Projektunterstützung durch Verlaufsanalysen und Agenten

Nusser Markus
mnusser@edu.uni-klu.ac.at

April 2003

Diplomarbeit in Angewandter Informatik

durchgeführt am

ISYS – Institut für Informatik-Systeme
der Universität Klagenfurt

Begutachter: O.Univ.Prof. Dipl.-Ing. Mag. Dr. Roland Mittermeir
Betreuer: Univ.Ass. Dipl.-Ing. Andreas Bollin

Danksagung

Ich versichere, diese Arbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubter Hilfsmittel bedient zu haben.

ZUSAMMENFASSUNG

Muss noch genau überlegt werden

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

Kapitel 1

MOTIVATIONSTEIL UND EINLEITUNG

Für ein erfolgreiches Führen von Softwareprojekten bedarf es einer Menge an Erfahrung und Wissen. Die Ausbildung an den Universitäten und Hochschulen im Bereich des Softwareprojektmanagements kann den Studierenden zwar das benötigte Wissen vermitteln, aber die Erfahrung kann nur in der Praxis gesammelt werden. Das Sammeln von Erfahrung ist aber leider ein sehr risikobehafteter und kostspieliger Prozess, den kein Unternehmen bereit ist einzugehen. Mit Hilfe des Ameisesystem ist es nun erstmals möglich, die größe Lücke zwischen der Theorie und der Praxis des Softwareprojektmanagements zu verkleinern. AMEISE steht für „A MEdia Initiative for Software Engineering“ und bietet Studierenden einen Simulator an, mit dessen Hilfe das Managen von Softwareprojekten eigenständig gelernt werden kann. Um den Sutdierenden das eigenständige Lernen zu ermöglichen, wurden Unterstützungswerkzeuge integriert, die den Tutor ersetzen.
Da der Simulator auf empirisch gehaltvollen Daten basiert, kann der Studierende in einem geschützten Umfeld Erfahrungen sammeln, die er später für seine berufliche Praxis verwenden kann. Fehler, die im Simulator begangen werden, besitzen zwar die gleichen Auswirkungen wie in der Praxis und der Studierende kann aus ihnen lernen, aber die erschreckende Folgen für den Betrieb bleiben aus.

Um die besten Resultate bei der Ausbildung im Bereich des Softwareprojektmanagements zu erzielen, sollte meiner Meinung nach, nicht mehr auf das Ameisesystem verzichtet werden, da es nur mit dessen Hilfe möglich ist, eine vollständige Abdeckung, über den zu erlernenden Bereich, zu erzielen.

1.1 Problemstellung

Die projektmanagementbezogene Ausbildung an den Universitäten beschränkt sich vorwiegend auf die Theorie. Die Probleme, die im Softwareprojektmanagement auftreten, werden zwar erklärt, doch Studierenden sind in den mei-

sten Fälle skeptisch, ob diese auch der Wirklichkeit entsprechen. Während der Ausbildung haben die Studenten zwar öfter die Aufgabe kleine Projekte durchzuführen und zu leiten, aber der Umfang dieser Projekt kann nicht mit den Projekten verglichen werden, die in der Praxis vorkommen.

Die Theorie an den Universitäten ist wichtig, kann aber die Praxis des Projektmanagement nicht ersetzen. Nur wenn Personen in der Praxis ein Projekt zu leiten haben, werden die auftretenden Probleme offensichtlich und sie lernen, wie man solche Probleme löst bzw. in weiterer Folge vermeidet.

1.2 Zielgruppe

Zu der Zielgruppe des Ameisesystems zählen vor allem Studenten und Lernende, die sich in der Ausbildung befinden. Die Ausbildung an den Universitäten und Hochschulen im Projektmanagement beschränkt sich nur auf die theoretisch Aspekte des Projektmanagements. Das Ameisesystem soll diesen Mangel an praktischer Projektmanagement Erfahrung vermindern, in dem es Studierenden die Möglichkeit bietet, in einer Simulationsumgebung Projektmanagementerfahrung zu sammeln. Diese Erfahrung ist für den weiteren Werdegang der Studierenden unerlässlich. In dieser Simulationsumgebung kann der Studierende, wie der Pilot in einem Flugsimulator, ein grösseres Softwareprojekt führen. Natürlich kann diese Simulation die Praxiserfahrung nicht ersetzen, es bildet aber einen Grundstock an Wissen, der sich später im praktischen Umgang mit der Thematik des Projektmanagements, als äußerst hilfreich erweist. Fehler in der Ameisesimulation haben nicht so gravierende Auswirkungen, wie Fehler in tatsächlichen Projekten. Die Idee besteht darin, dass jeder Fehler machen muss, um daraus zu lernen.

1.3 Gliederung der Arbeit

Zu Beginn dieser Arbeit wird das Ameise Projekt vorgestellt. Dabei werden speziell die Unterschiede und Gemeinsamkeiten des Ameise- bzw. des Sesamsystems herausgearbeitet. Im nächsten Kapitel wird die Projektunterstützungen in der Praxis und im Ameisesystem vorgestellt. Die Projektunterstützung im Ameisesystem besteht aus mehreren Komponenten: dem Vergleich, dem Ratgeber und dem väterlichen Freund. Dahinter stehen drei Konzepte, die ebenfalls vorgestellt werden. Die Basis für die Vergleichskomponente bilden sogenannte Synchronisationspunkte: spezielle Punkte die während des Spielverlaufs markiert werden, um einen fairen Vergleich zwischen mehreren Spielen zu ermöglichen. Diese Synchronisationspunkte werden dann in Kapitel 4 vorgestellt. Das nächste Kapitel, Kapitel 5, beschäftigt sich mit

dem Markierungsagenten. Diese Komponente ist jene Komponente, die das Markieren von relevanten Punkten während des Spiels durchführt. Um den Vergleich durchführen zu können, werden neben den Synchronisationspunkten auch noch Vergleichsdaten benötigt. Diese Vergleichsdaten werden im Kapitel 6 beschrieben. In Kapitel 7, werden die Eingriffe in das System und die Erweiterungsmöglichkeiten beschrieben. Im 8. Kapitel werden die Implementierung und der Einsatz der entwickelten Komponenten vorgestellt. Dabei wird speziell auf die Clientarchitektur, den Markierungsagent, den Ratgeber und den väterlichen Freund eingegangen. Unterstützt wird die Beschreibung der Komponenten durch Beispiele, die neben der Oberfläche der Komponenten auch deren Einsatz beschreiben. Das letzte Kapitel, Kapitel 9, beinhaltet eine Zusammenfassung und bietet einen Ausblick auf die Zukunft des Ameisesystems.

Kapitel 2

DAS AMEISE PROJEKT

In diesem Kapitel werden folgende Punkte geklärt:

- Was ist SESAM?
- Was verbirgt sich hinter dem AMEISE Projekt?
- Warum wurde das Ameisesystem entwickelt?
- Welche Vorteile bietet das Ameisesystem gegenüber dem Sesamsystem?
- Worin bestehen die Unterschiede und Gemeinsamkeiten zwischen den beiden Systemen?

2.1 Das Sesam System

Das Sesamsystem bietet für zukünftige Projektleiter eine Trainingsumgebung an [Drappa 1999].

Diese Trainingsumgebung bietet dem Spieler folgende Möglichkeiten:

- Dem Spieler wird durch die Simulation gezeigt, wie er Ressourcen verwenden soll.
- Verändert der Spieler die Ressourcen oder den Entwicklungsprozess, dann zeigt die Simulation dem Spieler die Konsequenzen seiner Handlungen.
- Der Spieler wird mit Problemen und Situationen konfrontiert, die bei realen Projekten entstehen.

Um einen flexiblen Simulationsmechanismus bereitzustellen, wurde in Stuttgart eine Modellsprache entwickelt. Bei der Modellsprache handelt es sich um eine regelbasierte Sprache, die aus drei Teilen besteht: dem Typdefinitionsteil,

dem Dateninitialisierungsteil und dem Regelteil. Der erste Teil, der Typdefinitionsteil, beschreibt die vorhandenen Typen und definiert Relationen. Der Dateninitialisierungsteil definiert den internen Zustand der Simulation, durch die Erzeugung von Instanzen der vorher definierten Typen. Der Regelteil beschreibt die Veränderungen, die durch das Voranschreiten der Simulationszeit entstehen. Das Modell hat dabei die Aufgabe, alle möglichen Interaktionen zwischen dem Spieler und dem Modell zu definieren.

2.1.1 Architektur

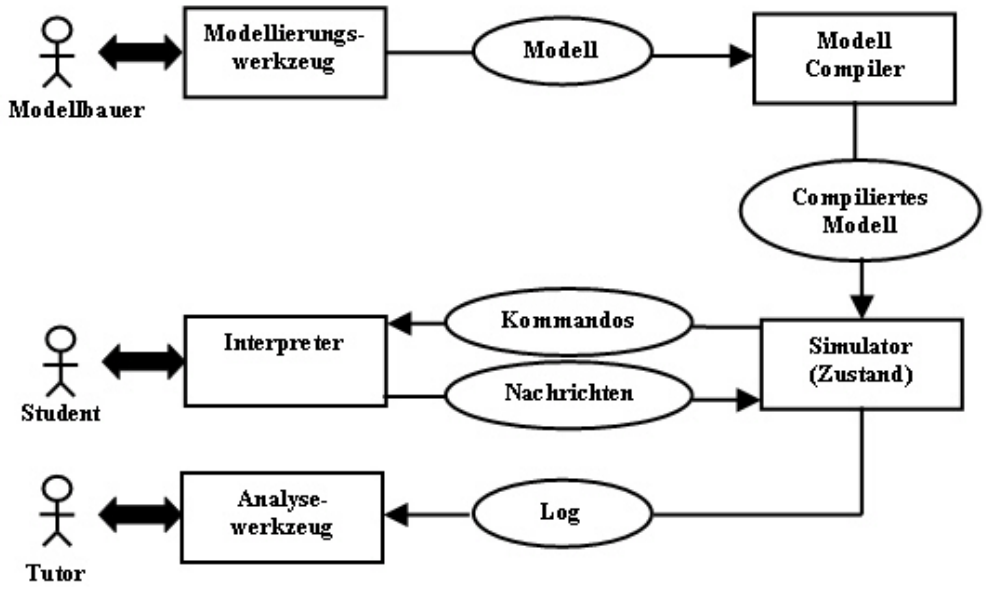

Fig. 2.1: Architektur des Sesamsystems

Die Architektur des Sesamsystems besteht, wie in Abbildung 2.1 ersichtlich ist, aus drei Teilen. Die erste Ebene unterstützt den Modellbauer. Er erstellt mit dem Modellierungswerkzeug ein Modell, welches durch den Modellcompiler compiliert wird. Der Simulator, der sich auf der zweiten Ebene befindet, nimmt dieses compilierte Modell auf und verarbeitet es. Weiters befindet sich im Simulator der aktuelle Spielzustand. Der Student interagiert mit dem System über den Interpreter, der eine annähernd natürliche Sprache erkennt und mit dem Simulator interagiert.

Der aktuelle Zustand des Systems (Spielzustand) stellt einen anotierten Graphen da. Dabei stellt der Regelteil, die erweiterte Grammatik des Gra-

phen dar. Der Simulator identifiziert alle Subgraphen, die mit der linken Seite der Grammatik übereinstimmen und ersetzt diese durch die entsprechende rechte Seite. Dabei werden zusätzlich die Werte der Attribture verändert. Die Interaktionen der Spieler mit dem System stellen einfache Regeln dar, die ausgeführt werden. [Anke Drappa 1999]

Die letzte Ebene der Architektur besteht aus dem Analysewerkzeug und dessen Input. Im Log wird das simulierte Projekt mitprotokolliert. Dieses erstellte Log ist der Ausgangspunkt für die Analysewerkzeuge. Der Output der Analysewerkzeuge muss von einem Tutor aufbereitet werden. Mit Hilfe dieser Informationen kann der Tutor den Spielern Feedback über deren Projekt geben.

2.1.2 QS Modell

Die Daten für das QS Modell (Qualitätssicherungsmodell) stammen aus kleinen bis mittleren Softwareprojekten. Alle Entwicklungsaktivitäten (von der Spezifikation bis zur Übergabe des Produktes) sind im QS Modell enthalten. Weiters beinhaltet das QS Modell die Möglichkeit Reviews von allen Dokumenten und verschiedene Tests durchzuführen. Die Wissensbasis, auf der das QS Modell beruht, wurde hauptsächlich aus der Arbeit von Capers Jones ermittelt. Jones leitet ein Beratungsunternehmen und hatte dadurch Zugang zu 6700 Softwareprojekten [Anke Drappa 1999]. Diese Daten bedurften noch eine Nachbearbeitung, weil eine gewisse Ungenauigkeit in den Daten von Jones vorlag.

