenden eine ID, die jeweils um eins erhöht ist. |
| Anlegen von Benutzerkommandos | Es werden drei Kommandos mit formalen Entitäten und Relationen in den Strukturteilen und Zeiten erzeugt.<br>Es werden die folgenden Eigenschaften geprüft:<br>• Sind die eindeutigen ID's der Kommandos korrekt erzeugt worden? Das erste erzeugte Kommando muss die ID = 1 besitzen, alle darauffolgenden eine ID, die jeweils um eins erhöht ist. |
| Erzeugen von Instanzen | Es wird je eine Instanz der bestehenden Regeln und eine Instanz der bestehenden Kommandos erzeugt.<br>Es werden die folgenden Eigenschaften geprüft:<br>• Ist die Anzahl der benötigten Entitäten für die Instanz korrekt? |

**Tabelle 14. Unit-Tests**

| Test | Beschreibung |
|---|---|
| Initialisierung der Caches | Der Partinstance_Cache wird initialisiert und die erzeugten Caches ausgelesen. Dieser Test wird zweimal durchgeführt, um zu zeigen, dass die Initialisierung auch bei bereits belegtem Partinstance_Cache korrekt durchgeführt wird.<br>Es werden die folgenden Eigenschaften geprüft:<br>• Ist die Anzahl der erzeugten Instanzen in den Caches korrekt? In Regel-Cache-Eins und Kommando-Cache sollte jeweils eine Instanz von jeder Regel und jedem Kommando existieren. In Regel-Cache-Zwei sollten keine Regelinstanzen existieren, der Cache sollte aber initialisiert sein.<br>• Sind die Instanzen der Regeln in den korrekten Elementen des Caches eingetragen? Der Index des Elementes des Caches der die Instanz enthält muss mit der ID der Regel oder des Kommandos übereinstimmen, von dem die Instanz erzeugt wurde. |
| Initialisierung der Link_Container | Der Link_Container wird initialisiert und die erzeugten Container ausgelesen. Dieser Test wird zweimal durchgeführt, um zu zeigen, dass die Initialisierung auch bei bereits belegtem Link_Container korrekt durchgeführt wird.<br>Es werden die folgenden Eigenschaften geprüft:<br>• Ist die Länge der Rule / Command-Declarations-Pairs-Listen korrekt? Diese entspricht der Anzahl der Regeln / Kommandos, an die eine Entität oder Relation von einem bestimmten Typ gebunden werden kann.<br>• Sind die richtigen Regeln und Kommandos in der Liste enthalten?<br>• Stimmen die Declarations, an deren Position die Entitäten / Relationen an einer Instanz der Regel / Kommandos gebunden werden sollen? Die länge der Listen für die einzelnen Typen werden geprüft und die angegebenen Regeln / Kommandos müssen enthalten sein. Für jede Regel oder jedes Kommando müssen die Declarations stimmen. Dies wird durch die Position geprüft, die eine Declaration in einer Regel einnimmt. |

**Tabelle 14. Unit-Tests**

| Test | Beschreibung |
|---|---|
| Berechnung der Instanzen | Es werden mehrere Entitäten und Relationen erzeugt und wieder gelöscht. Dabei wird geprüft, welche Instanzen im Partinstance_Cache existieren. Dieser Test wird ebenfalls zweimal durchgeführt.<br>Es wird das Stateprotocol in die Datei “testprot.txt” geschrieben.<br>Es werden die folgenden Eigenschaften geprüft. Die Eigenschaften werden nur anhand von Regelinstanzen geprüft, da die Behandlung von Kommandos dadurch auch mit abgedeckt ist:<br>• Ist die richtige Anzahl an Instanzen vorhanden?<br>• Sind alle Entitäten und Relationen korrekt an die Instanzen gebunden?<br>Es wird geprüft, ob die richtigen Regel- und Kommandoinstanzen im Partinstance_Cache enthalten sind.<br>---> Hier wird der Testfall “Regelauswertungszyklus durchführen“ ausgeführt.<br>Bei einer Regel besteht ein Konflikt zwischen zwei Relationen. Nach dem Löschen einer der Relationen müssen die Instanzen noch alle benötigten Entitäten gebunden haben.<br>Nach dem Löschen der restlichen Entitäten und Relationen darf in Regel-Cache-Zwei keine Instanz mehr existieren und in Regel-Cache-Eins keine der Instanzen mehr eine Entität oder Relation gebunden haben. |
| Regelauswertungs-zyklus durchführen | Es wird ein Regelauswertungszyklus und es werden alle Kommandos einmal ausgeführt.<br>Es werden die folgenden Eigenschaften geprüft:<br>• Die verbrauchte Zeit nach dem Regelauswertungszyklus muss korrekt sein.<br>• Die verbrauchte Zeit nach den ausgeführten Kommandos muss korrekt sein. |
| Das Stateprotocol ausgeben | Das Stateprotocol wird während dem Test “Berechnung der Instanzen” in die Datei “testprot.txt” geschrieben. Diese muss von Hand geprüft werden. |

**Tabelle 14. Unit-Tests**

### 6.2.2 Testergebnisse

Keiner der durchgeführten Unit-Tests war erfolgreich, es traten also während des Tests keine Fehler auf.

## 6.3 Regressionstest

Der Regressionstest findet in mehreren Stufen und auf verschiedenen Ebenen statt. Die erste Stufe stellt einen Unit-Test dar, der die einzelnen Pakete der Basismaschine testet. Tests zwei und drei setzen Modelle ein, die in die komplett übersetzte Basismaschine geladen und durch Skripte ausgeführt werden.

Für jeden Testlauf wird eine Datei erzeugt, die die Testergebnisse enthält. Die Ergebnisse vor den Veränderungen erhalten die Endung ‘.soll’, die Ergebnisse nach den Veränderungen die Endung ‘.ist’.

### 6.3.1 Veränderungen an der Basismaschine für den Test

Um besser vergleichbare Ergebnisse zu bekommen, wird der Zufallsgenerator der Basismaschine abgeschaltet. Der Zufallsgenerator wird normalerweise verwendet, um Spielverläufe weniger vorhersehbar und damit interessanter für den Spieler zu gestalten.

### 6.3.2 Unit-Test (Whitebox Test)

Bereits bestehende Unit-Tests für Pakete der Basismaschine werden ausgeführt und die Ergebnisse mit den Sollergebnissen verglichen.

In der folgenden Tabelle werden die getesteten Pakete und Art der Tests vorgestellt.

| **Paket** | **Tests** |
|---|---|
| execute | • Es werden die verschiedenen Pakete des Teilsystem Execute getestet. Die Startsituation eines Modells wird ausgeführt. Außerdem werden Regeln und Benutzerkommandos angelegt und ausgeführt. |
| global<br>- glist<br>- globaltypes<br>- random | • Das Paket Global_Types wird getestet.<br>• Das Paket Glist wird getestet.<br>• Das Paket Random wird getestet. |
| message | • Test der Pakete zur Behandlung von User-, Tutor- und Debug-Messages. Es werden Nachrichten erzeugt, zu Strings umgewandelt und in die Nachrichtenliste eingefügt und gelöscht. |
| state | • Das Paket State_Manager wird getestet. Es werden Enitäten und Relation erzeugt und gelöscht und alle möglichen Operationen auf diesen ausgeführt.<br>• Die Pakete des Value_Manager werden getestet. Es werden Variablen von den Typen Integer, Real, Boolean, Date, String, Record, List, Entity und Relation erzeugt und jeweils alle möglichen Operationen ausgeführt. |
| symboltabelle | • Es werden die verschiedenen Pakete der Symboltabelle getestet. |

**Tabelle 15. Unit-Tests**

### 6.3.3 Blackbox-Test Funktionalität

Dieser Test prüft die Funktionalität der Basismaschine, indem Modelle erstellt werden, die dann durch Skripte und das Tool Basetest ausgeführt werden. Die Ergebnisse werden als Spielverlaufsprotokoll gespeichert. Mit den folgenden Modellen soll versucht werden, die komplette Funktionalität abzudecken.

#### 6.3.3.1 Datentypen

| Modell | Tests |
|---|---|
| booleanTest | Führt alle möglichen Aktionen auf Variablen vom Typ Boolean durch. |
| dateTest | Führt alle möglichen Aktionen auf Variablen vom Typ Date durch. |
| integerTest | Führt alle möglichen Aktionen auf Variablen vom Typ Integer durch. |
| listTest | Erzeugt Listen von den möglichen Typen Boolean, Date, Integer, Real, String, Record und führt Listenoperationen auf diesen aus. |
| realTest | Führt alle möglichen Aktionen auf Variablen vom Typ Real durch. |
| recordTest | Führt alle möglichen Aktionen auf Variablen vom Typ Record durch. |
| sincosTest | Überprüft die mathematischen Funktionen sin, cos, ln, exp, random. |
| stringTest | Führt alle möglichen Aktionen auf Variablen vom Typ String durch. |

**Tabelle 16. Blackboxtest Funktionalität - Datentypen**

#### 6.3.3.2 Entitäten und Relationen

| Modell | Tests |
|---|---|
| entityTest | Legt zueinander konforme Entitäten an und zählt die Anzahl der Entitäten je Typ und die Anzahl aller zum Typ konformen Entitäten. |
| entityTest1 | Legt zueinander konforme Entitäten an und prüft die Konformität. |
| entityTest2 | Legt zueinander konforme Entitäten an und prüft, ob diese Member von anderen Entitäten sind. |
| relationTest | Legt zueinander konforme Relationen an und führt die folgenden Aktionen aus:<br>- Prüfen der Konformität.<br>- Prüfen der Membereigenschaft.<br>- Prüfen, ob eine Relation des Typs existiert.<br>- Prüfen, ob eine Relation eines konformen Typs existiert.<br>- Die Relationen eines Typs zählen.<br>- Die Relationen eines Typs und aller konformen Typen zählen. |

**Tabelle 17. Blackboxtest Funktionalität - Entitäten und Relationen**

#### 6.3.3.3 Regeln

| Modell | Tests |
|---|---|
| ruleTest | Definiert Regeln, die dann durch erzeugen einer Relation ausführbar werden. |
| ruleTest1 | Führt Regeln aus und zeigt die verbrauchte Zeit und das Datum an. |
| ruleTest2 | Definiert mehrere Regeln auf unterschiedlichen Prioritätsstufen, die Berechnungen ausführen, deren Ergebniss nach einem Regelauswertungszyklus ausgegeben wird. |
| ruleTest3 | Testet, ob konforme Relationen und Entitäten, die im Bedingungsteil der Regeln vorkommen, richtig behandelt werden. |

**Tabelle 18. Blackboxtest Funktionalität - Regeln**

| Modell | Tests |
|---|---|
| ruleTest4 | Testet die Constraints der Regeln durch Abfragen, ob Relationen existieren, Attributsbedingungen und Funktionsaufrufe. |
| ruleTest5 | Definiert mehrere Regeln auf unterschiedlichen Prioritätsebenen. Eine Regel auf einer mittleren Prioritätseben schafft die Vorraussetzung für Regeln auf Prioritätsebenen über, gleich und unter der aktuellen Regel. |

**Tabelle 18. Blackboxtest Funktionalität - Regeln**

## 6.3.4 Blackbox-Test Simulation

Wie bereits bei der Effizienzmessung werden die Spielverläufe für die beiden Modelle QSVA und QS aus der Diplomarbeit von [Kalajzic, 2001] für die dritte Stufe des Regressionstests verwendet.

Es wird das Commandprotocoll und das Stateprotocoll mitgeschrieben.

Nachdem der Regressionstest bei dem veränderten System durchgeführt ist, werden zuerst die kompletten Spielverläufe in der ".ist" Datei verglichen. Treten hier keine Veränderungen auf, so kann davon ausgegangen werden, dass die Funktionalität nicht verändert wurde. Treten Unterschiede auf, so muss mit Hilfe der ausgegebenen Stateprotokolle genauer untersucht werden, wo diese auftreten.

Um Veränderungen der Spielverläufe durch Überschneidungen in den Aktivitäten der Entwickler auszuschließen, wird noch ein weiterer Spielverlauf für das QSVA-Modell erzeugt. In diesem Spielverlauf hat jeder Entwickler zu jedem Zeitpunkt nur eine Tätigkeit und es wird nur durch eine Tätigkeit ein Dokument bearbeitet. Es wird zum Beispiel erst mit dem Review der Spezifikation begonnen, sobald alle Entwickler die Tätigkeit "Spezifikation schreiben" beendet haben.

### 6.3.4.1 Commandprotocoll

Die Spielverläufe für die Modelle QSVA und QS verwenden Tutorkommandos, die genauere Aussagen über den Zustand des Simulators zulassen.

## 6.3.5 Testergebnisse

Die Ergebnisse des Regressionstest werden in Dateien gespeichert und werden nicht in dieses Dokument übernommen. Hier werden stelltvertretend für den Test die erzeugten Dateien aufgelistet und das Ergebnis des jeweiligen Test beschrieben. Es werden nur Ergebnisse aufgelistet, die nicht vollständig mit dem Sollergebnis übereinstimmen.

Der Platzhalter '*' steht für die Endung 'ist' oder 'soll':

### 6.3.5.1 Unit-Test (Whitebox Test)

| Test | Ergebnis |
|---|---|
| state-unit.* | OK - Die Entitäten und Relationen werden in einer anderen Reihenfolge ausgegeben.<br>Da die Container, die die Typen und die dazugehörigen Entitäten oder Relationen enthalten, nicht mehr alphabetisch, nach dem Namen des Typs, sortiert sind, entsteht bei der Ausgabe aller Entitäten oder Relationen des State eine andere Reihenfolge. |

#### 6.3.5.2 Blackbox-Test Funktionalität

| Test | Ergebnis |
|---|---|
| ruletest2.* | OK - Die Reihenfolge der ausgeführten Regeln verändert sich innerhalb einer Prioritätsstufe. Dies tritt auf, da die Regeln nicht mehr nach ihrem Namen alphabetisch, sortiert ausgeführt werden. |
| ruletest4.* | OK - Die Reihenfolge der ausgeführten Regeln verändert sich innerhalb einer Prioritätsstufe. Dies tritt auf, da die Regeln nicht mehr nach ihrem Namen alphabetisch ,sortiert ausgeführt werden. |

#### 6.3.5.3 Blackbox-Test Simulation

| | |
|---|---|
| qsva_high_400_a_B_Opt.ist | OK - Geringe Abweichungen in den Entitätsattributen, hervorgerufen durch eine veränderte Reihenfolge der ausgeführten Regeln. Problem tritt auf bei Überschneidung von Aktivitäten an einem Dokument. |
| qs_high_400_a_B_Opt.ist | OK - Geringe Abweichungen in den Entitätsattributen, hervorgerufen durch eine veränderte Reihenfolge der ausgeführten Regeln. Problem tritt auf bei Überschneidung von Aktivitäten an einem Dokument. |

Die aufgetretenen Abweichungen werden anhand des Codes und des Gesamtaufwands am Ende der Simulation des QSVA-Modells gezeigt:

| Maßzahl | Wert Soll | Wert Ist | Abweichung % |
|---|---|---|---|
| Code LOC | 19147 | 19168 | 1,1 |
| Code AFP | 390,74 | 391,17 | 1,1 |
| Code Fehler pro KLOC | 7,05 | 6,89 | 2,3 |
| Gesamtaufwand | 31,4 | 31,26 | 0,5 |
| Gesamtkosten | 15779,3 | 15849,9 | 0,4 |

**Tabelle 19. Abweichungen während Blackboxtest - Simulation**

# Kapitel 7
# Ergebnisse

*Dieses Kapitel stellt die Ergebnisse der Diplomarbeit dar. Es werden Messungen vor und nach den Veränderungen gegenübergestellt und bewertet. Im letzten Abschnitt werden die Ergebnisse zusammengefasst.*

## 7.1 Einführung
Nachdem der Regressionstest gezeigt hat, dass nach den Veränderungen an der Basismaschine der SESAM-Simulator die an ihn gestellten Anforderungen noch erfüllt und dieselben Ergebnisse liefert, kommt der interessanteste Teil dieser Diplomarbeit. Durch Effizienzmessungen werden die Auswirkungen der Veränderungen auf die Laufzeit der Basismaschine gezeigt.

## 7.2 Beschreibung der Messung
Um die Effizienz der Variante mit dem Original der Basismaschine vergleichen zu können, wird die bereits durchgeführte Effizienzmessung wiederholt und die ermittelten Ergebnisse gegenübergestellt. Durch die Veränderungen am Code der Basismaschine ist es allerdings nicht möglich, die Messungen unverändert zu wiederholen.

Die größten Veränderungen betreffen dabei die Suche nach Instanzen, da zum einen die IC_Statements wegfallen und zum anderen die Suche nach Instanzen stark verändert ist.

### 7.2.1 Kennzahlen
Aufgrund der Veränderungen entfallen die folgenden Kennzahlen:

- Besuchte Knoten je Regelinstanzsuche.
- IC_Kommandos je Regel.
- Entitäten je Entitätstyp.
- Ausgeführte Instanzsuchen bei einer Regel.

Es kommen allerdings auch neue Kennzahlen hinzu:

- Anzahl der erzeugten und gelöschten Entitäten.
- Anzahl der erzeugten und gelöschten Relationen.
- Anzahl der Instanzen in den drei Caches.

### 7.2.2 Zeiten
Die gemessenen Zeiten bleiben unverändert. Nur die Art der Messung ändert sich bei den folgenden Zeiten:

- Suchen einer Regelinstanz:
  Da der Algorithmus zur Regelinstanzsuche nicht mehr verwendet wird, beschreibt diese Messung die Zeit, die aufgewendet wird, um die Veränderungen an der Menge der Instanzen zu berechnen, plus der Zeit zur Überprüfung der Attributsbedingungen.

### 7.2.3 Durchführung der Messung

Um mögliche Fehler durch Textausgaben und sonstige Berechnungen, die nicht Teil der eigentlichen Basismaschine sind, zu vermeiden, werden auch die Messungen der bereits betrachteten Modelle, vor den Veränderungen, wiederholt.

Bei den folgenden Modellen werden Messungen mit und ohne die Ausgabe der Stateprotocoldatei durchgeführt, jeweils vor und nach den Veränderungen an der Basismaschine:

- qs - qs_high_400_a.b2
  QS-Modell - beschrieben in Kapitel 2.
- qsva - qsva_high_400_a.b2
  QSVA-Modell - beschrieben in Kapitel 2. Allerdings ohne Auswirkungen der Motivation.
- motgut - motgut_medium.b2
  QSVA-Modell mit guter Motivation der Mitarbeiter.
- motneutral - motneutral_medium.b2
  QSVA-Modell mit neutraler Motivation der Mitarbeiter.
- motschlecht - motschlecht_medium.b2
  QSVA-Modell mit schlechter Motivation der Mitarbeiter.
- ts_regression - test_erweiterung200_regression.b2
  Feingranulare Variante des QS-Modells - beschrieben in Kapitel 2.

### 7.2.4 Messumgebung

Wie bei der ursprünglichen Effizienzmessung werden alle Messungen auf dem Abteilungsrechner “genf” der Abteilung Software-Engineering durchgeführt:

| | |
|---|---|
| Rechnersystem: | SUN Workstation Ultra-4 |
| Betriebssystem: | Solaris 2.7 |
| Entwicklungsumgebung: | Gnat 3.13p<br>ohne Compileroptimierung |

## 7.3 Auswertung der Ergebnisse

Das wichtigste Ergebnis der Effizienzmessungen ist, dass die durchgeführten Veränderungen zum größten Teil erfolgreich sind. Die Effizienzsteigerung soll zuerst anhand aller Regelauswertungszyklen gezeigt werden, da diese den Spieler aufgrund ihrer Dauer am meisten betreffen. Danach wird dargestellt, wie sich die Zeit auf die Regelsuche und die Regelausführung verteilt und welche Effizienzsteigerungen hier beobachtet werden. Am Ende wird noch auf die Ladezeit der Modelle und die Benutzerkommandos eingegangen, sowie auf interessante Kennzahlen.

### 7.3.1 Regelauswertungszyklus

Da, wie sich später zeigen wird, die Zeit zum Laden der Modelle und der Ausführung der Benutzerkommandos sich kaum verändert hat, kann die Zeitersparnis während der Regelauswertungszyklen mit der Gesamtersparnis gleichgesetzt werden. Die Zeiten bei den Modellen mit Schrittweite von einem Tag betragen zwischen 94 - 75 %, bei der feingranularen Variante von bis zu 45 % der Originalzeiten.

Die Folgende Tabelle und das Schaubild zeigen die gemessenen Zeiten für alle Regelauswertungszyklen und die prozentuale Verbesserung.

| Modell | original mit state (ms) | optimiert mit state (ms) | % | original ohne state (ms) | optimiert ohne state (ms) | % |
|---|---|---|---|---|---|---|
| **qs** | 1011860 | 950510 | 94 | 601450 | 555500 | 93 |
| **qsva** | 1108450 | 954500 | 86 | 678540 | 538870 | 80 |
| **motgut** | 1129630 | 1007600 | 89 | 710060 | 558670 | 79 |
| **motneutral** | 1140220 | 1021350 | 90 | 727490 | 569360 | 79 |
| **motschlecht** | 1260860 | 1103320 | 88 | 840020 | 626630 | 75 |
| **ts_regression** | 818070 | 530450 | 65 | 535020 | 245480 | 45 |

**Tabelle 20. Zeit für alle Regelauswertungszyklen**

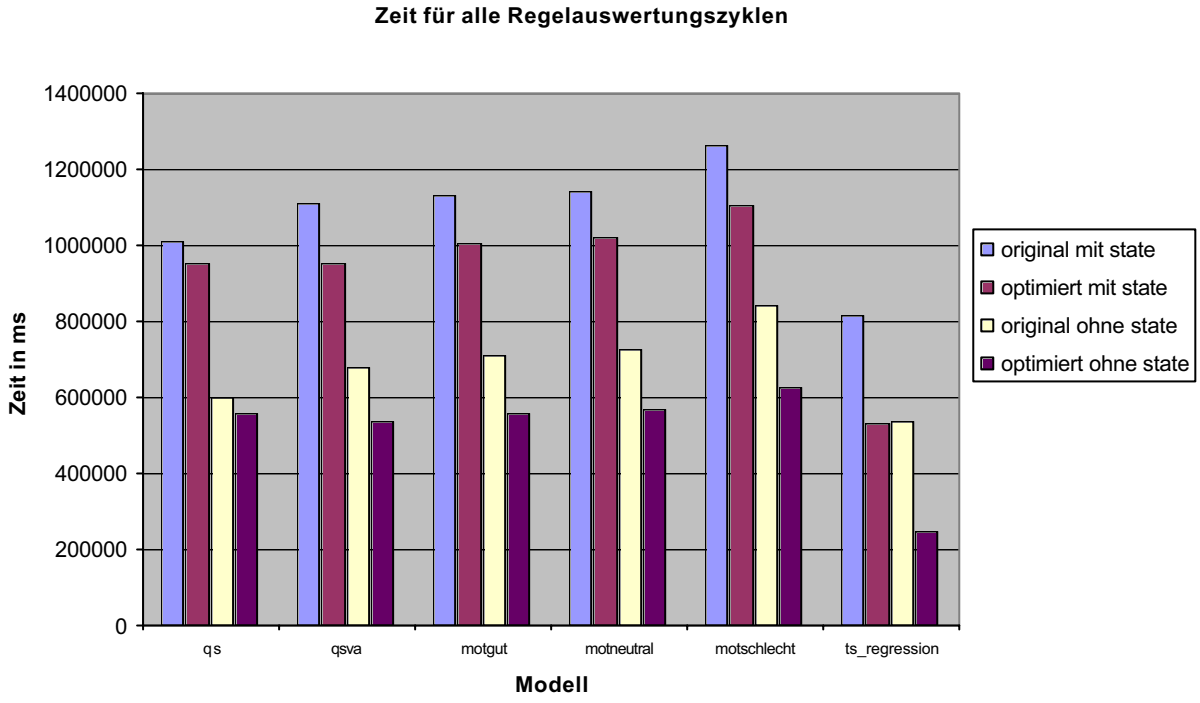



**Abbildung 21. Vergleich der Zeiten aller Regelauswertungszyklen**

Auffallend ist, dass der erste Regelauswertungszyklus nach dem Laden eines Modells deutlich mehr Zeit in Anspruch nimmt als zuvor (s. Tabelle 19). Dies wird durch die großen Veränderungen am Zustand der Basismaschine hervorgerufen. Beim Laden des Modells werden nur die von dem Modellbauer in der Startsituation definierten Entitäten und Relationen, wie die Entwickler, angelegt. Während des ersten Zyklus allerdings werden sehr viele Entitäten, wie die Dokumente, und Relationen erzeugt, die im Verlauf der Simulation benötigt werden. Somit müssen sehr viele Veränderungen an der Menge der Instanzen berechnet werden. Dies zahlt sich allerdings später aus, da diese Arbeit nicht wiederholt werden muss.

| Modell | original mit state (ms) | optimiert mit state (ms) | original ohne state (ms) | optimiert ohne state (ms) |
|---|---|---|---|---|
| **qs** | 6090 | 8560 | 1330 | 3720 |
| **qsva** | 6300 | 9020 | 1450 | 4210 |
| **motgut** | 6270 | 9190 | 1450 | 4240 |
| **motneutral** | 6250 | 9210 | 1450 | 4230 |
| **motschlecht** | 6250 | 9400 | 1450 | 4180 |
| **ts_regression** | 3150 | 3300 | 750 | 940 |

**Tabelle 21. Zeit für den ersten Zyklus**

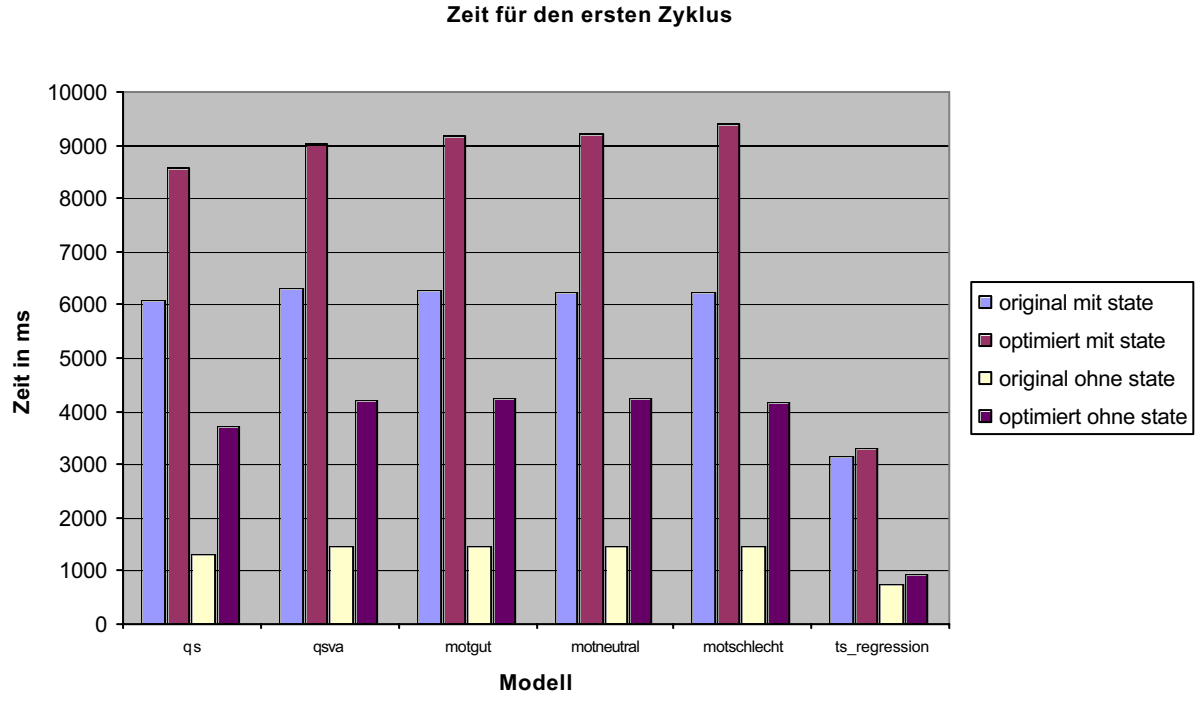



**Abbildung 22. Vergleich der Zeiten des ersten Regelauswertungszyklus**

Die Zeit, die für den Spieler hauptsächlich von Bedeutung ist, ist die durchschnittliche Dauer eines Regelauswertungszyklus. Die Folgende Tabelle zeigt die durchschnittliche Zeit eines Regelauswertungszyklus, ohne den ersten Zyklus einer Simulation und wieviel Prozent der Originalzeit benötigt wird.

| Modell | original mit state (ms) | optimiert mit state (ms) | % | original ohne state (ms) | optimiert ohne state (ms) | % |
|---|---|---|---|---|---|---|
| **qs** | 3363 | 3150 | 94 | 2007 | 1845 | 91 |
| **qsva** | 3601 | 3089 | 85 | 2212 | 1747 | 78 |
| **motgut** | 3612 | 3210 | 88 | 2278 | 1782 | 78 |
| **motneutral** | 3577 | 3192 | 89 | 2290 | 1782 | 77 |
| **motschlecht** | 3284 | 2863 | 87 | 2195 | 1629 | 74 |
| **ts_regression** | 691 | 443 | 64 | 453 | 205 | 45 |

**Tabelle 22. Durchschnittliche Zeit für einen Zyklus**

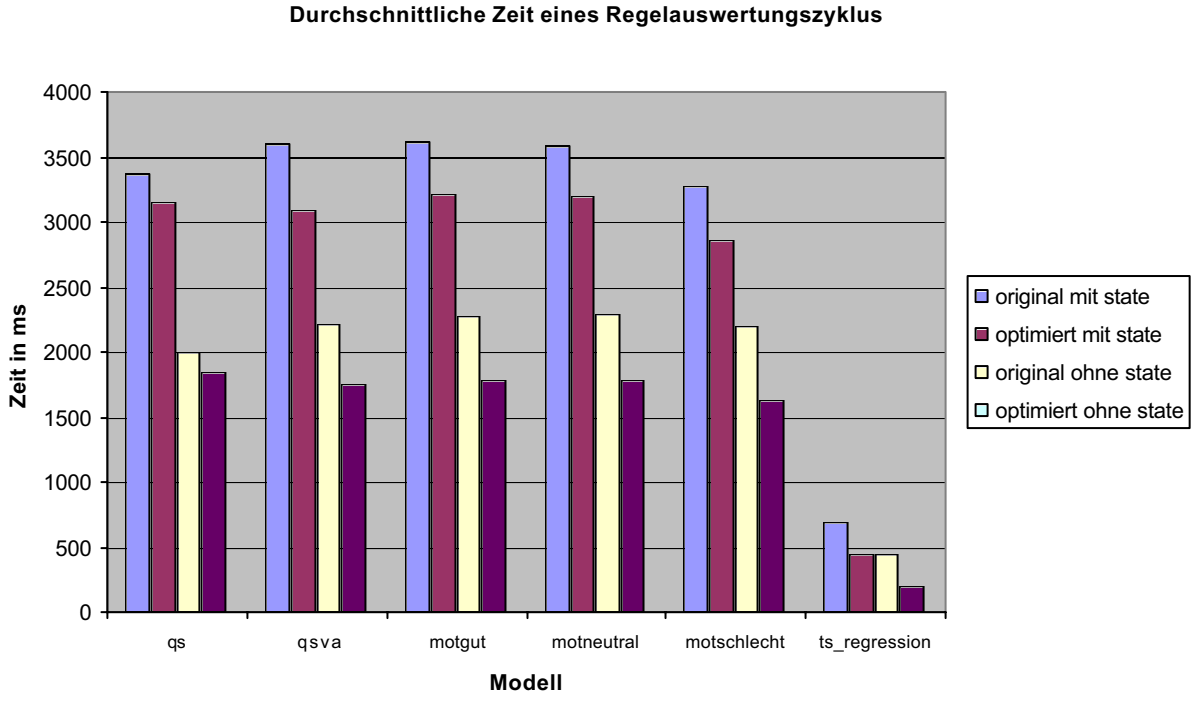



**Abbildung 23. Vergleich der durchschnittlichen Zeiten eines Regelauswertungszyklus**

## 7.3.2 Berechnung der Regelinstanzen

Die gesteigerte Effizienz wird hauptsächlich durch die Vorberechnung der Regelinstanzen erreicht. Insgesamt muss für die Suche nach Regelinstanzen nur noch zwischen 76 - 34 % bei den Modellen mit einem Zeitschritt von einem Tag aufgewendet werden, bei der Feingranularen Variante nur noch 17 %.