Diese Ungenauigkeit lag laut Jones:

- In der ungenauen Zuordnung von Aufwand und Dauer.
- Die Aufwandsabschätzungen wurde verfälscht, weil Überstunden nicht berücksichtigt wurden.
- In fehlenden Abgrenzungen von Aktivitäten.

Wie die genaue Nachbearbeitung der Daten von statten ging, ist in der Dissertation von Anke Drappa [Drappa 1999] nachzulesen.

Der Hauptbestandteil des QS Modells sind Dokumente und Entwickler. Entwickler produzieren, inspizieren oder verbessern unterschiedliche Dokumente des Softwarentwicklungsprozesses. Zugeteilt zu diesen Tätigkeiten werden die Entwickler vom Spieler, der den Projektleiter mimt. Bei der Erstellung, Inspizierung oder der Überarbeitung werden Dokumente erzeugt.

Die Dokumente werden durch ihren Inhalt, ihrer Größe und ihrer Qualität beurteilt. Der Inhalt der Dokumente wird durch die Anforderungen modelliert. Gemessen werden diese Anforderungen durch die Function Point Methode. Diese Function Point Methode ist durch die IFPUG (International Function Point User Group) 1994 standardisiert worden und wird weltweit eingesetzt. Mit Hilfe dieses Ansatzes ist es möglich, unterschiedlich erhaltene Ergebnisse von verschiedenen Phasen quantitativ zu vergleichen. Die Größe der Dokumente entspricht der Anzahl der Seiten eines Dokumentes. Ermittelt wird sie durch die, im Dokument enthaltenen Adjusted Function Points (AFP). Die Qualität eines Dokuments wird durch die Anzahl der Fehler dargestellt.

Die Entwickler stellen die wichtigste Ressource dar. Wichtig bei den Entwicklern sind deren Erfahrungen und deren Kenntnisse. Um für jeden Entwickler ein individuelles Profil zu erhalten, werden die Fähigkeiten (in Bezug auf gewisse Tätigkeiten) und die Kenntnisse kombiniert.

Im Spiel werden die beiden Bestandteile des QS Modells (Dokumente, Entwickler) dynamischen verbunden.

2.2 Das Ameise System

Das Projekt AMEISE („A MEdia Initiative for Software Engineering“) wurde von der Universität Klagenfurt, der Universität Linz und der Universität Wien gegründet. Im Laufe des Projektes, trennte sich die Universität Wien vom Projekt. Ihren Platz hat die Fachhochschule für Telematik in Klagenfurt übernommen.

Das Ameise Projekt verfolgt folgende Projektziele:

- Studenten sollen das Managen von Projekten erlernen
  - Es soll nicht nur das theoretischen Wissen des Projektmanagements gelehrt werden, sondern der Lernende soll in einer Simulationsumgebung die Möglichkeit besitzen, das Gelernte anzuwenden.
  - Zu jedem Spiel soll es Feedbackrunden geben, die begangene Fehler aufzeigen und dem Spieler die Möglichkeit bieten, Fragen zu stellen.
  - Das Ameisesystem stellt dabei einen „Flugsimulator“ für das Softwareprojektmanagement dar.

- Der existierende Prototyp der Universität Stuttgart (SESAM) soll um folgende Punkte erweitert werden:
  - Einführen einer Kommunikationskomponente.
  - Einführen einer Rollback Komponente.
  - Einführen einer Erklärungskomponenten
  - Einführen einer Feedback - Schleife.
  - Erweitern der empirischen Basis für feingranulare Verfahrensschritte.

Zu der Zielgruppe des Ameisesystems zählen:

- Studierende der Informatik, Betriebs- und Wirtschaftsinformatik
- Studierende anderer Ausbildungseinrichtungen mit vergleichbarem Studienprofil

2.2.1 Architektur

Die in der folgenden Abbildung dargestellte Architektur 2.2 stellt das Ameisesystem dar. Die Architektur des Systems ist an die Client - Server Architektur angelehnt, um eine Verteilung der unterschiedlichen Komponenten zu ermöglichen. Der Simulator, der sich im Wrapper befindet, kann dadurch auf einen stärkeren Rechner (Server) untergebracht werden. Die benutzerseitigen Anforderungen an die Hardware sind minimal.

Das Ameisesystem besteht aus folgenden Komponenten:

- Client
- LBM
- Wrapper
- AdminTool
- DB Server
- Datenbank

Im folgenden Abschnitt werden die Komponenten des Ameisesystems kurz erklärt. Eine detailiertere Beschreibung der einzelnen Bestandteile ist auf dem AmeiseServer der Universität Klagenfurt [1] verfügbar.

[1] http:\\ameise.uni-klu.ac.at

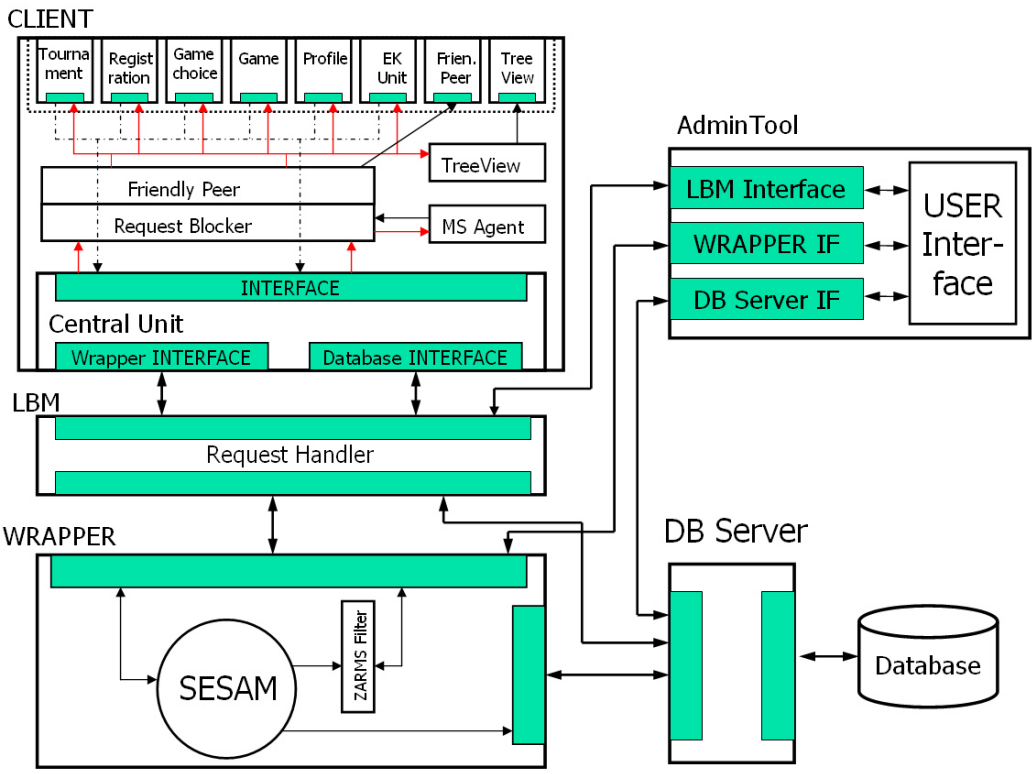

Fig. 2.2: Architektur des Ameisesystems

Client

Die Hauptaufgabe des Ameiseclients besteht darin, den Tutor bei der Durchführung zu unterstützen und dem Spieler das Spiel zu erleichtern. Dadurch wird ein Selbststudium ermöglicht.

Zu den weiteren Aufgaben des Clients zählen:

- Ermöglichen der Client-Server Kommunikation.
  Im Ameisesystem ist es möglich, den Client auf einem eigenständigen Rechner laufen zu lassen. Dieser Rechner muss keine der anderen Komponenten beinhalten. Um dies zu ermöglichen, bedarf es während des Spiels einer ständigen Kommunikation mit den anderen Komponenten. In dieser Kommunikation werden neben den Spielinformationen auch verschiedene Verwaltungsinformationen übermittelt.

- Einbinden der verschiedenen Benutzeroberflächen.
  Das Ameisesystem besitzt neben der Spieloberfläche (Simulationsober-

fläche) auch andere Oberflächen, die den Spieler mit Informationen versorgen. Diese Informationen können den Spieler während des Spiels unterstützen, administrative Informationen anzeigen oder die Möglichkeit bieten, den Spielverlauf zu betrachten.

- Benutzeridentifikation zu ermöglichen.
  Der Spieler muss sich vor einem Spiel am Client anmelden. Da das Ameisesystem sehr stark mit der Datenbank verbunden ist, ist diese Anmeldung notwendig, um den Spieler zu identifizieren und seine Daten in den Simulator zu laden. Weiters wird in der Datenbank ermittelt, ob der Spieler berechtigt ist, sich am Client anzumelden.

- Eine Spielauswahl zu ermöglichen.
  Der Spieler hat im Ameisesystem die Möglichkeit mehrere Spiele, sofern diese vorhanden sind, zu spielen. Diese unterschiedlichen Spiele, werden in der Datenbank gespeichert und können vom Spieler ausgewählt werden.

Die Clientarchitektur wurde im Rahmen der Diplomarbeit überarbeitet und wird im Kapitel 7.1.2 erklärt.

LBM (Load Balance Manager)

Der LBM hat die Aufgabe die Last mehrerer Clients auf mehrere Wrapper zu verteilen. Um die notwendigen Informationen für die Aufteilung zu erhalten, müssen die Wrapper am LBM angemeldet werden. Anhand dieser Information werden Clientanfragen auf die registrierten Wrapper verteilt, um eine gleichmäßige Last auf jedem Wrapper zu erhalten. Eine weitere Aufgabe des LBMs besteht darin, die kommenden Clientnachrichten zu analysieren und diese an die entsprechenden Komponenten weiterzuleiten. Damit wird gewährleistet, dass jede Komponente nur jene Nachrichten erhält, die für sie bestimmt sind. Weiters besitzt der LBM die Aufgabe, Abfragen an die Datenbank durchzuführen. Dieses Abfragen stammen von der EK-Einheit, die am LBM beheimatet ist.

Wrapper

Der Wrapper ist jene Komponente, die den Sesamkern enthält. Dieser Sesamkern wurde aus dem Sesamsystem übernommen. Der Wrapper besitzt die Aufgabe, Abfragen an den Sesamkern zu stellen und die Ergebnisse wieder zurück an den Client und die Datenbank zu übermitteln. Um nur die relevanten Daten in die Datenbank zu übertragen, besitzt der Wrapper einen

Filter (ZARMS Filter), der die Filterung der Daten übernimmt. Diese gefilterten Daten werden von den Hilfsmittelkomponenten des Ameisesystems benötigt.

DB-Server

Der DB-Server stellt die Schnittstelle zur Datenbank dar. Alle Datenbankzugriffe laufen über diese Schnittstelle. Die eingesetzte Datenbank ist die frei erhältliche Open-Source Datenbank „MySQL“.

Database

Die Datenbank besitzt eine zentrale Rolle im Ameisesystem. In ihr werden alle relevanten Daten gespeichtert. Diese Daten betreffen, den Spieler, die Spiele, die Projektunterstützungswerkzeuge, etc.. Eine weitere wichtige Aufgabe besitzt die Datenbank. Da der Sesamkern nicht Mehrbenutzerfähig ist, muss in jedem Spielzug eine Ein- und Auslagerung der Spielerdatei in den Sesamkern durchgeführt werden. Diese Spielerdatei wird in jedem Spielzug aus der Datenbank ausgelesen, in den Sesamkern geladen und nach der Verarbeitung wird die veränderte Spielerdatei wieder in der Datenbank abgelegt.

AdminTool

Beim AdminTool handelt es sich um ein Administrationswerkzeug, mit dessen Hilfe es möglich ist, Daten in der Datenbank zu bearbeiten. Datenbankeinträge können erstellt, verändert und wieder gelöscht werden. Dieses Werkzeug wurde notwendig, um die Administration der Datenbank zu erleichtern. Alle in der Datenbank befindlichen Daten, können über das AdminTool bearbeitet werden.

2.2.2 Eine Ameise Schulung

Um den Spieler den Einstieg in das System zu erleichtern, sollte er zuvor mit der Benutzeroberfläche vertraut werden. Dies kann durch den Lehrveranstaltungsleiter oder Betreuer erfolgen. Ebenso wichtig für den Spieler ist es, den korrekten Umgang mit dem System vorab zu erlernen. Weiters sollen die Antworten des Systems dem Spieler gezeigt werden, damit er dann darauf adäquat reagieren kann. Eine Beschreibung der Ameisebenutzeroberfläche ist im Anhang im Kapitel A ab Seite 92 ersichtlich.