| Modell | original mit state (ms) | optimiert mit state (ms) | % | original ohne state (ms) | optimiert ohne state (ms) | % |
|---|---|---|---|---|---|---|
| **qs** | 148410 | 113470 | 76 | 147330 | 112010 | 76 |
| **qsva** | 206280 | 70280 | 34 | 207970 | 69960 | 34 |
| **motgut** | 217880 | 80940 | 37 | 221230 | 79300 | 36 |
| **motneutral** | 223000 | 82950 | 37 | 227410 | 80920 | 36 |
| **motschlecht** | 275870 | 92890 | 34 | 283830 | 90500 | 32 |
| **ts_regression** | 310740 | 55350 | 18 | 317240 | 54230 | 17 |

**Tabelle 23. Zeit für alle Regelinstanzsuchen**

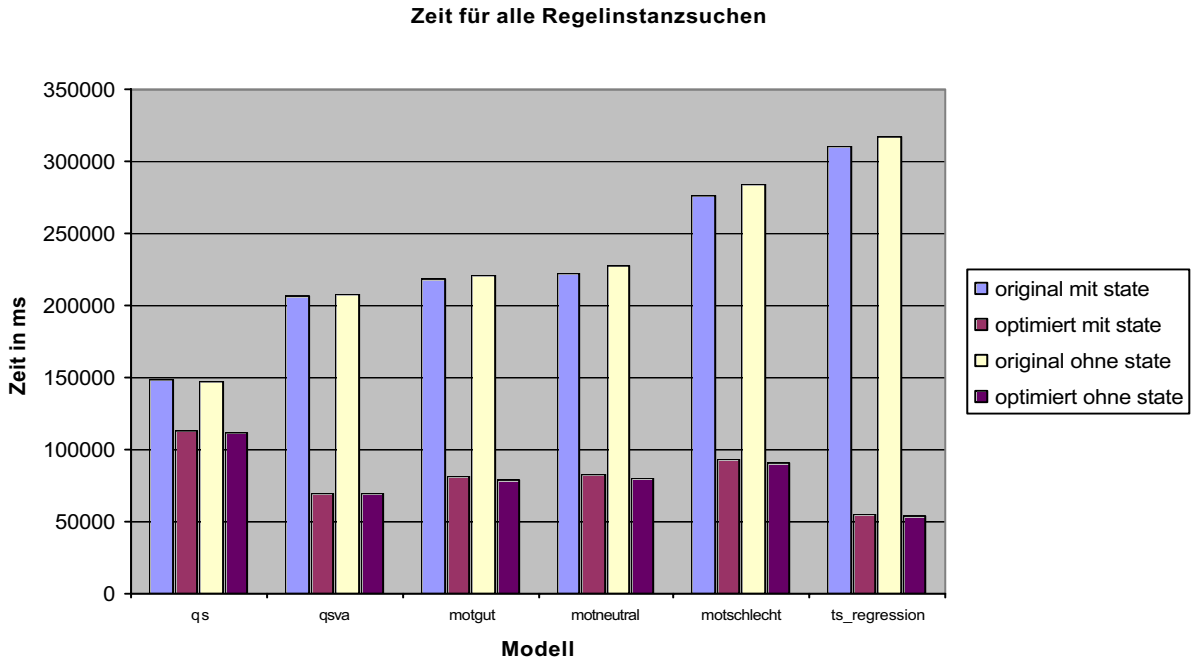



**Abbildung 24. Vergleich der Zeiten aller Regelinstanzsuchen**

Dies liegt zum einen daran, dass durchschnittlich nur noch 86 - 60 % beziehungsweise 43 % für die Suche nach einer Regelinstanz aufgewandt werden müssen. Zum anderen liegt dies aber auch daran, dass insgesamt weniger Regelinstanzen während der Regelauswertungszyklen überprüft werden müssen. Die beiden folgenden Tabellen und Schaubilder zeigen die durchschnittliche Zeit für die Suche nach einer Regelinstanz und die Anzahl der gesuchten Regelinstanzen.

| Modell | original mit state (ms) | optimiert mit state (ms) | % | original ohne state (ms) | optimiert ohne state (ms) | % |
|---|---|---|---|---|---|---|
| **qs** | 0,51 | 0,44 | 86 | 0,51 | 0,43 | 86 |

**Tabelle 24. Durschnittliche Zeit für Regelinstanzsuchen**

| | | | | | | |
|---|---|---|---|---|---|---|
| **qsva** | 0,48 | 0,28 | 59 | 0,48 | 0,28 | 59 |
| **motgut** | 0,50 | 0,31 | 62 | 0,51 | 0,31 | 60 |
| **motneutral** | 0,51 | 0,31 | 60 | 0,52 | 0,30 | 58 |
| **motschlecht** | 0,53 | 0,30 | 56 | 0,55 | 0,29 | 53 |
| **ts_regression** | 0,45 | 0,19 | 44 | 0,46 | 0,19 | 42 |

**Tabelle 24. Durschnittliche Zeit für Regelinstanzsuchen**

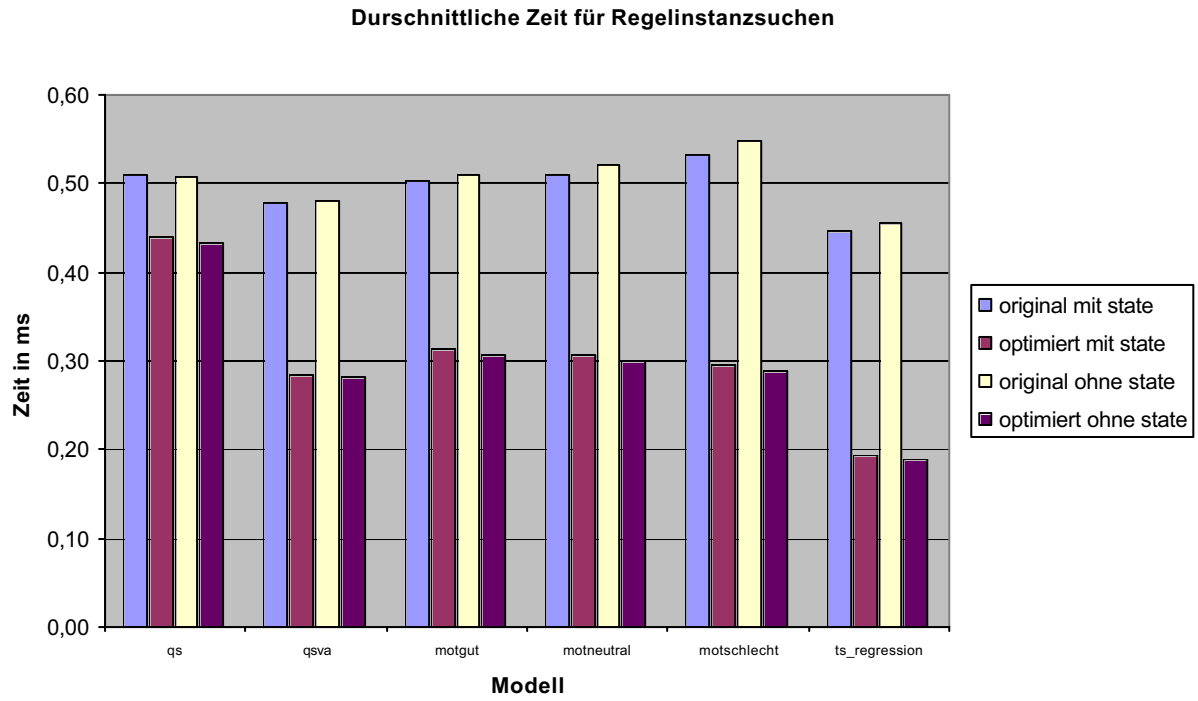



**Abbildung 25. Vergleich der durchschnittlichen Zeit der Regelinstanzsuchen**

| Modell | original mit state | optimiert mit state | % | original ohne state | optimiert ohne state | % |
|---|---|---|---|---|---|---|
| **qs** | 290735 | 258042 | 89 | 290735 | 258042 | 89 |
| **qsva** | 432027 | 247565 | 57 | 432027 | 247565 | 57 |
| **motgut** | 433279 | 258888 | 60 | 433279 | 258888 | 60 |
| **motneutral** | 437225 | 270281 | 62 | 437225 | 270281 | 62 |
| **motschlecht** | 518406 | 313825 | 61 | 518406 | 313825 | 61 |
| **ts_regression** | 696701 | 285218 | 41 | 696701 | 285218 | 41 |

**Tabelle 25. Anzahl der gesuchten Regelinstanzen**

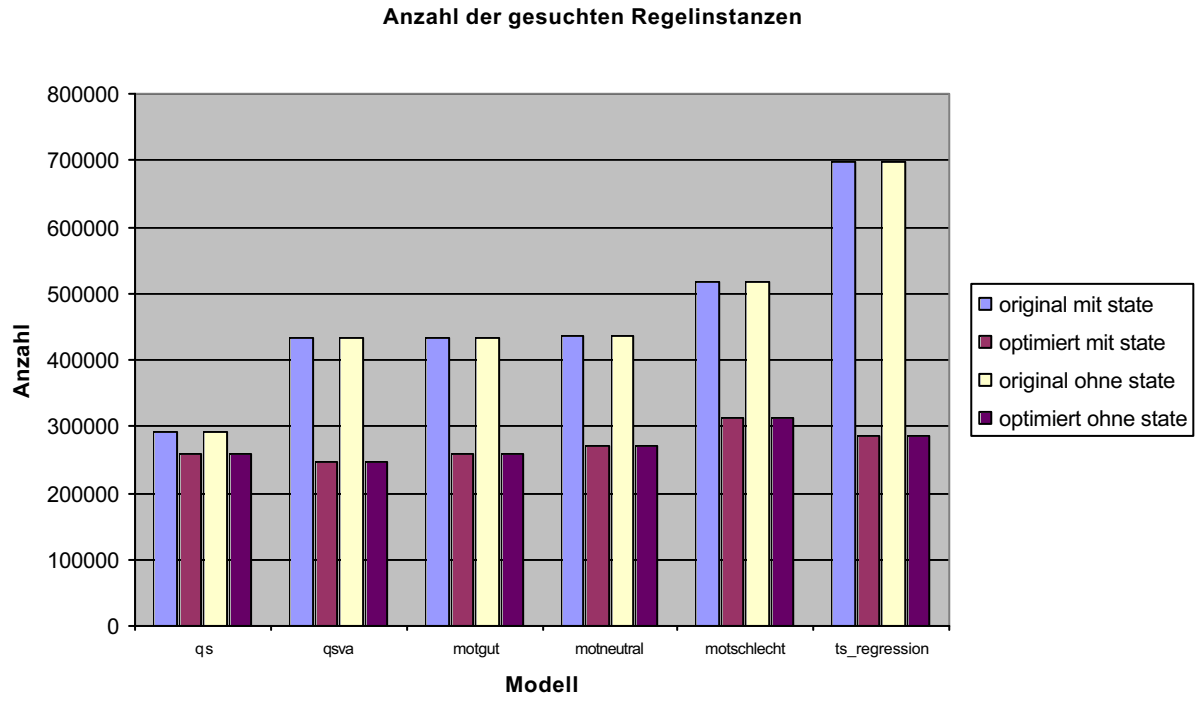



**Abbildung 26. Vergleich der Anzahl der gesuchten Regelinstanzen**

### 7.3.3 Ausführen der Regelinstanzen

Im Gegensatz zur Regelinstanzsuche erhöht sich die Zeit für die Regelausführung leicht. Dies war einerseits zu erwarten, da während der Ausführung die Veränderungen an der Menge der Instanzen berechnet werden. Andererseits fällt auf, dass bei der optimierten Version mit eingeschaltetem Stateprotocol keine Verbesserung erkennbar ist. Dies sollte jedoch durch die Umsetzung des Vorschlags “Stateprotocolcache” erreicht werden.

| Modell | original mit state (ms) | optimiert mit state (ms) | original ohne state (ms) | optimiert ohne state (ms) |
|---|---|---|---|---|
| **qs** | 849870 | 851590 | 440330 | 457770 |
| **qsva** | 881960 | 908850 | 451200 | 493710 |
| **motgut** | 891770 | 952550 | 468840 | 504290 |
| **motneutral** | 897790 | 962940 | 480060 | 512500 |
| **motschlecht** | 962590 | 1035990 | 532360 | 560180 |
| **ts_regression** | 482100 | 475640 | 191280 | 191800 |

**Tabelle 26. Zeit für alle ausgeführten Regelinstanzen**

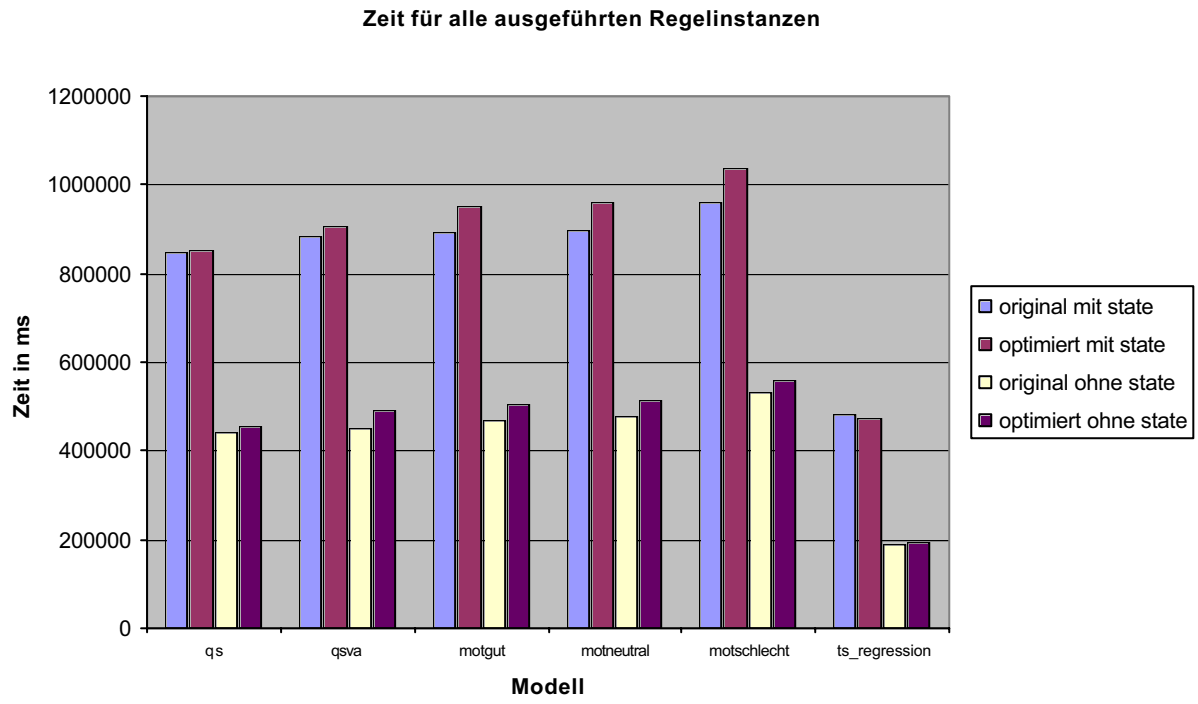



**Abbildung 27. Vergleich der Zeiten aller ausgeführten Regelinstanzen**

Wie bereits im Regressionstest bemerkt, ergeben sich geringe Änderungen der Ergebnisse einer Simulation durch die veränderte Ausführreihenfolge der Regeln. Daraus resultieren auch die leichten Schwankungen in der Anzahl der ausgeführten Regeln. Bei dem entzerrten Spielverlauf, der während des Regressionstest verwendet wurde, treten jedoch keine Veränderungen mehr auf.

| Modell | original mit state | optimiert mit state | original ohne state | optimiert ohne state |
|---|---|---|---|---|
| **qs** | 13107 | 13032 | 13107 | 13032 |
| **qsva** | 19425 | 19603 | 19425 | 19603 |
| **motgut** | 20588 | 20625 | 20588 | 20625 |
| **motneutral** | 21118 | 21271 | 21118 | 21271 |
| **motschlecht** | 26649 | 26014 | 26649 | 26014 |
| **ts_regression** | 46504 | 46305 | 46504 | 46305 |

**Tabelle 27. Anzahl der ausgeführten Regelinstanzen**

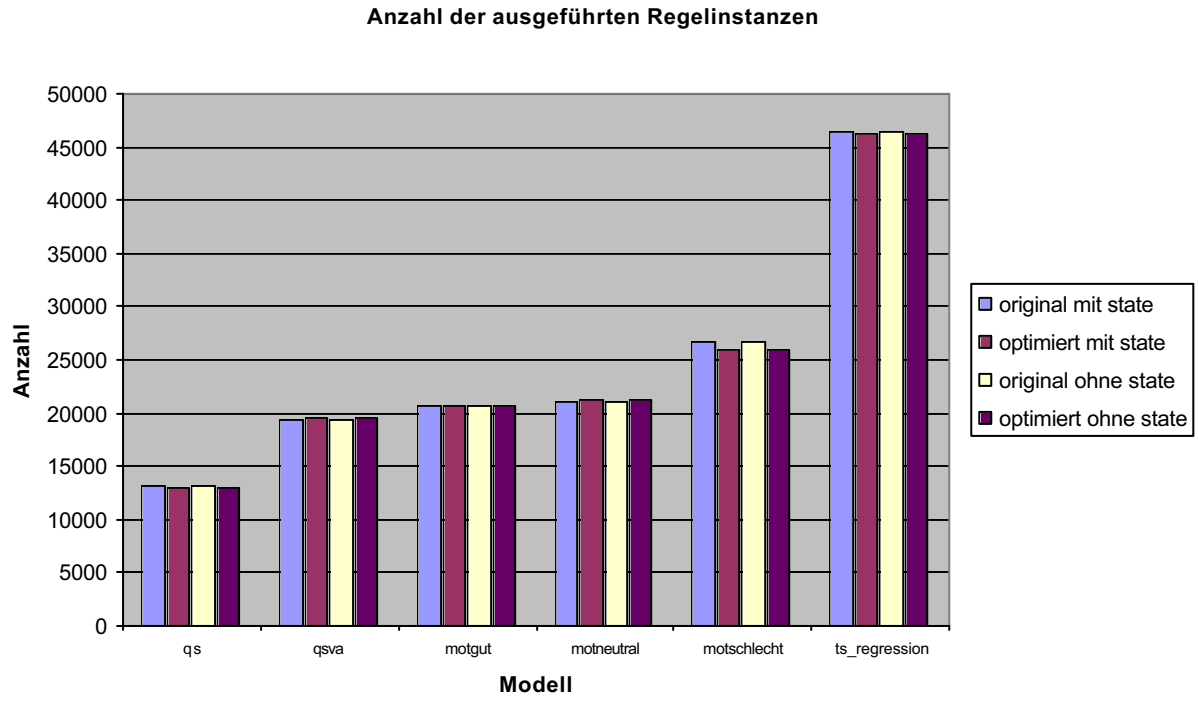



**Abbildung 28. Vergleich der Anzahl ausgeführter Regelinstanzen**

### 7.3.4 Laden der Modelle

Die Ladezeit der Modelle ist weitestgehend konstant geblieben, obwohl durch die Veränderungen, Datenstrukturen zusätzlich initialisiert werden müssen. Da die IC_Statements allerdings nicht mehr berechnet werden müssen, wird der Mehraufwand ausgeglichen.

| Modell | original mit state (ms) | optimiert mit state (ms) | original ohne state (ms) | optimiert ohne state (ms) |
|---|---|---|---|---|
| **qs** | 47520 | 49310 | 47270 | 49160 |
| **qsva** | 59900 | 61000 | 59790 | 60760 |
| **motgut** | 60820 | 60850 | 59960 | 60990 |
| **motneutral** | 60860 | 61120 | 59730 | 60820 |
| **motschlecht** | 60570 | 62000 | 59710 | 60920 |
| **ts_regression** | 26810 | 26870 | 26250 | 26550 |

**Tabelle 28. Modell Ladezeit**

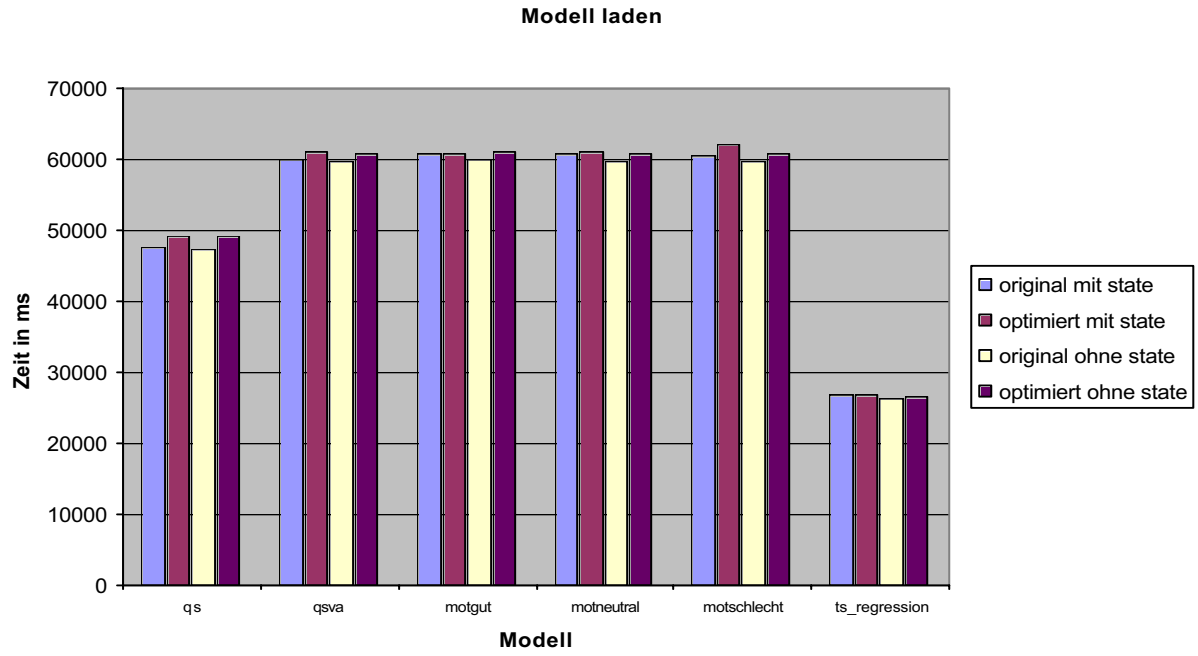



**Abbildung 29. Vergleich der Zeit zum Laden eines Modells**

### 7.3.5 Ausführen der Kommandos

Für die Ausführung der Benutzerkommandos haben sich, wie erwartet, kaum Veränderungen in der Dauer ergeben. Diese werden im Vergleich zu Regeln viel zu selten ausgeführt, als dass Vorberechnen der Instanzen einen Vorteil bringen würde.

| Modell | original mit state (ms) | optimiert mit state (ms) | original ohne state (ms) | optimiert ohne state (ms) |
|---|---|---|---|---|
| **qs** | 6050 | 6030 | 5650 | 5810 |
| **qsva** | 7200 | 7020 | 6900 | 6720 |
| **motgut** | 6030 | 6510 | 5710 | 5760 |
| **motneutral** | 5950 | 6330 | 5670 | 5750 |
| **motschlecht** | 6250 | 6660 | 6030 | 5890 |
| **ts_regression** | 420 | 390 | 170 | 160 |

**Tabelle 29. Zeit für alle Kommandos**

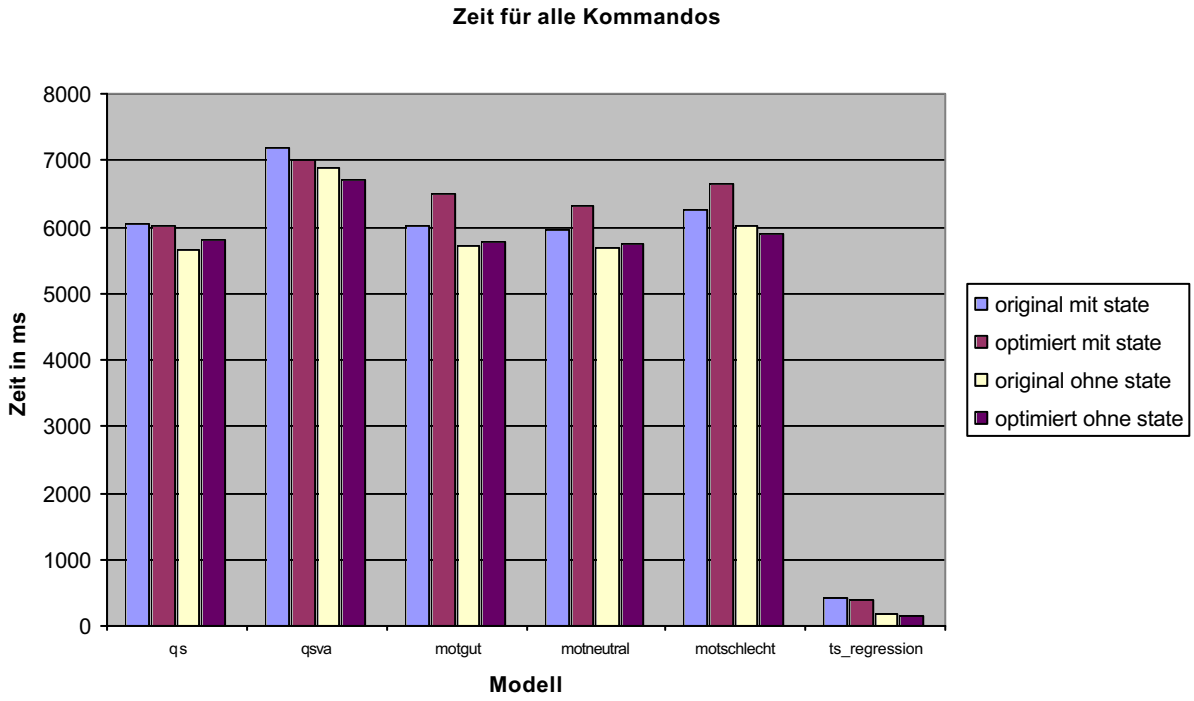



**Abbildung 30. Vergleich der Zeiten aller Benutzerkommandos**

### 7.3.6 Interessante Kennzahlen

Die folgenden Zahlen zeigen die Anzahl der Instanzen in den einzelnen Caches während der Simulation mit dem jeweiligen Modell. Zum besseren Vergleich werden auch die Anzahl der Regeln und Kommandos der Modelle angegeben. Es zeigt sich, dass sich die durchschnittliche Zahl der Instanzen je Regel in dem Regelcache eins zwischen 2 und 8 liegt, was von entscheidender Bedeutung für die Effizienz ist, da bei jeder betrachteten Regel alle Instanzen der Regel untersucht werden müssen.

Mit Ausnahme des QS-Modells liegt die durchschnittliche Anzahl der Instanzen in Regelcache 2 unter der Anzahl der Regeln in den Modellen. Dies zeigt, dass viele Regeln während des Regelauswertungszyklus nicht betrachtet werden müssen.

Von Benutzerkommandos existieren höchstens zwei Instanzen je Kommando in dem Kommandocache während einer Simulation.

| Modell | Regeln im Modell | Kommandos im Modell | Rule Cache One | Rule Cache Two | Command Cache |
|---|---|---|---|---|---|
| qs | 586 | 68 | min: 2877<br>max: 4918<br>avg: 4377 | min: 2<br>max: 1484<br>avg: 722 | min: 68<br>max: 76<br>avg: 72 |
| qsva | 692 | 79 | min: 1072<br>max: 3942<br>avg: 3030 | min: 23<br>max: 405<br>avg: 259 | min: 79<br>max: 143<br>avg: 138 |
| motgut | 692 | 79 | min: 1072<br>max: 3974<br>avg: 2923 | min: 23<br>max: 418<br>avg: 242 | min: 79<br>max: 143<br>avg: 137 |
| motneutral | 692 | 79 | min:1072<br>max: 3986<br>avg: 3087 | min: 23<br>max: 434<br>avg: 241 | min: 79<br>max: 143<br>avg: 139 |
| motschlecht | 692 | 79 | min: 1072<br>max: 3967<br>avg: 3096 | min: 23<br>max: 418<br>avg: 242 | min: 79<br>max:143<br>avg: 139 |
| ts_regression | 327 | 112 | min: 404<br>max: 2387<br>avg: 1773 | min: 2<br>max: 173<br>avg: 138 | min: 112<br>max:168<br>avg: 139 |

**Tabelle 30. Anzahl der Instanzen in den Caches**

## 7.4 Zusammenfassung

Dieses Kapitel fasst die Ergebnisse nochmals zusammen und legt dar, inwieweit die umgesetzten Vorschläge erfolgreich waren.

### 7.4.1 Vorschlag “Zwei Caches”

Durch den Vorschlag “Zwei Caches” sollte die Zeit verringert werden, die für die Suche nach Regelinstanzen aufgewendet werden muss. Dies geschieht zum einen dadurch, dass die gleichen Regelinstanzen nicht mehrmals hintereinander gesucht werden müssen, sondern nur einmal berechnet werden. Zum anderen müssen während eines Regelauswertungszyklus nur noch vollständig gebundene Regelinstanzen auf ihre Attributsbedingungen hin geprüft werden.

#### 7.4.1.1 Anzahl der Regelinstanzen

Die Zahl der während der Regelauswertungszyklen betrachteten Regelinstanzen hat sich deutlich verringert. Bei den Modellen mit einem Zeitschritt von einem Tag liegt die Anzahl bei 88 - 60 % der ursprünglichen Werte. Bei dem feingranularen Modell mit einem Zeitschritt von 2 Stunden sogar nur bei 40%, was sich durch die häufigere Ausführung des Regelauswertungszyklus und daraus resultierenden geringeren Veränderungen am Zustand erklären lässt. Dadurch müssen weniger Regelinstanzen zwischen zwei Regelauswertungszyklen neu berechnet werden.

#### 7.4.1.2 Zeit für die Regelinstanzsuche

Insgesamt muss für die Suche nach Regelinstanzen nur noch zwischen 76 - 34 % der ursprünglichen Zeit bei den Modellen mit einem Zeitschritt von einem Tag aufgewendet werden, bei der Feingranularen Variante sogar nur noch 17 %.

#### 7.4.1.3 Gesamtzeit

Es war zu erwarten, dass sich die Zeit für die Ausführung der Regelinstanzen leicht erhöht, da während der Ausführung jetzt auch die Veränderungen an den Regelinstanzen berechnet werden. Beides zusammen, die Regelsuche und Regelausführung, ergibt eine Zeiteinsparung von 6 - 25 % bzw. 54 %.

Da das Laden der Modelle und die Ausführung der Benutzerkommandos nur unwesentlich von den ursprünglichen Werten abweichen, kann die Zeitersparnis bei den Regelauswertungszyklen als Ersparnis für die komplette Basismaschine übernommen werden.

Auffallend ist, dass der erste Regelauswertungszyklus nach dem Laden eines Modells deutlich mehr Zeit in Anspruch nimmt als zuvor. Dies wird durch die großen Veränderungen am Zustand der Basismaschine hervorgerufen, da im ersten Zyklus alle Entitäten und Relationen erzeugt und initialisiert werden, die im Verlauf der Simulation benötigt werden. Somit müssen sehr viele Veränderungen an der Menge der Instanzen berechnet werden. Dies zahlt sich allerdings später aus, da diese Arbeit nicht wiederholt werden muss.

### 7.4.2 Vorschlag “State”

Die Verbesserungen durch den Vorschlag “State” sind schwer durch diese Messung nachzuweisen, da sie durch andere Effekte überlagert werden. Die Veränderungen beschränken sich auf die Verwaltung des States und der Zugriffsfunktionen auf den State, da effizientere Implementierungen von Funktionen von der jetzigen Implementierung der Regelinstanzsuche nicht mehr benutzt werden.

### 7.4.3 Vorschlag “Stateprotocol”

Die Verbesserungen durch den Vorschlag “Stateprotocol” haben sich nicht eingestellt. Es kann keine Effizienzsteigerung zwischen originaler und optimierter Version beobachtet werden, wenn das Stateprotocol ausgegeben wird. Entgegen der Annahme fällt die Ausgabe der Daten in die Stateprotocoldatei zeitlich nicht ins Gewicht. Dies hat sich bei Versuchen ergeben, in denen die Daten zwar aus dem Zustand ausgelesen, aber nicht in die Stateprotocoldatei geschrieben werden.