2.3 Relevante Literatur

[Drappa 1999], [Anke Drappa 1999], [Drappa et al. 1995]

Kapitel 3

IDEALE PROJEKTUNTERSTÜTZUNG

3.1 Projektunterstützung in der Praxis

Die Softwareprojekte der Vergangenheit haben immer mit dem Problem der Kosten und Terminüberschreitung gekämpft. Ein weiterer Kritikpunkt, ist die schlechte Qualität der erstellten Software [Anke Drappa 1999]. Trotz der Verbesserung der Softwareentwicklungsmethoden und der Softwaretechnologien hat sich weder der Softwareentwicklungsprozess noch die gelieferte Softwarequalität verbessert. Ein Grund für diese schlechten Resultate liegen in der Ausbildung der Softwareprojektmanager. Eine der Hauptaufgaben eines Softwareprojektmanagers besteht darin, die zeitlichen und finanziellen Aktivitäten und Ressourcen zu organisieren [Cori 1985]. Zu den Ressourcen zählen das Personal, das Material und das Equipment. Um ein guter Softwareprojektmanager zu werden, bedarf es hoher Kosten, die aus Fehlentscheidungen in Projekten resultieren.

3.2 Projektunterstützung im Ameisesystem

Ziel der Projektunterstützung im Ameisesystem ist es, dem Projektleiter (Spieler) eine möglichst breite Hilfestellung und einen großen Motivationsfaktor im Umgang mit dem Ameisesystem zu bieten. Durch die Vergleichskomponente des Ameisesystems wird dem Spieler ein Anreiz geboten, sein Spiel mit dem Spiel anderer Spieler zu vergleichen. Das System hat bei der Unterstützung des Spielers dafür zu sorgen, dass der Spieler mit seinen Fragen und Probleme bezüglich des Spieles nicht alleine dasteht. Dem Spieler soll zusätzlich einen Freund zur Verfügung haben, der ihn bei Problemen mit Ratschlägen unterstützt. Im Ameisesystem wird diese Unterstützung durch den so genannten Ratgeber und den väterlichen Freund realisiert. Zwei ähnliche Konzepte, die beide ein Ziel verfolgen: Die ideale Unterstützung des Spielers.

Folge Komponenten werden im Ameisesystem für die Projektunterstützung herangezogen:

Projektunterstützungswerkzeuge		
Komponente	Aufgabe	Eigenschaften
Markierungsagent	markieren von Spielzügen Blockieren der Eingabe	Hilfskomponente Agent nicht vom Spieler beeinflussbar
Vergleichskomponente	Spielevergleich Motivationsförderung	aktive Komponente vom Spieler beeinflussbar
Ratgeber	Spielerunterstützung	aktive Komponente vom Spieler beeinflussbar
väterlicher Freund	Spielerunterstützung	aktive Komponente Agent nicht vom Spieler beeinflussbar

3.2.1 Gruppen- und Einzelvergleich

Die Vergleichskomponente bietet die Möglichkeit, das aktuelle Spiel des Spielers mit Spielen von einem anderen Spieler oder mit dem Ergebnis von einer Gruppe von Spielern zu vergleichen. Im ersten Fall spricht man von einem Einzelvergleich, der zweite Fall beschreibt den Gruppenvergleich. Da die Ergebnisse des Spielers nicht mit allen Ergebnissen der Gruppe verglichen werden können, kann ausgewählt werden, ob Maximum, Minimum oder der Durchschnitt der Gruppe als Vergleichswert herangezogen werden soll. Diese Auswahl kann der Spieler treffen. Weiters besitzt der Spieler die Möglichkeit, die Gruppen und Einzelpersonen für den Vergleich aus einer vom Tutor vorgegebenen Menge frei zu wählen.

Um die Anonymität der Spieler zu gewährleisten, ist bei der Auswahl nicht der Name des Spielers ersichtlich, sondern ein selbstgewählter Nickname, der keine Rückschlüsse auf den Spieler zulässt.

Der Vergleich findet anhand von Vergleichsdaten statt. Diese Vergleichsdaten sind Daten, mit deren Hilfe man den Projektfortschritt des Spielers messen kann. Eine vollständige Auflistung und mehr Informationen über die identifizierten Vergleichsdaten werden im Kapitel 6 auf Seite 56 angeführt.

Um zwei Spiele vergleichen zu können, werden neben den Vergleichsdaten auch noch Punkte benötigt, die einen fairen Vergleich zulassen. Diese Punkte werden Synchronisationspunkte genannt. Sie kennzeichnen Punkte im Spielverlauf, bei denen ein Vergleich möglich ist. Die Synchronisations-

punkte werden in zwei Gruppen eingeteilt:

- Synchronisationspunkte, die sich mit den Ressourcen (Zeit, Budget) beschäftigen. Ein Punkt im Spielverlauf wird markiert, wenn zum Beispiel 25 % des zeitlichen Budgets verbraucht sind.
- Synchronisationspunkte, die Entwicklungsphasen markieren. Zu diesen Punkten zählt zum Beispiel das Ende der Spezifikationsphase.

Genauere Informationen über die Synchronisationspunkte und eine Auflistung alle Synchronisationspunkte wird im Kapitel 4 auf Seite 25 beschrieben.

Der Aktivierungszeitpunkt des Vergleich entscheidet, um welche Art von Vergleich es sich handelt. Wird der Vergleich im Spiel durchgeführt, so wird von einem Zwischenvergleich gesprochen. Vergleiche, die am Ende eines Spiels durchgeführt werden, werden Endvergleiche genannt. Bei Zwischenvergleich muss der Spieler auswählen, ab welchem Synchronisationspunkt ein Vergleich anhand welcher Eckdaten stattfinden soll. Die Auswahl des Synchronisationspunktes fällt beim Endvergleich weg. Beim Endvergleich muss der Spieler nur auswählen, anhand welcher Eckdaten, der Vergleich durchgeführt werden soll.

Um nicht alle Daten der Spieler für längere Zeit in der Datenbank zu halten, ist geplant, dass Spielerdaten nach einer bestimmte Zeit auf die, für den Vergleich relevanten Daten, komprimiert werden. Der Komprimierungsgrad bestimmt dann, welche Eckdaten für den Endvergleich noch anwendbar sind. Im Kapitel 6 ab Seite 56 werden die Vergleichsdaten für den Einzel- und den Zwischenvergleich vorgestellt.

Der, im folgenden Abschnitt gezeigt Vergleich, stellt einen Zwischenvergleich dar.

Der Vergleichsprozess

Die Abbildung 3.1 zeigt zwei Spielverläufe von zwei Spielern. Diese Spieler sind Spieler A und Spieler B, die unterschiedlich im Projekt stehen. Wobei bei Spieler A und Spieler B die Projekte unterschiedlich weit fortgeschritten sind. Spieler B hat begonnen, den Entwurf zu erstellen, Spieler A muss den bereits erstellten und gereviewten Entwurf nur noch korrigieren, um diese Entwicklungsphase zu beenden. Spieler B stößt den Vergleich an, wobei er das Ende der Korrekturphase als Synchronisationspunkt wählt. Spieler B wählt Spieler A für den Vergleich aus. Da Spieler A diese Phase bereits abgeschlossen hat, ist der Vergleich zulässig und wird durchgeführt. Die Ergebnisse des Vergleichs werden Spieler B nach deren Auswertung präsentiert. Die

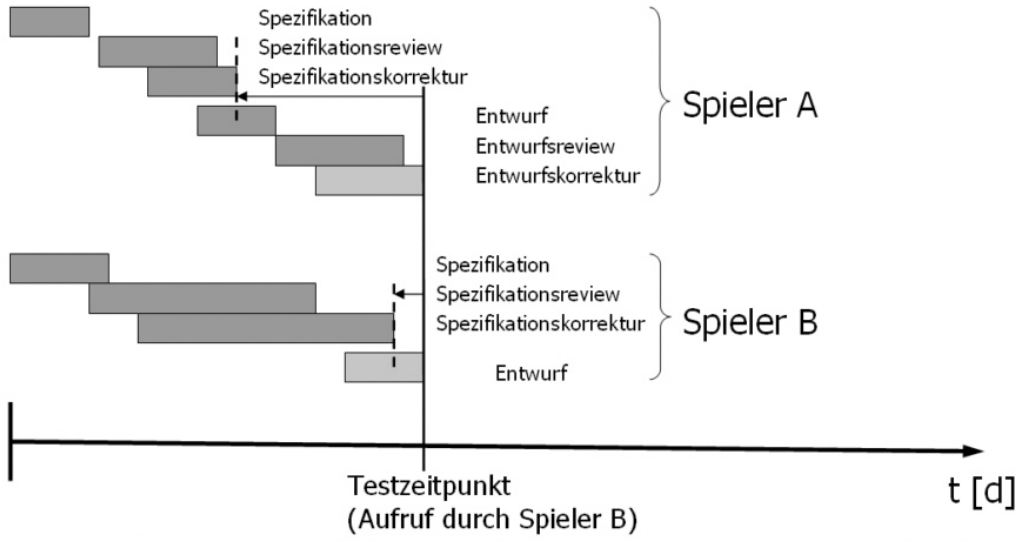

Fig. 3.1: Vergleich von Spieler B mit Spieler A

Präsentation der Vergleichsergebnisse (Diagramme und Text / Text) hängt von den gewählten Eckdaten ab.

Um Probleme beim Vergleichen zwischen zwei Spielen zu vermeiden, muss der Vergleich wie folgt durchgeführt werden:

Der Zwischenvergleich

1. Der Spieler, der den Vergleich durchführen möchte, muss den Synchronisationspunkt wählen, ab dem ein Vergleich durchgeführt werden soll.

2. Nach dem der Synchronisationspunkt festgelegt wurde, wird in mittels einer Auswahlliste jene Spieler angezeigt, mit denen ein Vergleich möglich ist. Hat der Spieler diesen Synchronisationspunkt noch nicht überschritten, so scheint er nicht in der Liste auf und kann somit auch nicht für den Vergleich gewählt werden.

3. Nach der Spielerauswahl werden die Eckdaten gewählt, anhand derer ein Vergleich durchgeführt werden soll. Dabei kann nur ein Eckdatum oder eine Menge an Eckdaten gewählt werden.

4. Jetzt sind alle Daten für den Vergleich bereit und er kann durchgeführt werden.

5. Die ermittelten Daten werden dem Spieler, der den Vergleich durchgeführt hat, angezeigt. Dieses Anzeigen kann sowohl in textueller, wie auch in textuell/graphischer Form erfolgen. Die Auswahl der Anzeige hängt nur von den gewählten Eckpunkten ab.

Der Endvergleich

Der Endvergleich unterscheidet sich vom Zwischenvergleich nur durch das Wegfallen der Synchronisationspunktauswahl. Beim Endvergleich wird immer vom Endpunkt (abgeschlossenes Spiel) ausgegangen. Die Vorgehensweise beim Anstoßen unterscheidet sich sonst nicht von der Vorgehensweise des Zwischenvergleichs.

3.2.2 Ratgeber

Um den Spieler während des Spieles zu unterstützen, wurde der Ratgeber implementiert. Da der Ratgeber nur über einen begrenzten Antwortsatz verfügt, muss der Spieler aus einer Menge von Fragen selbst wählen, welchen Ratschlag der Ratgeber erteilen soll. Diese Fragen werden dem Spieler anhand einer Ratschlagliste präsentiert. Wählt der Spieler einen Ratschlag aus der Liste der Fragen aus, so antwortet der Ratgeber auf diese Frage. Der Ratgeber besitzt eine eigene Oberfläche, die die Liste der Fragen beinhaltet und in der die Antworten dem Spieler präsentiert werden. Detailliertere Informationen über den Ratgeber werden im Kapitel 8.1.3 auf Seite 80 präsentiert.

Die Ratschläge, die der Ratgeber erteilen kann, sind Momentaufnahmen, d.h. sie beziehen sich nur auf den aktuellen Zustand des Systems. Er besitzt nicht die Möglichkeit über Tätigkeit Aussagen zu treffen, die über einen längeren Zeitraum beobachtet werden müssen.

Diese Liste der möglichen Ratschläge entstand aus der Analyse von Ameise / Sesamspielen vom Sommersemester 2002 in der Lehrveranstaltung „Systementwicklungsprozess“. Dabei wurden speziell nach Möglichkeiten gesucht, das Spiel eines Spielers zu verbessern.

Zu den Ratschlägen, die der Ratgeber erteilen kann, zählen folgende Punkte:

1. Auf die Vorgängerphasen wurde vergessen.
   Wenn Vorgängerphasen vergessen werden, kann nicht korrekt mit der Nachfolgephase begonnen werden, da Phasen immer ein Vorgängerdokument benötigen, auf das sie aufbauen. Beim Fehlen dieses Dokuments ist die Phase zum Scheitern verurteilt.

2. Die Vorgängerphasen wurden noch nicht beendet.
   Für die Nachfolgerphase ist es wichtig, dass die Vorgängerphase und damit das entstandene Vorgabedokument bereits mehr als 50% fertig ist, um eine Überlappung der Phasen zu ermöglichen. Die Grenze von 50% ist modellabhängig (QS Modell). Ist dies nicht der Fall, scheitert die begonnene Phase.

3. Die Vorgängerphasen wurden noch nicht überprüft.
   Um auf die Ergebnisse von Phasen aufbauen zu können, ist es wichtig, dass diese Phasen bereits überprüft wurden. Nach der Überprüfung weiss der Spieler, dass das Ergebnis dieser Phase noch Fehler beinhaltet. Diese Fehler müssen aber unbedingt noch korrigiert werden, um diese Fehler nicht mit in die nächste Phase zu übernehmen.

4. Die Vorgängerphasen wurden noch nicht korrigiert.
   Nach der Überprüfung ist die Korrektur sehr wichtig, um die gefundenen Fehler auszubessern. Ohne der Korrektur werden die Fehler in die Nachfolgephasen übernommen.

5. Nach dem Durchführen von Tests, wurde auf die Korrektur vergessen.
   Neben dem Review, kann der Spieler auch durch Tests Fehler in Dokumenten finden. Die dort gefundenen Fehler müssen ebenso korrigiert werden, um ein korrekteres Dokument zu erhalten.

6. Die erneute Überarbeitung von Phasen vorschlagen.
   Im Spiel kommt es oft vor, dass einzelne Phasen noch zu wenige Anforderungen erfüllen, oder zu viele Fehler beinhalten. Wird diese Situation vom Ratgeber erkannt, so kann er darauf reagieren und dem Spieler einen Ratschlag zur Verbesserung der Phase liefern.

7. „Alleskorrektur“ vorschlagen.
   Die „Alleskorrektur“ drückt im Ameisespiel die Möglichkeit des Spielers aus, alle Dokumente erneut zu korrigieren.

Der erteilte Ratschlag kann für den Spieler sowohl positiv, als auch negativ sein. Diese Ausprägung hängt vom aktuellen Zustand, der in der Frage vorkommenden Tätigkeit oder Phase ab. Wird zum Beispiel gefragt, ob die Spezifikationsphase noch einmal überarbeitet werden muss, hängt dies von der Vollständigkeit der Spezifikationsphase ab, welche Meldung dem Spieler gezeigt wird. Genügt die Spezifikationsphase bereits den Anforderungen, wird das dem Spieler mitgeteilt, tritt aber der andere Fall ein, dann wird ihm ebenso mitgeteilt, welche Auswirkungen diese Situation, auf das Spiel besitzt.

Es kann auch vorkommen, dass der Ratgeber zu einer Frage keine Aussage treffen kann, weil z.B. die zu kontrollierende Phase noch nicht begonnen wurde. In diesem Fall wird diese Situation vom Ratgeber erkannt und eine dementsprechende Meldung erscheint. Diese Meldung teilt dem Spieler mit, dass nicht genügend Daten für diese Auswertung zur Verfügung stehen.

3.2.3 Väterliche Freund

Der väterliche Freund ist die zweite Komponente, die erstellt wurde, um den Spieler mit Hinweisen während des Spiels zu unterstützen. Der väterliche Freund unterscheidet sich vom Ratgeber dadurch, dass er vom Spieler nicht aufgerufen werden kann. Er wurde als Agent realisiert, dessen Aufgabe darin besteht, dem Spieler über die Schulter zu schauen und auf begangenen Fehler hinzuweisen. Die Realisierung, des väterlichen Freundes, findet sich im Kapitel 8.1.4 auf Seite 8.1.4 wieder.

In der Tatsache, dass der väterliche Freund nicht aufgerufen werden kann, liegt auch der größte Unterschied aus Spielersicht zwischen ihm und dem Ratgeber. Der Spieler kann nicht bestimmen wann der väterliche Freund einen Ratschlag anzeigt, um welche Art von Ratschlag es sich handelt und er kann den väterlichen Freund nicht aus- oder einschalten. Der größte Unterschied aus technischer Sicht zwischen dem väterlichen Freund und dem Ratgeber besteht darin, dass er auf Situationen aufmerksam machen kann, zu deren Beantwortung es eines Gedächtnisses bedarf. Dieses Gedächtnis wird im Lauf des Spieles aufgebaut. Ein Beispiel für einen solche Situation ist die Mitarbeiterauslastung. Bei der Mitarbeiterauslastung muss der väterliche Freund bei allen eingestellten Mitarbeitern deren Tätigkeitsdauer beobachten. Kommt es vor, dass ein Mitarbeiter mehrere Tage keiner Tätigkeit nachgeht, muss der Spieler davon in Kenntnis gesetzt werden. Für diesen Hinweis ist das Gedächtnis erforderlich.

Nicht alle Hinweise des väterlichen Freundes benötigen eine derartiges Gedächtnis. Die meisten Abfragen, die er durchführt, können aus der Datenbank beantwortet werden.

Die Möglichkeit den väterlichen Freund zu verwenden, oder nicht, wird vom LV Leiter zu Beginn des Spieles festgelegt und kann während des Spieles nicht mehr verändert werden.

Der väterliche Freund hat eine bestimmte Anzahl an Situationen, bei deren Eintreten er reagieren kann.

Eine weitere Eigenschaft unterscheidet den väterlichen Freund vom Ratgeber. Der väterliche Freund teilt dem Spieler Hinweise nur mit, wenn die Ereignisse, auf die er lauscht, auch tatsächlich eingetreten sind. Wenn keines der Ereignisse eintritt, bekommt der Spieler während des gesamten Spiels

keine Meldung vom väterlichen Freund.

Die folgende Auflistung beinhaltet Aktivitäten, bei deren Eintreten der väterliche Freund dem Spieler einen Ratschlag erteilen soll. Sie entstand aus der Analyse von Ameise / Sesamspielen vom Sommersemester 2002 in der Lehrveranstaltung „Systementwicklungsprozess“.

1. Den Autor als Gutachter eingesetzt. Kein Gedächtnis erforderlich.
   Ist der Autor Gutachter seines eigenen Dokuments, so findet er darin keine Fehler. Für andere Entwickler ist es leichter Fehler im Dokument zu entdecken.

2. Der Autor sollte die Korrektur durchführen. Kein Gedächtnis erforderlich.
   Nach dem beim Review oder beim Test Fehler entdeckt wurden, ist es die Aufgabe des Autors dieser Fehler im Dokument zu korrigieren, da er das Dokument erstellt hat und dadurch keine Einarbeitungszeit benötigt. Ebenso werden nur wenige neue Fehler dem Dokument hinzufügt.

3. Auf die Vorgängerphasen wurden vergessen. Kein Gedächtnis erforderlich.
   Wenn Vorgängerphasen vergessen werden, kann nicht korrekt mit der Nachfolgephase begonnen werden, da Phasen immer ein Vorgängerdokument benötigen, auf das sie aufbauen. Beim Fehlen dieses Dokuments ist die Phase zum Scheitern verurteilt.

4. Die Vorgängerphasen wurden noch nicht beendet. Kein Gedächtnis erforderlich.
   Für die Nachfolgerphase ist es wichtig, dass die Vorgängerphase und damit das entstandene Vorgabedokument bereits mehr als 50% fertig ist, um eine Überlappung der Phasen zu ermöglichen. Ist dies nicht der Fall, scheitert die begonnene Phase.

5. Vorgängerphasen wurden noch nicht überprüft. Kein Gedächtnis erforderlich.
   Um auf die Ergebnisse von Phasen aufbauen zu können ist es wichtig, dass diese Phasen bereits überprüft wurden. Nach der Überprüfung weiß der Spieler, dass das Ergebnis dieser Phase noch Fehler beinhaltet. Diese Fehler müssen aber unbedingt noch korrigiert werden, um diese Fehler nicht mit in die nächste Phase zu übernehmen.

6. Vorgängerphasen wurden noch nicht korrigiert. Kein Gedächtnis erforderlich.

Nach der Überprüfung ist die Korrektur sehr wichtig, um die gefundenen Fehler auszubessern. Ohne der Korrektur werden die Fehler in die Nachfolgephasen übernommen.

7. Ein erneuter Review soll erst nach einer erfolgreichen Korrektur stattfinden. Kein Gedächtnis erforderlich.
   Wurde ein Review durchgeführt und wurde auf die Korrektur vergessen, ist ein erneuter Reviewversuch nicht erfolgreich. Dieser Reviewversuch wird ohne Ergebnis beendet, aber die Mitarbeiter haben einen Tag vergebens daran gearbeitet. Somit hat der Review Kosten verursacht, aber nicht zu einer besseren Qualität beigetragen.

8. Nach den Tests nicht auf die Korrektur vergessen. Kein Gedächtnis erforderlich.
   Neben dem Review, kann der Spieler auch durch Tests Fehler in Dokumenten finden. Die dort gefundenen Fehler müssen ebenso korrigiert werden, um ein korrekteres Dokument zu erhalten.

9. Der Kunde soll bei den Reviews der Anfangsphasen miteinbezogen werden. Kein Gedächtnis erforderlich.
   Der Kunde kennt die Anforderungen an das System am besten. Daher ist es für einen Review, in den Anfangsphasen (Spezifikation, Handbuch), von großem Vorteil den Kunden am Review zu beteiligen. Der Kunde verbessert das Ergebnis merklich (35%) und kostet dem Projektleiter nichts.

10. Mit dem Handbuch soll bereits in den Anfangsphasen begonnen werden. Kein Gedächtnis erforderlich.
    Da die Erstellung des Handbuches auch als Review der Spezifikation angesehen wird, ist es sinnvoll mit dem Handbuch bereits in den Anfangsphasen zu beginnen. Bei den Reviewphasen werden mehr Fehler gefunden, wenn bereits mit der Erstellung des Handbuches begonnen wurde.

11. Keine häufigen Personalwechsel. Gedächtnis erforderlich.
    Wenn es zu häufigen Personalwechsel kommt, ist der Einarbeitungsaufwand der neuen Mitarbeiter extrem hoch. Die neuen Mitarbeiter müssen sich zuerst in das Projekt einarbeiten, um Projektaufgaben erledigen zu können.

12. Die Mitarbeiterbeschäftigung /-auslastung soll beobachtet werden. Gedächtnis erforderlich.

Da die Mitarbeiter sehr viel Geld kosten, ist es für den Projektmanager (Spieler) sehr wichtig die Mitarbeiterauslastung im Auge zu behalten. Dies zählt wohl zu einer der größten Aufgaben eines Projektleiters, die Mitarbeiter, die er in das Projektteam aufgenommen hat und selbstverständlich dafür zahlen muss, immer so zu beschäftigen, dass es zu keinen hohen Leerlaufzeiten kommt. Natürlich ist es nicht immer möglich die Mitarbeiter rund um die Uhr zu beschäftigen, aber längere Leerlaufzeiten sollen unbedingt vermieden werden.

13. Personaleinsatz in einer Phase beobachten. Gedächtnis erforderlich.
Der Spieler wird bei der Planung der einzelnen Aktivitäten immer vor die Wahl gestellt, wie viele Mitarbeiter er für die Phasen verwenden will. Werden zu viele Mitarbeiter in der Aktivität beschäftigt, ist der Kommunikationsaufwand zwischen den Mitarbeitern enorm, werden zu wenig Mitarbeiter eingesetzt, dauert die Phase zu lange. Zwei Mitarbeiter, benötigen ähnlich lange für eine Tätigkeit, wie drei Mitarbeiter. Drei Mitarbeiter kosten um einiges mehr, sind aber nur geringfühig effektiver. Ein klarer Fehler des Projektleiters ist es, wenn er zu einer Tätigkeit mehr als drei Mitarbeiter einteilt. So benötigen beispielsweise vier Mitarbeiter für die selbe Tätigkeit länger als zwei Mitarbeiter. Grund hierfür ist der oben bereits erwähnte Kommunikationsaufwand im Team.

14. Die Mitarbeiter sollen nicht vorzeitig aus einer Phase entfernen werden, da ein hoher Aufwand entsteht, wenn neue Mitarbeiter, deren Tätigkeit übernehmen sollen. Kein Gedächtnis erforderlich.
Wenn einzelne Mitarbeiter für Tätigkeiten in Phasen länger benötigen, ist es nicht sinnvoll, diesen Mitarbeiter von seiner Aktivität abzuziehen. Neue Mitarbeiter, die dessen Aufgabe fortführen sollen, benötigen eine Einarbeitungszeit, bevor sie die Aufgabe übernehmen können. Deshalb ist es besser, im Voraus nur die besten Mitarbeiter, für die Phasen einzuteilen. Dies ist leider nicht immer möglich, aber falls die Mitarbeiterwahl einmal nicht der Beste war, sollte man diesen, nicht so optimalen Mitarbeiter, seine Tätigkeit beenden lassen.