# Kapitel 8
# Zusammenfassung

*Zunächst wird der Projektverlauf zusammengefasst und mit der Planung verglichen. Danach werden mögliche weitere Veränderungen am SESAM-Simulator angedacht.*

## 8.1 Projektverlauf

Nachdem alle Ergebnisse der Diplomarbeit beschrieben sind, soll in diesem Kapitel ein Resüme über die einzelnen Phasen gezogen werden. Die einzelnen Arbeitspakete werden beschrieben und besondere Vorkommnisse behandelt. Auch werden Metriken über die erzeugten Dokumente und den Code erhoben.

### 8.1.1 Projektplan

Am Anfang stand der Projektplan, der die einzelnen Arbeitspakete sowohl inhaltlich, als auch ihre Dauer beschreibt. Für jedes Arbeitspaket wurde jeweils ein Meilenstein festgelegt, der je nach Größe und Zusammensetzung des Pakets in weitere Meilensteine unterteilt sein kann. Für Arbeitspakete, deren Länge nicht genau abzuschätzen waren, oder die durch mögliche Probleme in die Länge gezogen werden könnten, wurden zwei Puffer von je einer Woche eingeplant. Diese wurden auch vollständig zur Bearbeitung der Arbeitspakete benötigt. Es gab eine Änderung des geplanten Zeitplans durch die Verschiebung des Zwischenvortrags. Ansonsten wurden alle Arbeitspakete zu den vorgegebenen Zeitpunkten beendet.

Das folgende Gantt-Chart zeigt den tatsächlichen Projektverlauf während der Diplomarbeit. Die gestrichelt umrandeten, nur leicht gefüllten Flächen, zeigen den ursprünglichen Projektverlauf vor der Verschiebung des Zwischenvortrags. Die Tabelle zeigt die einzelnen Arbeitspakete und die genauen Start- und Enddaten.

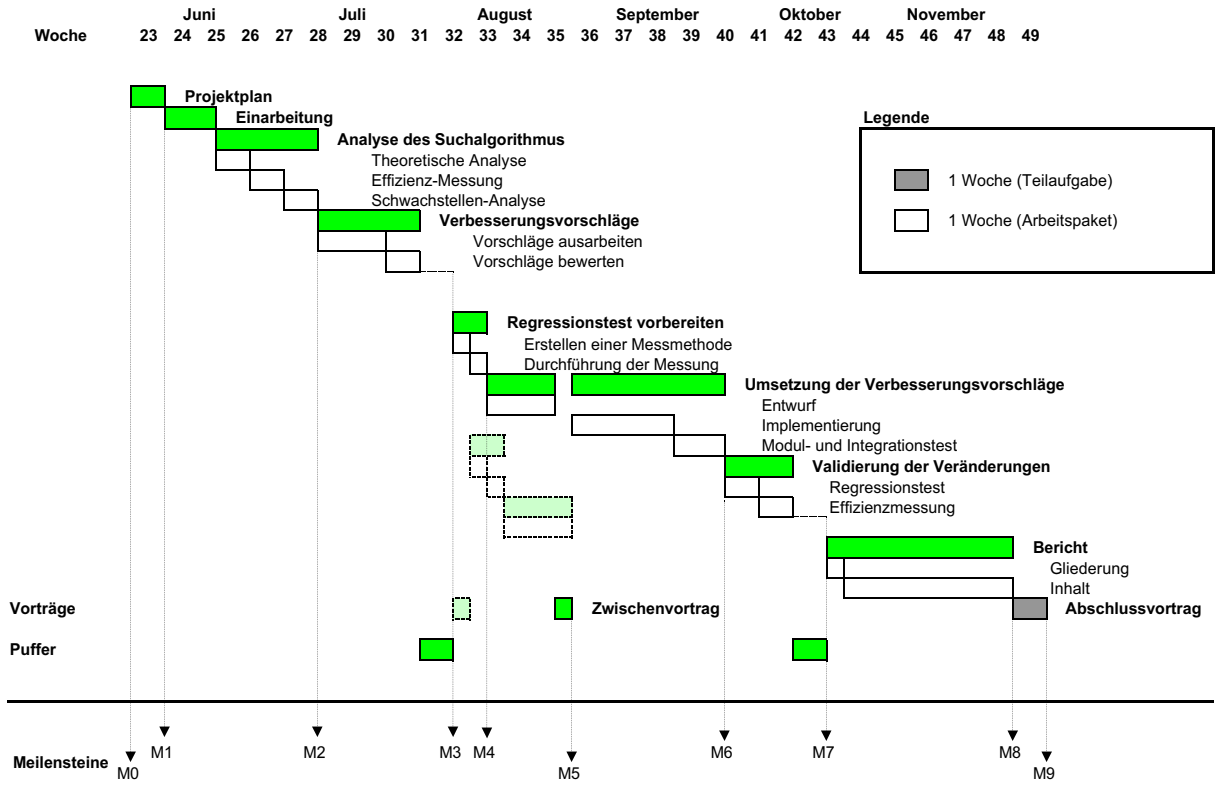



**Abbildung 31. Arbeitspakete und Meilensteine**

| Arbeitspaket | | Termin (Beginn - Ende) | Aufwand |
|---|---|---|---|
| Projektplan erstellen | | 03.06. - 07.06.2002 | 5 PT |
| Einarbeitung SESAM-System | | 10.06. - 19.06.2002 | 8 PT |
| Analyse des Suchalgorithmus | | 20.06. - 10.07.2002 | 15 PT |
| | Theoretische Analyse durchführen | 20.06. - 26.06.2002 | 5 PT |
| | Effizienz-Messungen durchführen | 27.06. - 03.07.2002 | 5 PT |
| | Schwachstellen-Analyse durchführen | 04.07. - 10.07.2002 | 5 PT |
| Verbesserungsvorschläge | | 11.07. - 31.07.2002 | 15 PT |
| | Vorschläge ausarbeiten | 11.07. - 24.07.2002 | 10 PT |
| | Vorschläge bewerten | 25.07. - 31.07.2002 | 5 PT |
| Puffer | | 01.08. - 07.08.2002 | 5 PT |
| Regressionstest vorbereiten | | 08.08. - 14.08.2002 | 5 PT |
| | Erstellen einer Messmethode | 08.08. - 09.08.2002 | 2 PT |
| | Durchführung der Messung | 09.08. - 14.08.2002 | 3 PT |
| Umsetzung der Verbesserungsvorschläge | | 15.08. - 02.10.2002 | 33 PT |

**Tabelle 31. Arbeitspakete und Termine**

| Arbeitspaket | | Termin (Beginn - Ende) | Aufwand |
|---|---|---|---|
| | Entwurf erstellen | 15.08. - 28.08.2002 | 10 PT |
| | Implementierung erstellen | 02.09. - 20.09.2002 | 15 PT |
| | Modul- und Integrationstest durchführen | 23.09. - 02.10.2002 | 8 PT |
| Zwischenvortrag | | 29.08. - 30.08.2002 | 2 PT |
| Validierung der Veränderungen | | 03.10. - 16.10.2002 | 10 PT |
| | Regressionstest durchführen | 03.10. - 09.10.2002 | 5 PT |
| | Effizienzmessung durchführen | 10.10. - 16.10.2002 | 5 PT |
| Puffer | | 17.10. - 23.10.2002 | 5 PT |
| Bericht erstellen | | 24.10. - 29.10.2002 | 27 PT |
| | Gliederung erstellen | 24.10. - 25.10.2002 | 2 PT |
| | Inhalt erstellen | 28.10. - 29.11.2002 | 25 PT |
| Abschlussvortrag | | 02.12. - 06.12.2002 | 5 PT |

**Tabelle 31. Arbeitspakete und Termine**

#### 8.1.1.1 Analyse

Die Analyse lässt sich in drei Teilbereiche gliedern. Als erstes Arbeitspaket nach der Einarbeitung wurde die Basismaschine auf ihre Komplexität analysiert. Genauer wurde dabei der Instanzsuchalgorithmus auf seine Laufzeit untersucht. Besonderes Augenmerk lag auf der Abhängigkeit der Laufzeit von dem Zustand der Basismaschine und dem Bedingungsteil der Regeln. Mit in die Untersuchung einbezogen wurden die zuvor gewonnenen Kennzahlen der Modelle. Hierdurch wurden für die aktuellen Modelle wichtige Stellen des Algorithmus hervorgehoben.

Die sich anschließende Effizienzmessung verwendet Daten aus der theoretischen Analyse, um Teile des Algorithmus einzugrenzen, die ein besonders schlechtes Laufzeitverhalten aufweisen. Es wurden neben den gemessenen Zeiten, Kennzahlen erhoben, mit denen es möglich war, die Ergebnisse der theoretische Analyse zu bestätigten. Bei der Auswertung der Messungen stellte sich jedoch heraus, dass die Grundannahme vor den Messungen falsch war. Nicht die Suche nach Regelinstanzen verbraucht die meiste Zeit, sondern die Ausführung der Regelinstanzen. Nach früheren Messungen mit alten Modellen, bei denen dies der Fall war, hat sich mit der Weiterentwicklung der Modelle der Zeitanteil deutlich verschoben. Als Ausnahme fiel dabei das Feingranulare QS-Modell auf, welches durch die höhere Anzahl an ausgeführten Regelauswertungszyklen wieder eine Verschiebung der Ausführdauer zur Regelsuche bewirkt.

Bei der Schwachstellenanalyse wurde zum einen die Möglichkeit gesehen, die Regelsuche weiter zu verbessern. Zum anderen wurde die Regelausführung genauer untersucht und dabei zwei weitere Schwachstellen entdeckt. Die Ausgabe des Stateprotocols hat sich negativ auf die Ausführung von verschiedenen Statements ausgewirkt, da hierbei während des Statements Daten an die Stateprotocoldatei angehängt werden. Weiter haben sich bei manchen Regeln sehr lange Ausführzeiten herausgestellt, was zum Teil mit einer ineffizienten Datenstruktur der Modelle für Anforderungen zusammenhängt.

#### 8.1.1.2 Verbesserungsvorschläge

Auf Grundlage der Analyse, besonders der gefundenen Schwachstellen, wurden Verbesserungsvorschläge in den Bereichen Regelinstanzsuche, Regelausführung und Modelle entwickelt und danach durch verschiedene Kriterien bewertet. Aufgrund dieser Bewertung wurden drei der Vorschläge ausgewählt, um sie während der Diplomarbeit umzusetzen.

#### 8.1.1.3 Umsetzung
Jeder der drei Vorschläge wurde jeweils als Entwurf ausgearbeitet und danach in Produktqualität implementiert. Da sich während der Implementierung herausgestellt hat, dass einer der Entwürfe nicht alle Spezialfälle abdeckt, musste dieser komplett überarbeitet und die Implementierung zu einem großen Anteil verändert werden. Die veränderten Bereiche der Basismaschine wurden durch Unit-Tests geprüft.

#### 8.1.1.4 Validierung
Die Validierung der Veränderungen besteht zum einen aus einem Regressionstest, der bereits vor der Umsetzung der Veränderungen vorbereitet wurde, und mehreren Effizienzmessungen. Durch den dreistufig aufgebauten Regressionstest kann mit großer Sicherheit festgestellt werden, dass weiterhin alle Anforderungen erfüllt bleiben. Da dies zu den Hauptbedingungen dieser Diplomarbeit zählt, wurde erst danach mit den vergleichenden Effizienzmessungen begonnen. Bei den Messungen stellte sich heraus, dass nicht alle der durchgeführten Verbesserungsvorschläge den erhofften Geschwindigkeitsvorteil erbringen konnten. Abhängig von dem simulierten Modell konnte jedoch eine Steigerung der Effizienz gemessen werden. Besonders stark wirkt sich die verbesserte Instanzsuche auf Simulationen mit der Feingranularen Variante des QS-Modells aus.

#### 8.1.1.5 Bericht
Die letzte Phase hat sich mit dem hier vorliegenden Dokument befasst. Dieser Bericht fasst alle Ergebnisse der vorangegangenen Phasen zusammen und versucht, die durchgeführte Arbeit zu bewerten. An diesen Bericht schließt sich noch der Abschlussvortrag an.

### 8.1.2 Aufwand und Ergebnisse
Während der Diplomarbeit wurde der Aufwand für die einzelnen Phasen in Stunden erfasst. Dadurch lässt sich der geschätzte Aufwand sehr genau mit dem tatsächlichen vergleichen. Als durchschnittliche Soll-Arbeitszeit pro Tag wurden 6h angenommen, was sich aus den vorgegebenen 800h Gesamtaufwand und 130 Arbeitstagen ergibt.

Die folgende Tabelle vergleicht zum einen den geschätzten und den tatsächlichen Aufwand. Zum anderen werden Metriken über die erzeugten Dokumente und den Code erhoben. Der Umfang von Dokumenten wird in Din-A4 Seiten angegeben. Bei Code wird der Aufwand in LOC angegeben. Eine Definition der Metrik LOC und eine genauere Untersuchung des Codes folgt in dem kommenden Kapitel.

| Arbeitspaket | Geschätzter Aufwand in h | Tatsächlicher Aufwand in h | Ergebnisse |
|---|---|---|---|
| Projektplan | 5 PT = 30h | 40h | Projektplan, 19 Seiten |
| Einarbeitung | 8 PT = 48h | 60h | |
| Analyse | 15 PT = 90h | 114h | Theoretische Analyse, 25 Seiten<br>Effizienzmessung, 14 Seiten<br>Schwachstellen, 9 Seiten |
| Vorschläge | 15 PT = 90h | 129h | Vorschlagsdokument, 36 Seiten |

**Tabelle 32. Arbeitspakete und Termine**

| Arbeitspaket | Geschätzter Aufwand in h | Tatsächlicher Aufwand in h | Ergebnisse |
|---|---|---|---|
| Umsetzung | 33 PT = 198h | 244h | Entwurf, 27 Seiten<br>Ada-Code, LOC<br>Test-Plan, 11 Seiten |
| Zwischenvortrag | 2 PT = 12h | 20h | Zwischenvortrag, 16 Folien |
| Regressionstest | 5 PT =30h | 36h | Regressionstest, 21 Seiten |
| Validierung | 10 PT = 60h | 113h | Validierung, 30 Seiten |
| Bericht | 27 PT = 162h | 225 | Endbericht, 120 Seiten |
| Gesamt | 130 PT = 800h | 981 | 312 Seiten, 2301 LOC, 16 Folien |

**Tabelle 32. Arbeitspakete und Termine**

#### 8.1.2.1 Untersuchung des Codes

Bei Codezeilen unterscheidet man zwischen Zeilen mit Anweisungen, Leerzeilen und Kommentarzeilen. Mischzeilen mit Anweisungen und Kommentaren werden zu den Anweisungszeilen gerechnet. Nach [Boehm, 1981] und [Jones, 1986] wird die Maßzahl LOC als die Anzahl der Zeilen definiert, die keine Leerzeilen und keine Kommantarzeilen sind. Es wird dabei nicht darauf geachtet, ob ein oder mehrere Statements in einer Zeile stehen.

Eine weitere Unterteilung findet statt in Code:

- der komplett neu entwickelt wurde - in neuen Modulen,
- der verändert wurde - in schon bestehenden Modulen,
- und Code der überflüssig wurde - von eingesparten schon bestehenden Modulen.

Der Code für die Unit-Tests wird getrennt aufgeführt.

| | LOC | Kommentarzeilen | Anzahl der Module |
|---|---|---|---|
| **Neuer Code** | 1551 | 1308 | 3 |
| **Veränderter Code** | 750 | 545 | 15 |
| **Eingesparter Code** | 1325 | 2175 | 6 |

**Tabelle 33. Codemetriken für die Basismaschine**

| | LOC | Kommentarzeilen | Anzahl der Module |
|---|---|---|---|
| **Neuer Code** | 1287 | 303 | 2 |
| **Veränderter Code** | 130 | 0 | 7 |

**Tabelle 34. Codemetriken für die Testfälle**

Um die Produktivität während der Implementierung zu ermitteln, werden sowohl neuer Code als auch veränderter Code addiert. Für die Implementierung waren 3 Wochen à 5 Tage angesetzt. Dies ergibt mit einer Anzahl von 2301 geschriebenen Codeszeilen eine Produktivität von circa 150 LOC pro Tag.

Hierbei muss jedoch beachtet werden, dass ein Großteil des neu entwickelten Codes aufgrund des Fehlers im Entwurf verworfen werden musste.

### 8.1.3 Abschließende Bewertung

#### 8.1.3.1 Verbesserungsvorschläge

Wie sich bei den Messungen herausgestellt hat, haben nicht alle Verbesserungsvorschläge ihr Ziel erreicht und die nicht implementierten werden sicherlich nicht in genau der beschriebenen Weise durchführbar sein. Sie bieten aber dennoch mehr als nur Anhaltpunkte für zukünftige Veränderungen, die auf die Effizienz der Basismaschine abzielen, wie sich bei der erfolgreich durchgeführten Implementierung zweier Vorschläge gezeigt hat.

#### 8.1.3.2 Implementierte Veränderungen

Alle der implementierten Veränderungen haben sich in Unit-Tests und Regressions-Tests als fehlerfrei herausgestellt. Es traten im Rahmen der Tests mit den bisher bestehenden Modellen keine Fehler auf. Somit kann die veränderte Version der Basismaschine ebenfalls für Schulungszwecke eingesetzt werden. Es kann jedoch nicht mit hundertprozentiger Sicherheit ausgeschlossen werden, dass durch einen nicht bedachten Spezialfall oder durch Veränderungen an den Modellen doch noch Fehler auftreten. Dies wird sich erst durch den Schulungseinsatz zeigen.

## 8.2 Ausblick

Die nicht in dieser Diplomarbeit durchgeführten Veränderungen werden sicherlich noch einen Geschwindigkeitsvorteil bringen, sollte entschieden werden, dass sich der Aufwand für ihre Umsetzung lohnt.

Eine Idee, die während des Zwischenvortrags aufkahm, sollte sicherlich weiterverfolgt werden. Anstatt wie bisher, alle Veränderungen während eines Regelauswertungszyklus sofort am Zustand des Simulators vorzunehmen, werden diese erst nach dem Ende des Zyklus durchgeführt. Dies wäre sicherlich für die Simulation der sauberere Ansatz und brächte auch für die Effizienz der Basismaschine einige Vorteile. Zum einen müssen keine Instanzen während eines Auswertungszyklus neu berechnet werden, zum anderen müssen die Regelinstanzen während des Zyklus nicht mehrmals geprüft werden, da nach einer Prüfung bekannt ist, ob diese ausgeführt werden können oder nicht.

# Anhang A

# Begriffslexikon

*In diesem Kapitel werden alle wichtigen Begriffe aus dem Umfeld des SESAM-Simulators definiert.*

Dieses Begiffslexikon ist übernommen aus dem internen Dokument der Abteilung Software-Engineering [Reißing, 1996, II]. Es wird um neue Begriffe aus dieser Diplomarbeit erweitert.

## 9.1 Aufgabe

Dieses Begriffslexikon stellt eine Sammlung von Begriffen aus dem SESAM-Projekt dar. Dabei wurden vor allem Begriffe aus dem SESAM-1- und SESAM-2-Umfeld berücksichtigt. Ziel dieses Begriffslexikons ist es, daß alle Projektbeteiligten mit jedem der hier aufgeführten Begriffe dieselbe Bedeutung assoziieren. Daher ist das Begriffslexikon primär als Nachschlagewerk (Referenz) konzipiert. Zugleich dient es der Vereinheitlichung der zum Teil stark unterschiedlichen Begriffsverwendungen.

## 9.2 Aufbau

Die Begriffe sind, zusammen mit ihrer Definition, alphabetisch sortiert aufgeführt. Begriffe mit beschreibenden Adjektiven sind hinter dem Begriff selbst einsortiert. Das Adjektiv wird nachgestellt (z.B. Modell, deskriptives). Querverweise auf andere Begriffe im Lexikon sind hervorgehoben.

Die Begriffe sind nach ihrem Kontext klassifiziert:

- SESAM-1 (S1),
- SESAM-2 (S2),
- SESAM-1 und -2 (S1/2),
- SESAM allgemein (SA),
- Diplomarbeit Franke 2002 (DA-FRANKE).

Die Klassifikation wird in eckige Klammern eingeschlossen und der Definition vorangestellt.

## 9.3 Begriffe

**Aggregation [SA, nicht realisisiert]**
Eine Art von Beziehungen zwischen *Entitäten*. Eine Aggregation drückt eine Enthaltenseinsbeziehung (Teil-von-Beziehung) von *Entitäten* aus. Dies könnte zum Beispiel durch eine spezielle Kennzeichnung von *Relationstypen* im *Schemamodell* realisiert werden. Der hier verwendete Aggregationsbegriff ist an den der objektorientierten Analyse angelehnt.

**Aktion [S1/2]**
Anweisung im *Aktionsblock*. Aktionen sind zum Beispiel:

- die Zuweisung an ein *Attribut*,

- das Zuweisen eines Werts an einen *Modifikator* eines *Attributs*,
- das Einfügen oder Löschen einer *Komponente* in der aktuellen *Situation,*
- sowie das Versenden einer *Nachricht*.

**Aktionsblock [S2]**
Bestandteil eines *Effekts*. Enthält die *Aktio*nen. Der Aktionsblock wird ausgeführt, wenn der *Bedingungsblock* des *Effekts* erfüllt ist. Bei einem *Effekt* mit *Aktivierung/Deaktivierung*-Semantik besteht der Aktionsblock aus einem *Aktivierungsblock*, einem *Aktivblock* und einem *Deaktivierungsblock*; bei einem *Effekt* mit *Feuern*-Semantik (z.B. einem *Benutzerkommando*) besteht der Aktionsblock lediglich aus einem *Aktivierungsblock*. [S1] In *SESAM-1* findet eine strenge Trennung der drei Blöcke nicht statt. Implizit sind die Blöcke jedoch vorhanden. Der *Aktivierungsblock* umfasst alle *Aktionen*, die bei der Instantiierung eines *Effekts* ausgeführt werden. Dazu gehört die Erzeugung neuer *Komponenten*, das Zuweisen von Werten an *Modifikatoren*, die Zuweisung an *Attribute* und das Versenden einer Aktivierungs-*Nachricht*. Der *Aktivblock* existiert nur implizit durch die im *Aktivierungsblock* vereinbarten *Modifikatoren*, deren Werte mit dem aktuellen *Attributwerten* verrechnet werden. Der *Deaktivierungsblock* entfernt implizit die *Modifikatoren* des *Aktivierungsblocks* und kann explizit eine Deaktivierungs-*Nachricht* versenden.

**Aktionsteil [S1/2, veraltet]**
siehe *Aktionsblock*.

**Aktivierung/Deaktivierung [S1/2]**
Spezielle Semantik der Ausführung für *Instanzen* kontinuierlicher *Effekte*. Der *Aktionsblock* eines *Effekts* mit Aktivierung/Deaktivierung-Semantik zerfällt in drei Teile: einen *Aktivierungsblock*, einen *Aktivblock* und einen *Deaktivierungsblock*. [S1: zur Zuordung der Blöcke siehe *Aktionsblock*] Kann in der aktuellen *Situation* eine *Instanz* eines kontinuierlicher *Effekts* gebildet werden, wird der *Effekt* instantiiert. Dabei wird der *Aktivierungsblock* der *EffektInstanz* ausgeführt. Solange der *Bedingungsblock* erfüllt ist, bleibt die *EffektInstanz* erhalten. Nun wird in jedem weiteren *Simulationsschritt* ihr *Aktivblock* ausgeführt. Wird die aktuelle *Situation* durch die *EffektInstanz* selbst oder durch andere *EffektInstanzen* so verändert, dass der *Bedingungsblock* nicht mehr erfüllt ist, so wird die *EffektInstanz* im folgenden *Simulationsschritt* entfernt. Dabei wird ihr *Deaktivierungsblock* ausgeführt.

**Aktivierungsblock [S2]**
Bestandteil des *Aktionsbloc*ks eines *Effekts*. Der *Aktivierungsblock* enthält die *Aktionen*, die ausgeführt werden sollen, wenn ein *Effekt* instanziiert wird. [S1: nur implizit, siehe *Aktionsbloc*k]

**Aktivierungsteil [S2, veraltet]**
siehe *Aktivierungsblock*.

**Aktivblock [S2]**
Bestandteil des *Aktionsblocks* eines kontinuierlichen *Effekts*. Der Aktivblock enthält die *Aktionen*, die in jedem *Simulationsschritt* für jede *EffektInstanz* ausgeführt werden sollen, sofern deren *Bindung* gültig ist. [S1: nur implizit, siehe *Aktionsblock*]

**Aktivteil [S2, veraltet]**
siehe *Aktivblock*.

**Aktivität [S2]**

Eine Aktivität steht für eine übergeordnete Tätigkeit im *SESAM-Modell*, zum Beispiel die Erstellung der Spezifikation. Eine Aktivität wird durch eine *Existenzregel* definiert. Eine Aktivität enthält eine Menge von *Effekten*. Da Aktivitäten *Effekte* sind, kann eine Aktivität wiederum Aktivitäten enthalten. Auf diese Weise können die *Effekte* des *Effektemodells* logisch und hierarchisch gruppiert werden. Falls die *Existenzregel* instantiiert wurde, sind die in der Aktivität enthaltenen (eingeschachtelten) *Effekte* aktivierbar, sonst nicht. Die enthaltenen *Effekte* können auf die durch die *Instanz* der *Existenzregel* der Aktivität gebundenen *Komponenten* zugreifen.

**Attribut [S1/2]**
Attribute dienen der Beschreibung von Eigenschaften einer *Komponente* [S1: und eines *Events*]. Sie haben einen *Namen*, einen *Attributtyp* und einen *Attributmodus*. Attribut, abgeleitetes. [S1/2] Attribut mit *Attributmodus* „abgeleitet“. Abgeleitete Attribute dienen der Modellierung uneigenständiger Eigenschaften, d.h. Eigenschaften, die aus anderen Eigenschaften berechenbar sind. Zu diesem Zweck muss zu einem abgeleiteten Attributs eine Ableitungsvorschrift (Berechnungsvorschrift) angegeben werden. Der Wert des abgeleiteten Attribut wird dann bei jedem Zugriff aus den aktuellen Werten der Attribute in der Ableitungsvorschrift neu berechnet.

**Attribut, diskretes [S1/2]**
Attribut mit *Attributmodus* „diskret“. Diskrete Attribute dienen der Modellierung von eigenständigen Eigenschaften, deren Werte sich nur sprunghaft ändern. Der Wert eines diskreten Attributs kann nur durch diskrete Zuweisungen im *Aktivierungsblock* bzw. im *Deaktivierungsblock* eines kontinuierlichen *Effekts* und im *Aktivierungsblock* eines diskreten *Effekts* verändert werden.

**Attribut, kontinuierliches [S1/2]**
Attribut mit *Attributmodus* „kontinuierlich“. Kontinuierliche Attribute dienen der Modellierung von eigenständigen Eigenschaften, deren Werte sich „fließend“ (kontinuierlich) ändern. Nach der Zuweisung eines Anfangswerts kann der Wert nur durch Angabe eines Veränderungswerts *(Modifikato*r) verändert werden. Der *Modifikator* wird in jedem *Simulationsschritt* auf den Attributwert angewandt.

**Attributbedingung [S1/2]**
Bestandteil des *Attributbedingungsblocks* eines *Effekts*. Ein Boolsches Prädikat über den *Attributen* der formalen *Komponenten* des *Strukturbedingungsblocks*.

**Attributbedingungsblock [S2]**
Bestandteil des *Bedingungsblocks* eines *Effekts*. Damit der Attributbedingungsblock erfüllt ist, müssen alle enthaltenen *Attributbedingungen* erfüllt sein.

**Attributbedingungsteil [S1/2, veraltet]**
siehe *Attributbedingungsblock*.

**Attributmodus [S1/2]**
Bestandteil eines *Attributs*. Der Attributmodus bestimmt, wie sich der *Attributwert* über die Simulationszeit ändert. Außerdem legt er fest, in welchen Teilen des *Aktionsteils* eines *Effekts* der *Attributwert* verändert werden darf. Es gibt drei Attributmodi: diskret, kontinuierlich und abgeleitet (siehe *Attribut*).

**Attributtyp [S1/2]**
Typ für *Attribute*. Teil des *Schemamodells*.

**Ausführungswerkzeug**
*Werkzeu*g, mit dem der *Spieler* ein *SESAM-Modell* ausführen kann. In *SESAM-1* der *Simulato*r, i*n SESAM-2* die Kombination aus *Basismaschine* und *Dolmetscher.*

**Auswertungswerkzeug [S1/2]**
*Werkzeu*g, mit dem der *Tutor* ein *Spiel* analysieren kann. Er kann *Spielstände* und *Attribut*wertverläufe visualisieren und analysieren. Grundlage der Auswertung ist ein *Protokoll* des *Spiels*.

**BASE-2 [S2]**
Eine *Basissprach*e. BASE-2 ist dieAbkürzung für „Basissprache für SESAM-2“.

**Basismaschine [S2]**
*Ausführungswerkzeug* für Modellbeschreibungen in *Basissprach*e. Die Basismaschine erzeugt während der Modellausführung ein *Protokoll* des *Spiels*. Die Basismaschine realisiert damit einen Teil der Funktionalität des *Simulators* i*n SESAM*-1.

**Basissprache [S2]**
*Modellbeschreibungssprache* der *Basismaschin*e. Sie ist Zielsprache der Übersetzung einer *Hochsprache*. Die Basissprache besitzt im Vergleich zur *Hochsprache* wenige Konzepte, da sie primär ausführungsorientiert ist.

**Bedingungsblock [S1/2]**
Bestandteil eines *Effekts*. Bestimmt, ob der *Effekt* ausgeführt werden darf. Der Bedingungsblock besteht aus dem *Strukturbedingungsblock* und dem *Attributbedingungsblock*. [S1] Hinzu kommt in *SESAM-1* noch der *Eventbedingungsteil*. [S2] Hinzu kommt in *SESAM-2* noch der *Kausalitätsflussblock*. Damit ein *Effekt* ausgeführt werden darf, muss der Bedingungsblock erfüllt sein. Dies bedeutet, daß eine *Bindung* des *Strukturbedingungsblocks* gefunden wird und der *Attributbedingungsblock* ([S1] und der *Eventbedingungsteil*) ([S2] und der *Kausalitätsflußblock*) für diese *Bindung* erfüllt ist.

**Bedingungsteil [S1/2, veraltet]**
siehe *Bedingungsblock*.

**Benutzer [S1/2]**
Das *SESAM-System* verfügt über drei verschiedene Benutzerklassen. Das sind die *Modellierer*, die *Spieler* und die *Tutoren*. Benutzer ist der gemeinsame Oberbegriff. (Vorsicht: In vielen Dokumenten zum *SESAM-Projekt* wird unter dem Begriff Benutzer nur der *Spieler* verstanden.)

**Benutzerkommando [S2]**
Exogener, diskreter *Effekt*. Die Ausführung eines Benutzerkommandos kann nur vom Benutzer veranlasst werden. Dabei legt der Benutzer die Belegung der formalen *Parameter* fest. Der Aufbau eines Benutzerkommandos ist ähnlich dem einer diskreten *Regel*. Ein Benutzerkommando hat einen *Namen*, einen *Zeitverbrauch* und eine Liste formaler *Parameter*. Hinzu kommt ein *Bedingungsblock* und ein *Aktionsblock* (bestehend aus einem *Aktivierungsblock*).

**Betreuer [S1/2]**
Siehe *Tutor*.

**Bindung [S1/2]**

Isomorphe Abbildung der formalen *Komponenten* des *Strukturbedingungsblocks* in die aktuelle *Situation*. Zusätzlich gilt, dass jeder formalen *Komponente* eine *Komponente* typverträglichen Typs zugeordnet ist.

**Bindung, gültige [S2]**
*Bindung* eines *Strukturbedingungsblocks*, zu der der *Attributbedingungsblock* in der aktuellen *Situation* erfüllt ist.