15. Die Entlassung der Mitarbeiter bedenken, wenn sie später im Projekt wieder benötigt werden. Kein Gedächtnis erforderlich.
Natürlich sind Mitarbeiter sehr teuer, aber man sollte die Entlassung aus dem Projekt sehr genau durchdenken, vor allem dann, wenn sie in späteren Phasen wieder benötigt werden. Im Sesam- / Ameisesystem ist der Mitarbeiter nach seiner Entlassung zwischen 1 und 60 Tage [Anke Drappa 1999] nicht mehr verfügbar.

16. Qualifikation des Autors passt nicht zu der gerade durchgeführten Tätigkeit. Kein Gedächtnis erforderlich.
    Für eine Tätigkeit sollte immer der beste, gerade verfügbare Mitarbeiter eingesetzt werden, da sonst unnotwendige Verzögerungen entstehen können.

3.3 Beispiele

Im folgenden Abschnitt werden für den väterlichen Freund und den Ratgeber Implementierungsbeispiele angeführt.

3.3.1 Ratgeber

Um die Daten für einen Ratschlag zu erhalten, bedient sich der Ratgeber der Architektur der speziellen Hilfsmittelkomponente. Die Architektur und Arbeitsweise der speziellen Hilfsmittelkomponente werden in Kapitel 6.2 ab Seite 58 erklärt.

Das hier angeführte Beispiel beschreibt das Auslesen der Anzahl der AFPs der Spezifikationsphase.

```
SELECT s_entity.value
FROM  zarmstype,z_entity,z_attribute,
      comprises,s_entity,game,
      spaid_needs_zt,specific_aid
WHERE
      spaid_needs_zt.spaidid = specific_aid.spaidid AND
      spaid_needs_zt.zid = zarmstype.zid AND
      zarmstype.eorr = "E" AND
      zarmstype.z_type = "SPEZIFIKATION" AND
      zarmstype.zid = z_entity.zid AND
      z_entity.description = "Specification" AND
      z_entity.zeid  = comprises.zeid AND
      comprises.zaid = z_attribute.zaid AND
      z_attribute.name = "ANZ_AFP" AND
      comprises.compid = s_entity.compid AND
      s_entity.gid = game.gid AND
      game.gid = 112 AND
      s_entity.path = 1;
```

3.3.2 väterlicher Freund

Der väterliche Freund verwendet wie der Ratgeber ebenso die Architektur der speziellen Hilfsmittelkomponente. Die Architekturbeschreibung wird in Kapitel 6.2 ab Seite 58 dargestellt.

Das angegebene Beispiel des väterlichen Freunds ermittelt das Datum des Beginns der Spezifikationsphase.

```
SELECT s_entity.value
FROM  zarmstype,z_entity,z_attribute,
      comprises,s_entity,game,spaid_needs_zt,
      specific_aid
WHERE
      spaid_needs_zt.spaidid = specific_aid.spaidid AND
      spaid_needs_zt.zid = zarmstype.zid AND
      zarmstype.eorr = "E" AND
      zarmstype.z_type = "PROJEKTLOGBUCH" AND
      zarmstype.zid = z_entity.zid AND
      z_entity.description = "Projectlog" AND
      z_entity.zeid  = comprises.zeid AND
      comprises.zaid = z_attribute.zaid AND
      z_attribute.name = "ENTWURF_BEGINN" AND
      comprises.compid = s_entity.compid AND
      s_entity.gid = game.gid AND
      game.gid = 112 AND
      s_entity.path = 1;
```

3.4 Relevante Literatur

[Anke Drappa 1999],[Cori 1985]

Kapitel 4

SYNCHRONISATIONSPUNKTE

Um die Spiele von zwei oder mehreren Spielern vergleichen zu können, benötigt man Punkte im Projektverlauf, an denen ein Vergleich durchgeführt werden kann. Diese Punkte werden im Folgenden Synchronisationspunkte genannt.

Synchronisationspunkte können auf Grund ihrer Herkunft in zwei große Gruppen eingeteilt werden:

1. Synchronisationspunkte, die Projekt und Entwicklungsphasen abdecken und

2. Synchronisationspunkte, die sich mit den Ressourcen beschäftigen.

Die erste Gruppe von Synchronisationspunkten befasst sich mit jenen Punkten, die aus den Projekt- und Entwicklungsphasen resultieren. Zu ihnen zählen z.B. das Ende der Spezifikationsphase, das Ende der Entwurfsphase, usw. Die Unterscheidung zwischen Markierungspunkten und Synchronisationspunkten, die Entwicklungsphasen abdecken ist einfach. Markierungspunkte markieren den Beginn und das Ende der Entwicklungsphasen, zu den Synchronisationspunkten hingegen zählen nur jene Markierungspunkte, die nur das Ende von Entwicklungsphasen markieren. Die zweite Gruppe von Synchronisationspunkten befasst sich mit den Punkten, die während eines Spieles entstehen, wenn gewisse Ressourcegrenzen überschritten werden. Beispiele hierfür wären: 50% des Budgets verbraucht, 25% der maximalen Projektdauer überschritten, usw. Zwischen den Markierungspunkten die Ressourcen markieren und den Synchronisationspunkten die Ressourcen markieren, gibt es keinen Unterschied. Also sind Ressourcemarkierungspunkte auch gleich Ressourcesynchronisationspunkte.

Der in Abbildung 4.1 dargestellte Kreis stellte die Menge an Markierungspunkte dar. Diese Menge ist unterteilt in die Menge der Ressourcemarkierungspunkte und jenen Markierungspunkten, die die Entwicklungsphasen abbilden. Die grauen Flächen stellen die Synchronisationspunkte dar.

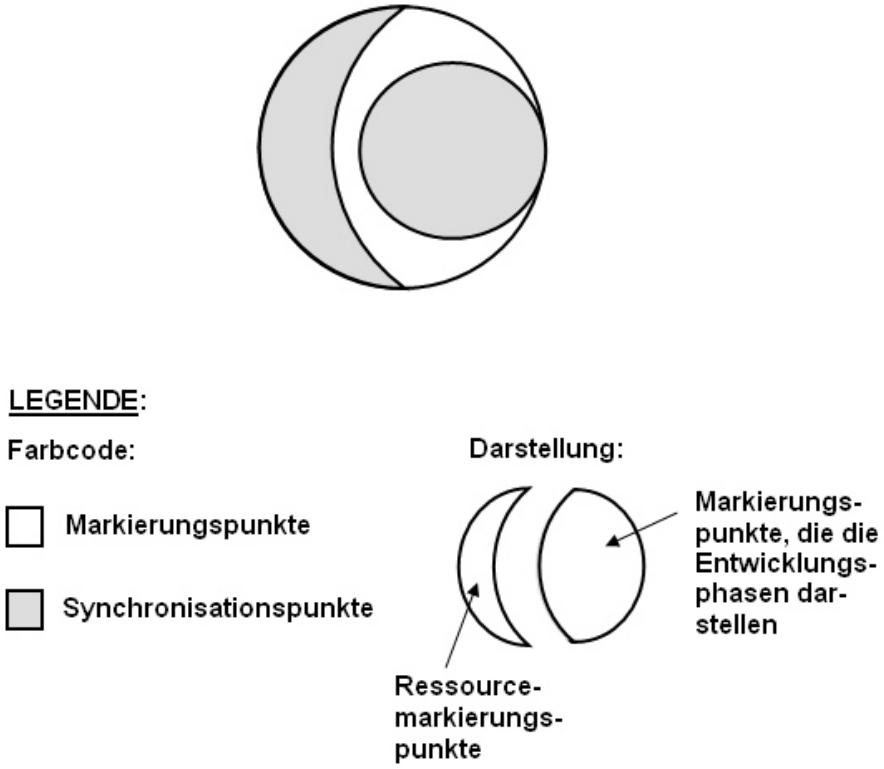

Fig. 4.1: Darstellung der Synchronisations- und Markierungspunkte

4.1 Projekt- und Entwicklungsphasen

Wie in den obigen Kapiteln bereits erwähnt wurde verwendet das Ameisesystem, das in Stuttgart erstellte QS Modell. Dieses QS Modell ist an das, von Winston Royce (1970) definierte, Wasserfallmodell angelehnt. Um eine möglichst große Flexibilität im QS Modell zu erzielen, ist es aber auch möglich, nicht nach dem Wasserfallmodell vorzugehen. Die Starrheit des Wasserfallmodells wurde in diesen Punkten aufgebrochen. So ist es im QS Modell zum Beispiel möglich, Phasen auszulassen oder Rücksprünge auf frühere Phasen zu tätigen. Rücksprünge sind im Modell nur dann möglich, wenn die Kundenanforderungen noch nicht vollständig berücksichtigt wurden.

Die folgende Grafik zeigt eine Auslistung der, im QS Modell befindlichen, Entwicklungsphasen und deren Verflechtung.

In Abbildung 4.2 werden die möglichen Abläufe des Ameisespiels dargestellt. In der Abbildung stellen die strichlierten Pfeile keine Abhängigkeit

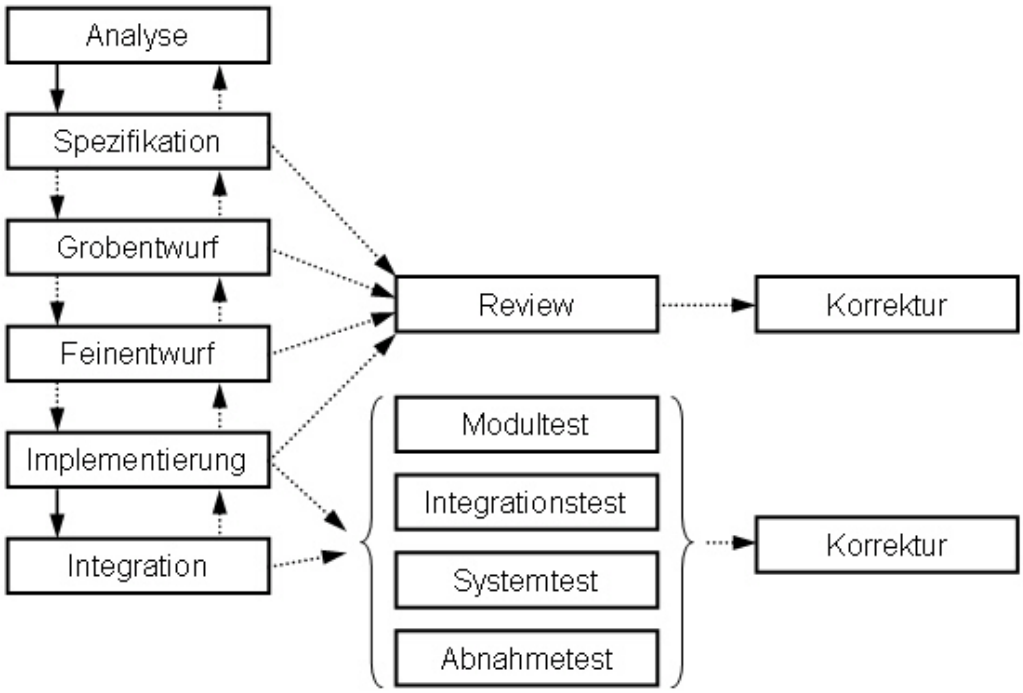

Fig. 4.2: Moegliche Ablaeufe im Sesammodell (Quelle [Drappa 1999]

zwischen den Phasen dar. So kann beispielsweise der Feinentwurf vor dem Grobentwurf stattfinden. Sind Aktivitäten durch einen, nicht strichlierten Pfeil verbunden, dann gibt es zwischen diesen Aktivitäten eine Reihenfolge. Die Integration kann laut Abbildung 4.2 nur nach der Implementierung erfolgen.

Weiters geht aus der Abbildung hervor, dass der Review und die Korrektur keinerlei zeitliche Einschränkungen unterworfen sind. Der Spieler kann selbst bestimmen, wann er einen Review oder einen Korrektur durchführt.

Obwohl die Entwicklungsphasen bereits in der Abbildung 4.2 enthalten sind, werden sie hier noch einmal erwähnt. Das QS Modell besitzt folgende Phasen:

- Spezifikationsphase
- Grobentwurfsphase
- Feinentwurfsphase
- Implementierungsphase

- Integrationsphase

Wenn der Spieler mit dem QS Modell zu spielen beginnt, dann bemerkt er, dass die Analyse bereits durchgeführt wurde. Aus diesem Grund wird sie nicht als Entwicklungsphase im Ameisesystem betrachtet.

4.1.1 Modellabhängige Beispiele

Wenn man von modellabhängigen Beispielen spricht, so haben die Synchronisationspunkte einen Bezug zum gerade verwendeten Modell. Im Ameisesystem wird das QS Modell verwendet. Wird ein anderes Modell eingesetzt, existieren diese Punkte nicht mehr unbedingt, sondern müssen vom Modellbauer erneut auf ihre Plausibilität geprüft werden. So gibt es für jedes Modell eine bestimmte Klasse von Punkten, die nur in diesem Modell ihre Gültigkeit besitzen.
Im QS Modell existieren folgende Synchronisationspunkte:

- Ende von Entwicklungsphasen.
- Ende des Review der Entwicklungsphasen.
- Ende der Korrektur der Entwicklungsphasen.