**Deaktivierungsblock [S2]**
Bestandteil des *Aktionsblocks* eines kontinuierlichen *Effekts*. Der Deaktivierungsblock enthält die *Aktionen*, die ausgeführt werden sollen, wenn der *Bedingungsblock* einer *EffektInstanz* nicht mehr erfüllt ist. Die *EffektInstanz* wird nach der Ausführung des Deaktivierungsblocks gelöscht. [S1: nur implizit, siehe *Aktionsblock*]

**Deaktivierungsteil [S2, veraltet]**
siehe *Deaktivierungsblock*.

**Derived [DA-FRANKE]**
Wird verwendet, um die Beziehung von Entitäts- oder Relationstypen zu beschreiben. Bezeichnet alle Typen, die konform zu einem Typ sind, also auch den Typ selbst. Dieser Begriff wurde aufgrund der widersprüchlichen Verwendung des Konformitätsprinzip in der Basismaschine eingeführt.

**Dolmetscher [S2]**
*Ausführungswerkzeug*, das zwischen dem *Spieler* und der *Basismaschine* vermittelt. Nimmt Eingaben des *Spielers* entgegen und wandelt sie in *Kommandos* an die *Basismaschine* um. Umgekehrt schickt die *Basismaschine Nachrich*ten an den Dolmetscher, der sie in Ausgaben an den *Spieler* übersetzt. Zur Vermittlung bedient sich der zur Zeit verfügbare Dolmetscher eines *Wörterbuchs*.

**Effekt [S2]**
Oberbegriff für *Regeln*, *Aktivitäten* und *Benutzerkommandos*. Effekte erzeugen die Dynamik eines *SESAM-Modells*. Sie werden im *Effektemodell* zusammengefasst. Es lassen sich endogene (modellinterne) und exogene (von außen auf das Modell einwirkende) Effekte unterscheiden. Alternativ lassen sich die Effekte nach ihrer Ausführungssemantik in kontinuierliche (mit *Aktivierung/Deaktivierun*g-Semantik) und diskrete Effekte (mit *Feuern*-Semantik) klassifizieren. [S1] In *SESAM-1* existiert dieser Begriff nicht, soll hier jedoch im Zusammenhang mit *SESAM-1* die kontinuierlichen *Regeln* bezeichnen. Diese sind die einzige Art von *SESAM*-2-Effekten, die es in *SESAM-1* gibt.

**Effekt, aktivierbarer.**
Ein Effekt, der für den Effektausführungsmechanismus sichtbar ist. Dies sind zum einen die Effekte, die in keiner Aktivität enthalten sind, und zum anderen die Effekte, die in einer aktiven Aktivität enthalten sind. Nicht aktivierbare Effekte sind grundsätzlich nicht instantiierbar.

**Effekt, diskreter**
Diskrete Effekte modellieren zeitpunktartige Geschehnisse. Sie haben *Feuern*-Semantik. Hierunter fallen diskrete *Regeln* und *Benutzerkommandos*.

**Effekt, endogener [S2]**

Endogene Effekte werden innerhalb des Modells ausgelöst. Hierunter fallen die *Regeln* und die *Aktivitäten*.

**Effekt, exogener [S2]**
Exogene Effekte werden außerhalb des Modells ausgelöst. Hierunter fallen die *Benutzerkommandos*.

**Effekt, instantiierbarer [S2]**
Ein Effekt, zu dem in der aktuellen *Situation* eine *Instanz* gebildet werden kann. Voraussetzung für die Instantiierbarkeit ist die Aktivierbarkeit (siehe *Effekt, aktivierbarer*).

**Effekt, kontinuierlicher [S2]**
Kontinuierliche Effekte modellieren zeitraumartige Geschehnisse. Sie haben *Aktivierung/Deaktivierung*-Semantik. Hierunter fallen kontinuierliche *Regeln* und *Aktivitäten*.

**Effektemodell [S2]**
Bestandteil eines *SESAM-Modells*. Enthält alle *Effekte*, also *Regeln*, *Aktivitäten* und *Benutzerkommandos*.

**Entität [S1/2]**
*Instanz* eines *Entitätstyps*. Besitzt einen externen *Namen*.

**Entität, formale [S1/2]**
Bestandteil der *Strukturbedingung* eines *Effekts*. Besteht aus einem *Namen* und einem *Entitätstyp*.

**Entitätstyp [S1/2]**
Typ für die Objekte der abstrakten Welt, die durch das Modell definiert ist und in der die Dynamik des Modell abläuft. Besteht aus einem *Nam*en, einer optionalen Vererbungsklausel und der Deklaration von *Attributen*. Es gibt abstrakte und konkrete Entitätstypen. Bestandteil des *Schemamodells*.

**Event [S1]**
Ereignis. *Instanz* eines *Eventtyps*. Es werden exogene Events und endogene Events unterschieden.

**Event, endogener [S1]**
Ereignis, das im Modell selbst erzeugt wird. Endogene Events dienen vor allem dazu, Ereignisse, die irgendwann in der Zukunft eintreten sollen, zu modellieren.

**Event, exogener [S1]**
Ereignis, das außerhalb des Modells erzeugt wird und dann auf das Modell einwirkt. Exogene Events dienen dazu, die Auswirkungen von Benutzeraktionen zu modellieren. In *SESAM-2* werden sie durch die *Benutzerkommandos* ersetzt.

**Event, formaler [S1]**
Bestandteil der *Eventbedingung* eines *Effek*ts. Besteht aus einem *Nam*en und einem *Eventty*p.

**Eventbedingung [S1]**
Bestandteil des *Eventbedingungsteils* eines *Effek*ts. Die Eventbedingung besteht aus einem formalen *Even*t.

**Eventbedingungsteil [S1]**
Bestandteil des *Bedingungsteils* eines *Effekts*.

**Eventtyp [S1]**
Typ für ein Ereignis, den sogenannten *Event*. Besteht aus einem *Namen*, *Rollen* und *Attributen*. In *SESAM-1* sind Eventtypen Bestandteil des *Schemamodells*.

**Existenzregel [S2]**
Eine Existenzregel ist eine Art kontinuierlicher *Regel*. Sie legt fest, wann eine *Aktivität* instantiiert werden darf: Wenn der *Bedingungsblock* der Existenzregel erfüllt ist, wird die *Aktivität* instantiiert. Für jede *Aktivitäten-Instanz* sind die enthaltenen *Effek*te aktivierbar und können auf die formalen *Komponenten* des *Strukturbedingungsblocks* der Existenzregel zugreifen. Der Aufbau einer Existenzregel entspricht dem Aufbau einer kontinuierlichen *Regel*.

**Feuern [S2]**
Spezielle Semantik der *Effek*tausführung. Im Gegensatz zur *Aktivierung/Deaktivierun*g, das eher zeitraumorientiert ist, entspricht das Feuern einer zeitpunktartigen *Effek*tausführung. Ein *Effekt* feuert, wenn sein *Bedingungsblock* erfüllt ist. Feuern bedeutet, dass der *Aktionsblock* des *Effekts* ausgeführt wird. Danach wartet der *Effekt* darauf, wieder feuern zu können.

**Historie [SA, nicht realisiert]**
Die Geschichte eines *Spiels* in Form einer zeitlich geordneten Liste aller bis zur aktuellen *Simulationszeit* durchlaufenen *Situationen*. Dabei wird für jeden *Simulationsschritt* nur die zeitlich letzte *Situation* gespeichert, nicht jedoch andere, während des *Simulationsschritts* durchlaufene Zwischenzustände. Die Historie soll es dem *SESAM-Modell* ermöglichen, auf bereits durchlaufene *Situationen* zurückzugreifen.

**Hochsprache [S2]**
*Modellbeschreibungssprache* von *SESAM*-2. Besitzt im Vergleich zur *Basissprache* viele, reichhaltige Konzepte. Kann durch einen *Hochsprachenübersetzer* in *Basissprache* übersetzt werden. Die Modellbeschreibung in *Hochsprache* wird von den *Modellierungswerkzeugen* erzeugt.

**Hochsprachenübersetzer [S2]**
*Werkzeu*g, das eine Modellbeschreibung in *Hochsprache* in eine Modellbeschreibung in *Basissprache* übersetzt.

**Instanz [S1/2]**
Wert (im Sinne einer konkreten Wertebelegung) eines Typs, z.B. eines *Effek*ts. Eine *Effek*t-Instanz ist ein *Effekt*, dessen *Strukturbedingungsblock* und dessen formale *Parameter* gültig gebunden sind.

**Kardinalität [SA, nicht realisiert]**
Mögliche Erweiterung einer *Rolle*. Die Kardinalität einer *Rolle* legt die minimale und maximale Anzahl von *Relatio*nen (zu einem Zeitpunkt) fest, in denen eine beliebige, aber feste *Instanz* des *Rolle*ntyps vorkommen darf. Kardinalitäten schränken damit die maximale Anzahl von *Instanz*en einer *Relation* ein. Beispiel: Ein *Relationstyp* Reviews verfügt über eine *Rolle* who vom Typ Developer und eine *Rolle* what vom Typ Document. Die *Rolle* who verfügt über eine Kardinalität von (0,1), d.h. ein Entwickler prüft kein oder höchstens ein Dokument gleichzeitig. Die *Rolle* what

verfügt über eine Kardinalität von (0,5), d.h. ein Dokument wird von keinem oder höchstens 5 Entwicklern geprüft.

**Kausalitätsfluss [S2]**
Durch den Kausalitätsfluss können kausale Abhängigkeiten zwischen endogenen *Effekten* definiert werden. Zum Beispiel kann die Ausführung einer *Regel* Voraussetzung der Ausführung einer anderen *Regel* sein, da sie entsprechende Voraussetzungen schafft. Im Rahmen des Kausalitätsflusses können Angaben darüber gemacht werden, wieviel Simulationsschritte nach der Instantiierung eines endogenen *Effekts* vergehen müssen, bevor ein anderer endogener *Effekt* aktivierbar ist. Zusätzlich kann noch angegeben werden, wieviele *Instanz*en der Nachfolger-*Effekt* maximal erzeugen darf.

**Kausalitätsflussblock [S2]**
Bestandteil eines endogenen *Effekts* zur Modellierung eines *Kausalitätsfluss* zwischen *Effekten*. Gibt an, wie oft und nach welcher Verzögerungszeit der *Effekt* nach der Aktivierung eines anderen endogenen *Effekts* instantiiert werden kann. Auf diese Weise können direkte Abhängigkeiten zwischen endogenen *Effekten* formuliert werden.

**Kausalitätsflussteil [S2, veraltet]**
siehe *Kausalitätsflussblock*

**Kommando [S2]**
Ergebnis der Übersetzung einer *Spieler*eingabe durch den *Dolmetscher*, das an die *Basismaschine* weitergeleitet wird und dort die Ausführung eines *Benutzerkommand*os auslösen kann. Ein Kommando hat einen *Namen* und eine (möglicherweise leere) Menge von *Parametern*.

**Komponente [S1/2]**
Oberbegriff für *Entitäten* und *Relationen*.

**Komponente, formale [S1/2]**
Oberbegriff für formale *Entitäten* und formale *Relationen*.

**Komponententyp [S1/2]**
Oberbegriff für *Entitätstypen* und *Relationstypen*.

**Konformität [DA-FRANKE]**
Konformität stellt eine Beziehung zwischen Typen dar. Wenn ein Typ A konform zu einem Typ B ist, kann ein Objekt vom Type A anstelle eines Objektes vom Typ B verwendet werden, da A alle Eigenschaften von B zur Verfügung stellt. Jeder Typ ist auch zu sich selbst typverträglich. Hierdurch wird das Vererbungskonzept aus der Hochsprache in die Basissprache umgesetzt. Die Konformität bezieht sich auf Entitäts- und Relationstypen.

Ein Entitätstyp A, der zu B konform ist, besitzt alle Attribute von B und kann noch weitere definieren.

Ein Relationstyp A, der zu B konform ist, besitzt alle Rollen von B und kann noch weitere definieren. Hier ist auch die Koformität zwischen Rollen definiert. Die Rolle A ist konform zur Rolle B, wenn die dazugehörigen Relationstypen konform sind und der Name der Rollen gleich ist.

**Modell [SA]**
Abbild eines *Originals*. Dient zur Veranschaulichung oder Erklärung bestimmter Verhaltensweisen des *Originals*. Häufig im Sinne von *SESAM-Modell* gebraucht, wobei das *Original* dann ein Software-Projekt ist.

**Modell, deskriptives [SA]**
Modelliert ein bestehendes *Original*, d.h. es ist eine phänomenologische Nachbildung (Abbild) dieses *Originals*. Der Spieler ist in seinen Eingaben nicht eingeschränkt. Ziel eines deskriptiven Modells ist es, die Wirkung eines beliebigen Verhaltens im *Original* aufzuzeigen. Gegensatz: Präskriptives Modell.

**Modell, präskriptives [SA]**
Modelliert ein fiktives *Original*. Der Spieler ist in seinen Eingaben stark eingeschränkt; in der Regel ist sein Verhalten vorgeschrieben. Ziel eines präskriptiven Modells ist es, die Wirkung eines (vom *Modellierer*) gewünschten Verhaltens aufzuzeigen, um damit die Konsequenzen (Vor- bzw. Nachteile) dieses Verhaltens zu verdeutlichen. Gleichzeitig dient ein präskriptives Modell als Anleitung für das gewünschte Verhalten (Vorbild). Gegensatz: Deskriptives Modell.

**Modell, qualitatives [SA]**
Modell, das durch qualitative Aussagen über seine Bestandteile und ihrer Beziehungen untereinander charakterisiert ist. Es können Aussagen über Zusammenhänge gemacht werden, diese Aussagen können jedoch nicht quantifiziert werden.

**Modell, quantitatives [SA]**
Modell, das durch formale, quantitative Aussagen über seine Bestandteile und ihre Beziehungen untereinander charakterisiert ist. Quantitative Modelle können, wenn sie in einer geeigneten Sprache formuliert sind, von einem Rechner simuliert werden.

**Modellbeschreibungssprache [S1/2]**
Sprache zur Beschreibung eines *SESAM-Modells*. [S2] In *SESAM-2* werden zwei Ebenen unterschieden: *Hochsprache* und *Basissprach*e.

**Modellbauer [S1/2]**
Siehe *Modellierer.*

**Modellierer [S1/2]**
Entwerfer und Realisierer eines *SESAM-Modells*. Der Modellierer verwendet die *Modellierungswerkzeuge* zum Zwecke der *Modell*erstellung oder -modifikation.

**Modellierungswerkzeuge [S1/2]**
*Werkzeu*ge, die zur interaktiven Erstellung einer *Modell*beschreibung in *Hochsprache* dienen.

**Modifikator [S1/2]**
Bezeichner/Speicherplatz für Veränderungsgrößen für kontinuierliche *Attribute*. Es gibt additive und multiplikative Modifikatoren. Es dürfen für ein kontinuierliches *Attribut* entweder beliebig viele additive oder beliebig viele multiplikative Modifikatoren gesetzt werden. Additive und multiplikative Modifikatoren schließen sich gegenseitig aus. [S1] Modifikatoren für kontinuierliche *Attribute* werden im *Aktivierungsblock* gesetzt und im *Deaktivierungsblock* einer *Regel* implizit

gelöscht. Statt Modifikator wird hier auch der Begriff „Rate“ verwendet. [S2] Modifikatoren für kontinuierliche *Attribute* können nur im *Aktivblock* eines kontinuierlichen *Effekts* gesetzt werden.

**Modifikator, additiver [S1/2]**
Wert, der in jedem *Simulationsschritt* zu dem aktuellen Wert des zugehörigen kontinuierlichen *Attributs* addiert wird. Häufig auch als „Delta“ bezeichnet.

**Modifikator, multiplikativer [S1/2]**
Wert, der in jedem *Simulationsschritt* mit dem aktuellen Wert des zugehörigen kontinuierlichen *Attributs* multipliziert wird. Häufig auch als „Filter“ bezeichnet.

**Nachricht**
Mitteilung des *SESAM-Modells* an den *Spiele*r. [S1] In *SESAM-1* sind Nachrichten durch sogenannte „Notes“ in den *Regeln* realisiert. [S2] In *SESAM-2* liegt die Nachricht in einer symbolischen Form (Nachrichtenbezeichner und Parameter) vor, die durch den *Dolmetscher* in eine dem *Spieler* verständliche Form gebracht und ausgegeben wird. Das Versenden einer Nachricht ist eine *Aktio*n.

**Name [S1/2]**
hier: Bezeicher eines Dinges, z.B. einer *Entität* oder eines *Entitätstyps*.

**Name, externer [S2]**
*Nam*e, unter dem eine *Entität* nach außen, d.h. dem *Dolmetscher* und dem *Spieler*, bekannt ist.

**Nicht-Struktur [SA, nicht realisiert]**
Struktur, die nicht vorhanden sein darf, damit der *Strukturbedingungsblock* eines *Effekts* erfüllt ist.

**Original [SA]**
Ausschnitt der realen Welt oder eines Modells, der zum Gegenstand einer Modellierung gemacht wird. In *SESAM-Modellen* in der Regel ein Software-Projekt.

**Parameter [S2]**
Bestandteil eines *Kommando*s. Ein Parameter ist damit der (aktuelle) Wert für den formalen Parameter des zum *Kommando* passenden *Benutzerkommando*s.

**Parameter, formaler [S2]**
Bestandteil eines *Benutzerkommando*s. Dient dazu, Parameter eines *Kommando*s vom Benutzer aufzunehmen. Besteht aus einem *Nam*en und einem Typ, der ein *Entitätstyp* oder ein *Bedingungsblock* sein kann.

**Priorität [S2]**
In *BASE-2* Bestandteil einer *Regel*. Je höher die Priorität ist, desto früher wird die *Regel* bei der *Rege*lausführung berücksichtigt.

**Protokoll [S1]**
Vom *Simulator* geführte Aufzeichnung über die eingegebenen Kommandos des *Spiele*rs. [S2] Von der *Basismaschine* geführte Aufzeichnungen über die eingegebenen Kommandos des *Spielers* und die *Spie*lstände während der Modellsimulation. Das Protokoll dient als Eingabe für die *Auswertungswerkzeuge* und damit zur Analyse eines *Spiels*.

**Regel [S1]**
Eine Regel hat einen *Nam*en, einen *Zeitverbrauc*h, eine Aktivierungs*Nachricht*, eine Deaktivierungs*Nachricht* und *Aktionen* zum Zeitpunkt der Instantiierung. Eine Regel hat *Aktivierung/Deaktivierun*g-Semantik. [S2] Eine Regel hat einen *Nam*en und einen *Zeitverbrauch* und besitzt einen *Bedingungsblock* und einen *Aktionsbloc*k. (In *BASE-2* kommt eine *Priorität* hinzu, dafür entfällt der *Kausalitätsflussblock* des *Bedingungsblocks*.) Es gibt zwei verschiedene Regelausführungssemantiken: das *Feuern* und die *Aktivierung/ Deaktivierung*. Danach werden diskrete und kontinuierliche Regeln unterschieden.

**Regelmodell [S1]**
Bestandteil eines *SESAM-Modells*. Enthält alle *Regeln*. [S2, veraltet] siehe *Effektemodell*.

**Relation [S1/2]**
*Instanz* eines *Relationstyps*.

**Relation, formale [S1/2]**
Bestandteil des *Strukturbedingungsblocks* eines *Effekts*. Besteht aus einem *Namen*, einem *Relationstyp* und einer *Rollen*belegung durch formale *Entitäten*.

**Relationstyp [S1/2]**
Typ für die Beziehungen zwischen den Objekte der abstrakten Welt des Modells, also für Beziehungen zwischen *Entitäten*. Bestandteil des *Schemamodells*. Relationstypen bestehen aus einem *Namen* ([S2], einer optionalen Vererbungsklausel) und der Deklaration von *Rollen* und *Attributen*. Ein Relationstyp muss mindestens zwei *Rollen* deklarieren. [S2] Es gibt abstrakte und konkrete Relationstypen.

**Rolle [S1/2]**
Bestandteil eines *Relationstyps* [S1: möglicherweise auch eines *Eventtyps*]. Eine Rolle hat einen *Namen* und einen Typ, der ein *Entitätstyp* sein muss. In einer *Instanz* des *Relationstyps* nehmen die Rollen die an der —durch die *Relation* beschriebenen — Beziehung beteiligten *Entitäten* auf. Man spricht dann von einer Rollenbelegung.

**Schemamodell [S1/2]**
Bestandteil eines *SESAM-Modells*. Beschreibt ähnlich einem Entity-Relationship-Modell die abstrakte Welt, in der das Modell abläuft. [S2] Besteht aus *Attributtypen*, *Entitätstypen* und *Relationstypen* [S1: zusätzlich auch noch die *Eventtypen*].

**SEMOS-2 [S2]**
Eine *Hochsprache*. Abkürzung für „SESAM-Modellbeschreibungssprache für SESAM-2“.

**SESAM [SA]**
Abkürzung für „Software Engineering Simulation by Animated Models“.

**SESAM-1 [SA]**
Erste Implementierung des *SESAM-Syste*ms. Dient als Pilotsystem für weitere Implementierungen.

**SESAM-2 [SA]**
*SESAM-Syste*m, das als Nachfolger des *SESAM*-1-Systems konzipiert wurde.

**SESAM-Lite [SA]**
Ein Prototyp für mögliche Konzepte von *SESAM*-2.

**SESAM-Modell [SA]**
Ein Simulationsmodell, das durch das *SESAM-System* animiert werden kann. Ein SESAM-Modell besteht aus einem *Schemamodell* und einem dazu passenden *Effektemodell* und *Situationsmodell*.

**SESAM-Projekt [SA]**
Projekt der Abteilung Software Engineering des Instituts für Informatik der Universität Stuttgart, in dessen Rahmen das *SESAM-System* entsteht. Umfasst auch die Forschungstätigkeiten zur Modellerstellung und -validierung.

**SESAM-System [SA]**
Konkrete Implementierung der Konzepte von *SESA*M. Umfasst Aspekte der Modellerstellung und Modellanimation sowie der Spielauswertung. Besteht aus einer Menge von *Werkzeugen*.

**Simulationsschritt [S1/2]**
Der Vorgang der Herstellung einer neuen *Situation* aus der aktuellen *Situation* durch Anwendung der *Effek*te des *Effektemodells*. Bei einem Simulationsschritt wird die *Simulationszeit* um die *Simulationsschrittweite* fortgeschaltet.

**Simulationsschrittweite [S1/2]**
Die Zeitspanne, die zur *Simulationszeit* hinzugezählt wird, wenn ein *Simulationsschritt* durchgeführt wird. Die Simulationsschrittweite definiert damit die Zeiteinheit der Simulation.

**Simulationszeit [S1/2]**
Zeitmarkierung des *Situationsmodel*ls.

**Simulator [S1]**
*Ausführungswerkzeug* in *SESAM*-1. Die Modellanimationsfunktionalität des Simulators übernimmt in *SESAM-2* die *Basismaschin*e, die Benutzerinteraktionsfunktionalität der *Dolmetsche*r.

**Situation [S1/2]**
kurz für *Situationsmodel*l.

**Situation, aktuelle [S1/2]**
Das *Situationsmodell*, das den momentanen *SESAM-Model*lzustand repräsentiert.

**Situationsmodell [S1/2]**
Bestandteil eines *SESAM-Modells*. Repräsentiert den eigentlichen Modellzustand (Szenario), auf den das *Effektemodell* angewandt wird. Es enthält *Instanzen* der im *Schemamodell* eingeführten *Entitätsty*pen und *Relationsty*pen [S1: zuzüglich *Instanzen* der *Eventtypen*]. Damit kann es als *Instanz* des *Schemamodells* aufgefasst werden. Zusätzlich umfasst das Situationsmodell eine *Simulationszei*t.

**Situationsmuster [S1/2]**
Das Situationsmuster gibt an, wie ein Ausschnitt der aktuellen *Situation* aussehen muss, um einen *Effekt* darauf ausführen zu können. Das Situationsmuster wird durch den *Bedingungsblock* eines *Effekts* beschrieben.

**Spiel [S1/2]**
Vorgang der Benutzung der *Ausführungswerkzeuge* durch einen *Spieler* zum Zwecke der (interaktiven) Ausführung eines *SESAM-Model*ls.

**Spieler [S1/2]**
Benutzer des *SESAM-Systems*, der den Projektleiter des simulierten Projekts mimt. Er kommuniziert über das *Ausführungswerkzeug* mit einem *SESAM-Modell*. Dabei kann er *Kommand*os an da*s SESAM-Modell* schicken und von ihm *Nachrich*ten empfangen.

**Spielstand [SA]**
Zustand einer Ausführung eines *SESAM-Modells*. Besteht aus einem *Situationsmodell* und der Menge der momentanen *Instanz*en kontinuierlicher *Effek*te.

**Startsituation [S1/2]**
*Situationsmodell*, das den Anfangszustand eines *SESAM-Model*ls beschreibt.

**Startzeit**
Anfangswert der *Simulationszeit* in der *Startsituatio*n.

**Struktur**
Der durch den *Strukturbedingungsblock* eines *Effekts* beschriebene Teil des *Situationsmusters*. Die Struktur setzt sich damit aus formalen *Entitäten* und ihren Beziehungen untereinander (in Form von formalen *Relationen*) zusammen.

**Struktur, verbotene [SA]**
siehe *Nicht-Struktur.*

**Strukturbedingung [S1/2]**
Bestandteil des *Strukturbedingungsblocks*. Eine Anforderung an die Existenz einer *Komponente* in der aktuellen *Situation*. Die *Komponente* wird durch eine formale *Entität* bzw. durch eine formale *Relation* beschrieben.

**Strukturbedingung, zusätzliche [S2]**
Es ist bei einem kontinuierlichen *Effekt* möglich, im *Strukturbedingungsblock* neue *Komponent*en zu erzeugen (die sogenannten zusätzlichen Strukturbedingungen). Diese *Komponent*en werden direkt vor der Ausführung des *Aktivierungsblocks* erzeugt. Das weitere Vorhandensein dieser *Komponent*en wird dann zur Strukturbedingung und damit zur Voraussetzung der Ausführung des *Aktivblocks*.

**Strukturbedingungsblock [S2]**
Bestandteil des *Bedingungsblocks* eines *Effekts*. Definiert die *Struktur*, die in der aktuellen *Situation* vorgefunden werden muss, damit der *Effekt* ausgeführt werden kann. Um den Strukturbedingungsblock zu erfüllen, müssen alle *Strukturbedingungen* erfüllt sein, d.h. es muss eine *Bindung* des Strukturbedingungsblocks existieren.

**Strukturbedingungsteil [S1/2, veraltet]**
siehe *Strukturbedingungsblock*.

**Strukturmodell [SA]**

Modell, das das Verhalten des *Originals* durch eine Nachbildung seiner Struktur zu imitieren versucht. Dabei werden innere Wirkungszusammenhänge des *Originals* in das Modell übertragen. Dadurch ist eine Analyse des Modellverhaltens anhand des Verhaltens von Teilen der Struktur (z. B. eines *Attributs* einer *Entität*) möglich. Gegensatz: *Verhaltensmodell*.

**Tutor [S1/2]**
Betreuer (Spielleiter) bei Spielen mit dem *SESAM-System*. Aufgabe des Tutors ist die Einführung des *Spielers* in das konkrete *SESAM-Modell*, die Vorbereitung geeigneter *Startsituationen*, die Betreuung des *Spielers* während des *Spiel*s und die Auswertung und Besprechung der *Spiel*ergebnisse mit dem *Spieler.*

**Typverträglichkeit [DA_FRANKE]**
siehe *Konformität*.

**Verhaltensmodell [SA]**
Modell, das das Verhalten des *Originals* durch eine einfache Transformation von Eingaben in Ausgaben zu imitieren versucht. Das *Original* wird dabei als black box modelliert, d. h. die tatsächlichen inneren Wirkungszusammenhänge spielen bei der Modellierung keine Rolle. Gegensatz: *Strukturmodell*.

**Werkzeug [S1/2]**
hier: Oberbegriff für die im *SESAM-System* integrierten Programme. [S1] Werkzeuge in *SESAM-1* sind die *Modellierungswerkzeuge*, der *Simulator* und die *Auswertungswerkzeuge*. Ab der Version 1.2 von *SESAM-1* wurde der *Dolmetscher* in den *Simulator* integriert. [S2] Werkzeuge in *SESAM-2* sind die *Modellierungswerkzeuge*, der *Hochsprachenübersetzer*, die *Basismaschine*, der *Dolmetscher* und die *Auswertungswerkzeuge*.

**Wörterbuch [S2]**
Wird vom *Dolmetscher* verwendet, um zwischen einem *SESAM-Modell* und dem *Spieler* vermitteln zu können. Spezifiziert, welche *Kommandos* das Modell versteht und welche *Nachrichten* es versenden kann. Zu jedem *Kommando* sind die möglichen Eingaben des *Spielers*, die dieses *Kommando* auslösen sollen, genannt. Für jede *Nachricht* des Modells wird der Text angegeben, der beim Erhalt einer *Nachricht* dieses Typs ausgegeben werden soll.

**Zeitverbrauch [S1/2]**
Gibt an, wieviele *Simulationsschritte* die Ausführung einer *Instanz* eines *Effekts* den Projektleiter (den *Spieler*) kostet. Die *Simulationszeit* wird nach der Ausführung der *Instanz* um den Zeitverbrauch erhöht.

## Anhang B

# Literaturverzeichnis


[Boehm, 1981] Boehm, B. W.: Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall 1981.

[Deininger, 1992] Deininger, M.; Lichter, H.; Ludewig, J.; Schneider, K.: Studien-Arbeiten - ein Leitfaden zur Vorbereitung, Durchführung und Betreuung von Studien-, Diplom-, und Doktorarbeiten am Beispiel Informatik. Zürich: vdf Hochschulverlag 1992.

[Drappa, 1997] Drappa, A.: Ada95-Programmierrichtlinien - Richtlinien für die Programmierung in Ada95 für SESAM-2. Internes Dokument, Abteilung Software Engineering, Universität Stuttgart 1997.

[Drappa, 2000] Drappa, A.: Quantitative Modellierung von Softwareprojekten. Dissertation, Shaker, Aachen 2000.

[Gordon, 1969] Gordon, G.: System Simulation. Englewood Cliffs, NJ : Prentice-Hall 1969.

[Hampp, 2001] Hampp, T.: Eine feingranulare SESAM Variante. Diplomarbeit Nr. 1931, Fakultät Informatik, Universität Stuttgart 2001.

[Jones, 1986] Jones, T. C.: Programming Productivity. New York: McGraw-Hill 1986.

[Kalajzic, 2001] Kalajzic, S.: Revision von SESAM-Modellen anhand von Metaregeln. Diplomarbeit Nr. 1941, Fakultät Informatik, Universität Stuttgart 2001.

[Knuth, 1997] Knuth, D., E.: The Art of Computer Programming, Volume 1, Fundamental Algorithms. Addison-Wessley 1997.

[Krauß, 1999] Krauß, S.: Feinentwurf des Teilsystem Execute. Internes Dokument, Abteilung Software Engineering, Universität Stuttgart 1999.

[Ludewig, 1994] Ludewig, J.: SESAM: Grundidee und Überblick. SESAM Software-Engineering-Simulation durch animierte Modelle, Bericht 5/94, Universität Stuttgart 1994.

[Mattern, 1998] Mattern, F.; Mehl, H.: Diskrete Simulation - Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung, Informatik-Spektrum, Springer-Verlag 1989.

[Melchisedech, 1998] Melchisedech, R.: Entwurf der Basismaschine. Internes Dokument, Abteilung Software Engineering, Universität Stuttgart 1998.

[Page, 1991] Page, B.: Diskrete Simulation - Eine Einführung mit Modula2. Berlin: Springer 1991.

[Reißing, 1996, I] Reißing, R.: Konzeption und Realisierung einer Basismaschine für SESAM-2. Diplomarbeit Nr. 1345, Fakultät Informatik, Universität Stuttgart 1996.

[Reißing, 1996, II] Reißing, R.: Begiffslexikon für SESAM. Internes Dokument, Abteilung Software Engineering, Universität Stuttgart 1996.

[Schmidt, 1985] Schmidt, B.: Systemanalyse und Modellaufbau. Berlin: Springer 1985.

[Schneider, 1999] Schneider, B.; Messner, M. P.; Müller, U.: Modellbeschreibungssprache für SESAM-2, Version 3.1. Internes Dokument, Abteilung Software Engineering, Universität Stuttgart 1999.

[Schneider, 1994] Schneider, K.: Ausführbare Modelle der Software-Entwicklung. Struktur und Realisierung eines Simulationssystems. Dissertation. Zürich: vdf 1994.

[Stachowiak, 1973] Stachowiak, H.: Allgemeine Modelltheorie. Wien: Springer Verlag 1973.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____ _____

Jörg Franke