Die Entwicklungsphasen, die im QS Modell existieren, wurden bereits im Kapitel 4.1 beschrieben.

Synchronisationspunkte, die sich mit den Ressourcen beschäftigen, sind nicht modellabhängig. Zu den Ressourcen, die vom Markierungsagent markiert werden, zählen im Ameisespiel das Budget und die verbrauchte Zeit. Andere Ressourcen werden zur Zeit nicht markiert.

4.1.2 Modellunabhängige Beispiele

Zu den modellunabhängigen Beispielen zählen jene Synchronisationspunkte, die unabhängige vom verwendeten Modell existieren. Diese Punkte verlieren auch bei einem Modellwechsel nicht ihre Gültigkeit. Zu dieser Art von Punkte zählen jene Synchronisationspunkte, die sich mit den Ressourcen beschäftigen. Da es bei jedem Modell Ressourcen gibt, verlieren die Ressourcesynchronisationspunkte auch bei einem Modellwechsel nicht ihre Gültigkeit. Im QS Modell sind die Ressourcesynchronisationspunkte die verbrauchte Zeit und das verbrauchte Budget. Diese Ressourcen existieren in allen Modellen.

4.2 Synchronisationspunkte in Sesam / Ameise

Die Synchronisationspunkte, die im Sesam- /Ameisespiel verwendet werden, beinhalten modellabhängige Synchronisationspunkte und modellunabhängige Synchronisationspunkte. Die modellabhängigen Synchronisationspunkte beziehen sich auf das, in Sesam / Ameise verwendete „QS“ Modell. Siehe dazu auch Kapitel 2.1.2 auf Seite 6. Dieses Modell wurde bereits bei etlichen Spielen verwendet. Zusätzlich zum „QS“ Modell existiert auch noch eine Verfeinerung des „QS“ Modells, aber dieses Modell hat auf Grund des hohen zeitlichen Spielaufwands den Weg noch nicht in die Lehre gefunden. Nähere Informationen über diese Verfeinerung des „QS“ Modells kann aus der Arbeit von Tilmann Hamp [Hampp 2001] entnommen werden.

Synchronisationspunkte, die das Überschreiten einer Ressourcengrenze anzeigen, sind die modellunabhängigen Synchronisationspunkte. Diese Synchronisationspunkte beziehen sich nicht direkt auf das „QS“ Modell, sondern finden auch in anderen Modellen ihre Verwendung.

4.2.1 Identifikation von Synchronisationspunkten in Sesam

Um im Ameise Spiel Spielstände synchronisieren zu können, wurden die Entwicklungsphasen und die Ressourcesynchronisationspunkte herangezogen. Bei den Entwicklungsphasen kam für einen Vergleich nur die Enden der jeweiligen Phasen in Frage. Der Anfang der Phasen ist für den Vergleich völlig belanglos, da noch keine Daten zu dieser Phase existieren. Nur am Ende der Phase können verwendbare Aussagen über sie getroffen werden. Der Markierungsagent besitzt die Aufgabe, diese Punkte als Markierungspunkte zu markieren.

Die Abbildung 4.3 zeigt eine typische Markierung eines Spielverlaufs. Bei diesem Markierungsbeispiel wurden nur die Phasenenden vom Markierungsagent markiert. Genauere Informationen über den Markierungsprozess werden im Kapitel 5 auf Seite 31 beschrieben.

Die Markierung von Synchronisationspunkte wird durch den Markierungsagent durchgeführt. Er besitzt bei Synchronisationspunkten und bei Markierungspunkten die gleiche Aufgabe. Seine Aufgabe besteht darin, den Systemzustand ständig zu überwachen. Der Zustand des Systems wird bei jedem „Proceed“ in die Datenbank übertragen. Dort ist es für den Markierungsagent einfach, die benötigten Daten auszulesen. Neben der Datenbank benötigt der Markierungsagent für seine Arbeit auch noch eine spezielle Datenstrukturen, die seine Logik steuern. Die exakte Erklärung der Arbeit des Markierungsagenten wird im Kapitel 5 auf Seite 31 genauer erklärt.

Die andere grosse Gruppe von Synchronisationspunkten, die vom Mar-

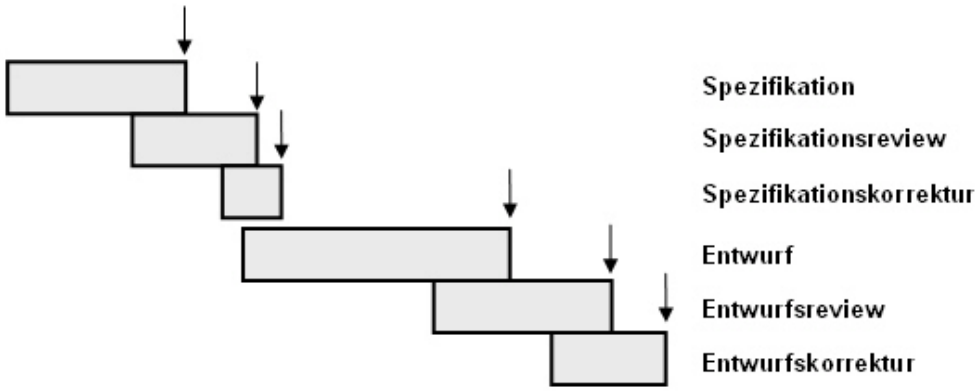

Fig. 4.3: Markierung eines Spielverlaufs

kierungsagent identifiziert werden müssen, sind die Ressourcesynchronisationspunkte. Ressourcesynchronisationspunkte unterscheiden sich nicht von Ressourcemarkierungspunkte. Um sie zu identifizieren, muss während des gesamten Spiels der Zustand, der zu markierende Ressource überwacht werden. Wird bei der Überwachung festgestellt, dass eine Ressourcenschranke (z.B. 50% des Budgets) überschritten wurde, dann wird dieser Spielzug markiert.

Neben den bereits, in Kapitel 4.1 auf Seite 26, erwähnten Projektphasen, werden auch deren Review- und Korrekturphasen markiert. Auch bei diesen markierten Subphasen zählen nur die Endmarkierungen als Synchronisationspunkte.

Weitere Synchronisationspunkte der Entwicklungsphasen sind die Tests, die nach der Kodierungsphase vom Spieler durchgeführt werden. Auch sie besitzen Korrekturphasen, die markiert werden.

Die Liste der Ressourcesynchronisationspunkte im QS Modell besteht zur Zeit aus folgenden Punkten

- 25,50,75,100% der Budgets wurden verbraucht.
- 25,50,75,100% des zeitlichen Budgets wurden verbraucht.

4.2.2 Schwierigkeiten bei der Entdeckung von Synchronisationspunkten

Die Schwierigkeiten bei der Entdeckung von Synchronisationspunkten bestand darin, den korrekten Spielzug zu markieren. Diese Problematik wird im Kapitel 5 auf Seite 31 genauer erklärt.

Kapitel 5

DER MARKIERUNGSAGENT

Dieses Kapitel beschäftigt sich mit der Arbeit des Markierungsagenten. Die Aufgaben und Eigenschaften des Markierungsagenten werden hier ebenso dargestellt, wie die Probleme, die bei der Markierung eines Spielzuges auftreten. Diese Probleme resultieren aus der Unterschiedlichkeit zwischen den verschiedenen Markierungspunkten, die in zwei Gruppen eingeteilt werden. Diese Gruppen bestehen aus den Phasenmilestones und den Ressourcenmilestones. Eine weitere Unterteilung gibt es bei den Phasenmilestones. Hier wird zwischen Punkten unterschieden, die Phasen einleiten und solchen die Phasen beenden. Auch bei dieser Unterteilung treten Probleme und Unterschiede auf, die hier aufgezeigt werden. Weiters wird die Frage geklärt, woher der Markierungsagent sämtliche Informationen bezieht, die er für das Markieren benötigt. Ebenso werden neue Datenstrukturen vorgestellt, die diesen Markierungsprozess erheblich erleichtern, und die technische Realisierung einer Markierung wird ausführlich erklärt.

5.1 Eigenschaften des Markierungsagenten

Der Markierungsprozess besitzt folgende Eigenschaften:

- Der Markierungsagent besitzt die Aufgabe relevante Punkte im Projektverlauf zu markieren. Zu diesen relevanten Punkten zählen Ressourcenmilestone (z.B.: 50 % des Budgets verbraucht) als auch Phasenmilestones (z.B.: Beginn der Spezifikation, Ende des Codereviews, etc.).

- Diese Markierungspunkte werden für die Vergleichkomponenten benötigt, um den Vergleich zwischen zwei oder mehreren Spielern fair zu gestalten.

- Der Markierungsagent lauscht auf Benutzerkommandos und auf Zustandsveränderungen. Benutzerkommandos markieren den Beginn einer Phase, Zustandsveränderungen markieren das Ende einer Phase.

Die Tatsache, dass Benutzerkommandos Phasen einleiten ist im „QS“ Modell so implementiert, in anderen Modellen muss dies nicht zutreffen. Wenn dies nicht der Fall ist, muss der Markierungsagent dahingehend erweitert werden.

5.2 Aufgaben des Markierungsagenten

Dieses Kapitel befasst sich mit den Aufgaben, die der Markierungsagent besitzt.
Zu diesen Aufgaben zählen:

- Auslesen der zu markierenden Milestones aus der Datenbank.
- Markieren von Anfangs- bzw. Endpunkten von Entwicklungsphasen (bzw. Subphasen). Jede Entwicklungsphase enthält Subphasen. Die Subphasen der Spezifikation sind die Spezifikationsreviewphase und die Spezifikationskorrekturphase.
- Nach dem Überschreiten gewisser Schranken sollen Spielzüge markiert werden (Resourcemilestones). Zu diesen Schranken zählen z.B.: 50% des Budgets verbraucht, 25% der Projektdauer überschritten, usw.
- Spielbeginn und Spielende müssen ebenfalls markiert werden.

5.3 Markierungsproblematik

Bei der Markierung von Phasen (Entwickungsphasen / Subphasen) treten Probleme auf. Diese Probleme resultieren aus der Verschiedenheit zwischen dem Phasenbeginn und dem Phasenende. „Axel specify“ ist eine Benutzereingabe die eine Entwicklungsphasen (Spezifikationsphase) einleitet, d.h. der Benutzer bestimmt beim „QS“ Modell, wann Phasen beginnen. Der Benutzer kann jedoch nicht durch das Absetzten eines Befehls ein Phasenende herbeiführen. Dieses Beispiel soll die Unterscheidung zwischen dem Phasenbeginn und dem Phasenende veranschaulichen.

5.3.1 Phasenbeginnmarkierungen

Sie werden durch einen erfolgreich abgesetzten Befehl angestoßen. Dieser Befehl wird vom Benutzer eingegeben und er bewirkt, dass die Phase beginnt.

Beispiele:
hire Axel; Axel specify = erfolgreich
hire Axel; Bernd specify = nicht erfolgreich

Die erste Befehlsfolge ist erfolgreich, da der Entwickler Axel zum Zeitpunkt der Befehlseingabe angestellt war. Bei der zweiten Befehlsfolge ist der Entwickler Bernd noch nicht eingestellt, weshalb er auch nicht mit der Spezifikation beginnen kann. Die Kontrolle, ob ein Befehl erfolgreich war oder nicht obliegt dem Sesamkern. Er kontrolliert alle Vorbedingungen, die für das Durchführen des Befehls („Bernd specify“) notwendig sind. Eine Vorbedingung für den Befehl „Bernd specify“ ist, dass Bernd bereits eingestellt wurde, was jedoch im Beispiel nicht der Fall ist.
Eine weitere Eigenschaft von Phasenanfangsmarkierungen ist jene, dass einmal markierte Phasenstartpunkte nicht wieder rückgängig gemacht werden können. Der Start der Spezifikationsphase kann während des gesamten Spieles nur einmal vorkommen. Für den Spieler besteht keine Möglichkeit den Phasenbeginn zu verändern.

5.3.2 Phasenendmarkierungen

Phasenendmarkierungen werden nicht durch abgesetzte Befehle eingeleitet, sondern durch den Systemzustand. Das System bestimmt, wann eine Phase endet. Diese Tatsache macht die Arbeit des Markierungsagenten bei den Phasenendmarkierungen viel schwieriger. Er muss ständig alle gerade aktiven Phasen (d.h. alle Phasen die bereits ein Phasenanfang, aber kein Phasenende besitzen) beobachten und prüfen, ob diese Phasen noch laufen. Wird nun erkannt, dass eine aktive Phase nicht mehr aktiv ist, bedeutet dies, dass diese Phase geendet hat. Daraus resultiert für den Markierungsagenten, dass der Vorgängerspielzug das eigentliche Phasenende darstellt, da das Sesamsystem dem Benutzer erst am nächsten Tag den Abschluss einer Phase mitteilt.

Die Abbildung 5.1 soll die Arbeit des Markierungsagenten beim Setzen von Phasenenden veranschaulichen. Die Punkte, die durch Linie verbunden sind, repräsentieren abgesetzte Befehle.

Der Spieler beginnt am 3.8.99 mit dem Spiel. Nach dem ersten „Proceed“ wird der Entwickler Axel eingestellt und darauf hin zum Spezifizieren eingeteilt. Anschließend wurde ein Proceed abgesetzt. Mit jedem „Proceed“ wird ein neuer Tag eingeleitet. Dieser Tag dauert dann bis zum nächsten „Proceed“. Zum Zeitpunkt des „Proceed“ Befehls wird erkannt, dass die Spezifikationsphase begonnen hat. Der genaue Ablauf der Markierung wird in Kapitel 5.7 auf Seite 48 noch genauer erklärt. Ab diesem Zeitpunkt muss der Markie-

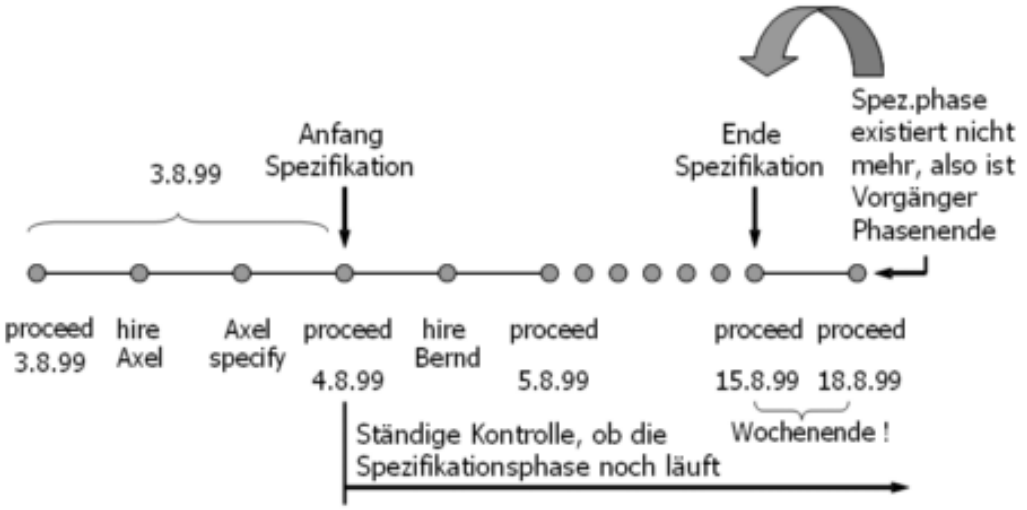

Fig. 5.1: Setzen des Endmilestones

rungsagent ständig lauschen, ob die Spezifikationsphase noch aktiv ist. Bei dem Proceed am 18.8.99 wird durch Kontrolle des Datenbankeintrages der Spezifikation festgestellt, dass die Spezifikationsphase geendet hat. Die genaue Funktionsweise wird im Kaptitel 5.4.1 auf Seite 36 genauer erklärt. Das bedeutet, dass der Vorgängerbefehl, in diesem Fall „Proceed“ am 15.8.99, das Phasenende markiert.

Nicht nur in der hier erklärten aufwändigeren Markierung unterscheiden sich Phasenanfangs- von Phasenendmarkierungen, sondern auch durch das andauernde Verändern des Phasenendes. Phasenanfangsmarkierungen bleiben während des gesamten Spiels bestehen, wohingegen sich Phasenendmarkierungen andauernd ändern. Diese Veränderungen sind erneut durchgeführte erfolgreiche Wiederholungen der Subphasen. Entwicklungsphasen können nicht mehr wiederholt werden, bzw. der erneute Versuch würde nicht erfolgreich sein. Siehe dazu auch das Kapitel der Markierungsevolutionsstufen auf Seite 35.

Diese Abbildung 5.2 zeigt eine vollständige Markierung der Spezifikationsreviewphase. Der obere graue Balken zeigt die Gesamtdauer dieser Phase, die weißen Balken repräsentieren die Tätigkeiten der Entwickler. In diesem Beispiel arbeiten an der Spezifikationsreviewphase Bernd und Christine. An der Anzahl der weißen Balken in horizontaler Richtung kann man erkennen, dass Bernd und Christine drei Mal versucht haben, die Spezifikation zu reviewen. Reviewversuch 1 und Reviewversuch 3 waren erfolgreich, Review-

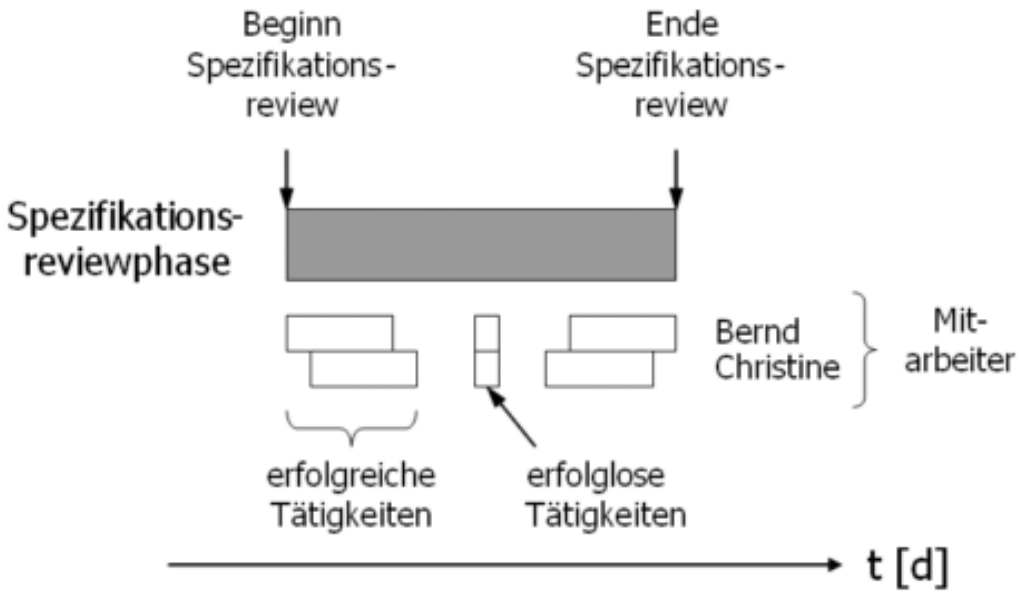

Fig. 5.2: Vollständige Markierung einer Subphase

versuch 2 scheiterte. Gründe für das Scheitern von Subphasen im folgenden Absatz erklärt.

Markierungsevolutionsstufen

Die Abbildung 5.3 zeigt die Situation nach dem ersten erfolgreichen Review der Spezifikation. Die Entwickler Bernd und Christine haben diesen Review durchgeführt. Erfolgreich ist ein Review dann, wenn er tatsächlich durchgeführt werden kann. Eine Phase / Subphase kann durchgeführt werden , wenn alle Mitarbeiter und alle Vordokumente verfügbar sind. Überprüft werden diese Vorbedingungen durch den SESAM Kern. Bei erfolglosen Phasen, fehlen diese soeben erwähnten Punkte.

Die Abbildung 5.4 zeigt einen erfolglosen Reviewversuch. Bei erfolglosen Versuchen verändern sich die Endmarkierungen von Phasen nicht, da es nie tatsächlich zu einer Tätigkeiten kam.

Nach dem erfolglosen Reviewversuch wurde in Abbildung 5.5 noch ein erfolgreicher Reviewversuch durchgeführt. Dieser erfolgreiche Reviewversuch hat zur Folge, dass die alte Endemarkierung ungülitig wird und eine neue Endspezifikationsreviewmarkierung gesetzt werden muss.

Nachdem dies der letzte Reviewversuch dieses Spiels war, ist diese Markierung für diese Subphase nun vollständig.

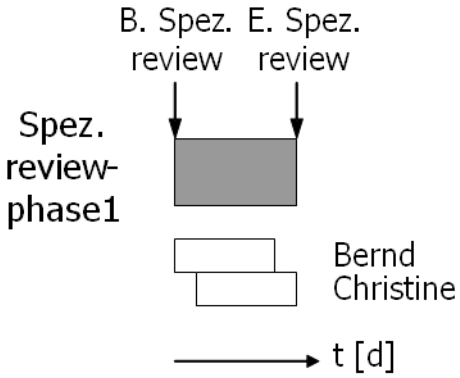

Fig. 5.3: Spezifikationsreviewphase 1

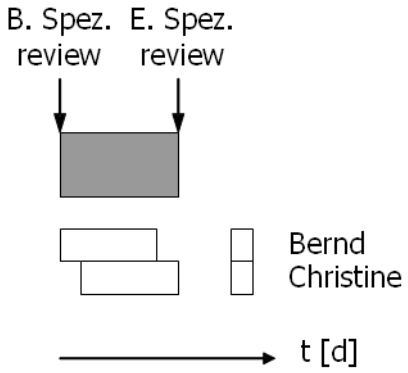

Fig. 5.4: Spezifikationsreviewphase 2

5.4 Technische Durchführung des Markierungsagenten

5.4.1 Einführung einer neuen Datenstruktur (MilestoneArchiv MSA)

Das Milestonearchiv ist ein Vector der Milestonearchivobjekte beinhaltet. Diese Milestonearchivobjekte bestehen aus Daten der Milestones, Spielinformationen und aus zwei Zustandsvariablen. Zu den Daten der Milestones zählen die Namen der Anfangs- bzw. Endmilestones und deren Beginn- und Enddaten. Die Spielinformationen bestehen aus dem aktuellen Pfad (wichtig für zukünftiges Rollback), der GameId (identifiziert das gerade aktuelle Spiel) und dem aktuellen Turn (identifiziert den gerade aktuellen Spielzug). Die Zustandsvariablen (change und written) sind boolesche Variablen, die den Zustand des gerade betrachteten Milestonearchivobjektes beschreiben.

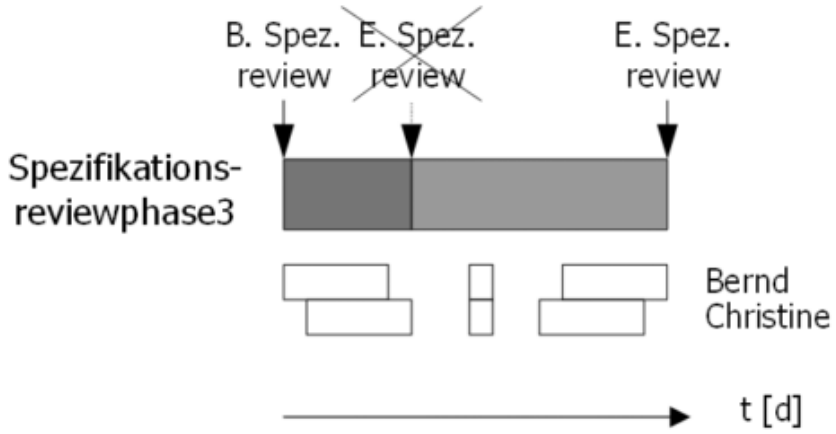

Fig. 5.5: Spezifikationsreviewphase 3

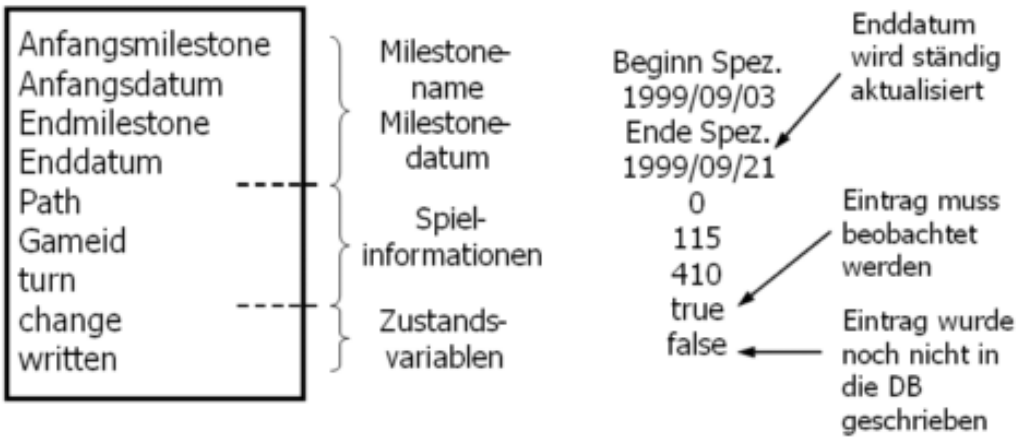

Fig. 5.6: Milestonearchivobjekt (MSAO)

Besitzt change den Wert true so bedeutet es, dass der MSA-Eintrag (gerade betrachtetes Milestonearchivobjekt) noch zu bearbeiten ist, weist es den Wert false auf, so kann dieser MSA-Eintrag bei der Bearbeitung außer Acht gelassen werden. Ist die Wertausprägung der zweiten Zustandsvariable (written) true so bedeutet dies darauf hin, dass der Endmilestoneeintrag bereits in die Datenbank geschrieben wurde. Bei einer Veränderung der Endmarkierung, muss der bereits geschriebene Datenbankeintrag wieder gelöscht werden. Hat hingegen written die Wertausprägung false so bedeutet dies, dass

der MSA-Eintrag noch nicht in die Datenbank geschrieben wurde.

In der Abbildung 5.6 wurde neben dem Milestonearchiv (Milestonearchivobjekt) auch ein konkretes Beispiel angeführt. Dieses Beispiel soll im folgenden Absatz die einzelnen Attribute des Milestoneagenten noch einmal anschaulich darstellen.

Zu Spielbeginn ist der Vector der das Milestonearchiv beinhaltet noch leer. Beginnt eine Phase, wie im folgenden Beispiel die Spezifikationsphase am 3.9.1999 so wird ab diesem Zeitpunkt dieser MilestoneArchiveintrag erstellt. Das Milestonearchiv (erster Eintrag im Milestonearchiv) sieht danach wie folgt aus:

Milestonearchiveintrag vom 3.9.1999:
Anfangsmilestone: Beginn Spezifikation
Anfangsdatum: 1999/09/03
Endmilestone: Ende Spezifikation
Enddatum: 1999/09/03
Path: 0
GameId: 115
Turn: 401
Change: true
Written: false

Die Namen des Anfangs- und Endmilestones werden aus der Datenbank ausgelesen. Wie man bei diesem Beispiel „Milestonearchiveintrag vom 3.9.1999“ im Kapitel 5.4.1 auf Seite 38 sieht, wird zu Beginn auch immer das Enddatum der Spezifikationsphase eingetragen. Nach dem Erstellen des Eintrags werden Anfangsmilestone, Anfangsdatum, Endmilestone und die GameId nicht mehr verändert. Der Patheintrag ändert sich nur bei einem „Rollback“ (Zurücksetzen des Spieles, auf einen vorher markierten Spielstand). Der Turneintrag verändert sich bei jedem Spielzug und wird bei einem aktivem Eintrag (change = true) bei jedem Proceed verändert. Die beiden Zustandsvariablen (change und written) werden ebenfalls in ihren Initialzustand gebracht. Der Zustand der beiden Zustandvariablen im Beispiel „Milestonearchiveintrag vom 3.9.1999“ bedeutet, dass dieser MSA-Eintrag noch bearbeitet wird und dass der Spezifikationsendmilestone noch nicht in die Datenbank geschrieben wurde. Der Spezifikationsbeginnmilestone wird aber bereits jetzt in die Datenbank eingetragen, weil er sich nicht mehr verändern kann. An den folgenden Tagen wird nun dieser MSA-Eintrag ausgelesen und mit dem Ergebnis der Verarbeitung des Sesamkerns, dass sich bereits in der Daten-

bank befindet, verglichen. Die Überführung der verarbeiteten Daten des Sesamkerns in die Datenbank finden bei jedem „Proceed“ statt. Aus diesem Vergleich geht hervor, ob diese Phase noch aktiv ist, oder bereits geendet hat. Wenn die Phase noch aktiv ist, wird das Enddatum und der „turn“ Eintrag auf den aktuellen Stand gebracht.

Milestonearchiveintrag vom 22.9.1999:
Anfangsmilestone: Beginn Spezifikation
Anfangsdatum: 1999/09/03
Endmilestone: Ende Spezifikation
Enddatum: 1999/09/22
Path: 0
GameId: 115
Turn: 410
Change: true
Written: false

Die Phase ist auch am 22.9.1999 noch aktiv. Aus dieser Erkenntnis könnte dann dieses Beispiel „Milestonearchiveintrag vom 22.9.1999“ auf Seite 39 vom 22.9.1999 resultieren. Die Spezifikationsphase endet am 22.9.1999. Dieses Enden der Phase wird dem Spieler und damit auch dem Markierungsagenten aber erst am 23.9.1999 mitgeteilt. Da jedoch bei jedem Proceed (also auch jenes Proceed vom 22.9.1999) das Datum des Endmilestones und der aktuelle Turn gesetzt werden, ist es nun ein Leichtes, diesen Milestonearchiveintrag in die Datenbank zu überführen. Alle Informationen, die für die korrekte Überführung in die Datenbank benötigen werden, befinden sich bereits im Milestonearchiv. Trotzdem wird dieser MSA-Eintrag nach dem Überführen in die Datenbank noch einmal verändert, um die Zustandsvariablen (change und written) auf ihren korrekten Zustand zu bringen, um die Konsistenz dieses Archivs mit der Datenbank aufrecht zu erhalten.

Milestonearchiveintrag vom 23.9.1999:
Anfangsmilestone: Beginn Spezifikation
Anfangsdatum: 1999/09/03
Endmilestone: Ende Spezifikation
Enddatum: 1999/09/22
Path: 0
GameId: 115

Turn: 410
Change: false
Written: true

Dieses Beispiel „Milestonearchiveintrag vom 23.9.1999“ auf Seite 39 zeigt den konsistenten Milestonearchiveintrag vom 23.9.1999. Die Daten des Beginn- und Endzeitpunktes der Phase stimmen mit dem tatsächlichen Ende der Phase überein. Wie diese Daten in die Datenbank eingetragen werden, beschreibt das Kapitel 5.5.

Die beiden Zustandsvariablen (change und written) sagen in ihrer derzeitigen Konstellation aus, dass dieser MSA-Eintrag nicht mehr bearbeitet werden muss (change = false) und dass das Ende dieser Phase bereits in die Datenbank überführt wurde.

5.4.2 Neustart / Rekonstruktion des Milestonearchives

Da sich das Milestonearchiv am Client des Spielers befindet und nicht am Server beheimatet ist, wird in Fall eines Neustarts oder eines Absturzes eine Rekonstruktion des Milestonearchives notwendig. Diese Rekonstruktion ist von Nöten, um :

1. das Milestonearchiv mit den Datenbankeinträgen konsistent zu halten und
2. um keine Phasenenden bei der Markierung zu vergessen.

Man kann in einem solchen Fall zwei Arten von Rekonstruktionen unterscheiden:

1. Rekonstruktion direkt nach einem Proceed, noch bevor ein anderer Befehl abgesetzt wurde und
2. Rekonstruktion nach beliebigen Befehlen.

Diese Abbildung 5.7 zeigt einen Absturz oder ein Beenden direkt nach einem Proceed. Es wurde kein anderer Befehl nach dem letzten Proceed abgesetzt. Nach dem „Proceed“ am 4.8.99 wird der Milestonearchiveintrag der Spezifikation angelegt. Zusätzlich wird der Beginnmilestone der Spezifikation in die Datenbank eingetragen. Wenn nun der Spieler den Client beendet, oder das System abstürzt, dann geht das Milestonearchiv verloren und mit

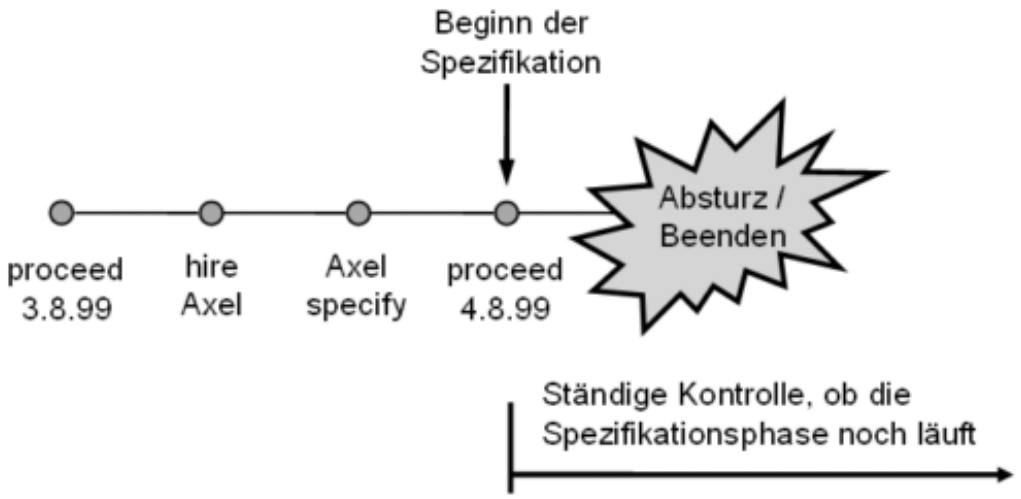

Fig. 5.7: Rekonstruktion direkt nach einem Proceed (Fall 1)

ihm auch der soeben erstellte Milestonearchiveintrag der Spezifikationsphase. Würde jetzt der Milestoneagent ohne Rekonstruktion seines Milestonearchivs die Arbeit fortsetzten, so würde das Ende dieser Spezifikationsphase nicht gesetzt werden, da der Milestoneagent nichts von der Existenz des Beginnmilestones dieser Phase wüsste und deshalb auch auf kein Ende warten würde.

Die Rekonstruktion dieses Milestonearchivs ist in diesem Fall einfach. Der Markierungsagent muss die Daten des Spieles in der Datenbank nach gesetzten Milestones durchsuchen. Werden Anfangs- und Endmilestones der Phasen in der Datenbank gefunden, so können deren Daten einfach in der Milestonearchiv übernommen werden. Die Zustandsvariablen sind in diesem Fall auf change = false und written = true zu setzen.

Werden Anfangsmilestones und keine analogen Endmilestones der jeweiligen Phasen gefunden, so wird das Anfangsdatum in den Milestoneagent übernommen und die Daten des Endemilestones werden auf den aktuellen Spielzug gesetzt. Zu diesen Daten die gesetzt werden müssen zählen der „turn“ Eintrag, der auf den aktuellen Spielstand gesetzt werden muss und das Datum des Endmilestones, welches ebenfalls auf das aktuelle Spieldatum gesetzt werden muss. Die Zustandsvariablen sind in diesem Fall auf change = true und written = false zu setzt. Nun kann der Markierungsagent seine Arbeit wieder aufnehmen.

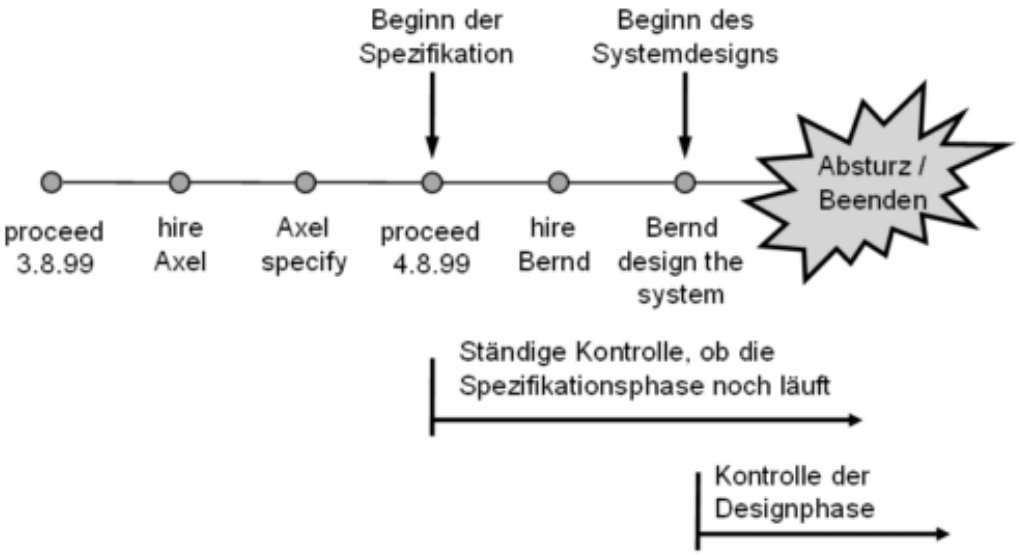

Fig. 5.8: Rekonstruktion nach einem normalen Befehl (Fall 2)

In dieser Abbildung 5.8 werden nach dem Proceed noch andere Befehle abgesetzt. Da nur bei einem Proceed die Übertragung der Daten an die Datenbank erfolgt, stehen wir hier vor einem Problem. Die Befehle „hire Bernd“ und „Bernd design the system“ führen zu einer neuen Phase, die aber noch nicht in der Datenbank eingetragen wurde. Dieses Fehlen der Daten in der Datenbank führt dazu, dass bei einer Rekonstruktion nur anhand der Daten der Datenbank diese Informationen verloren gehen. Aus diesem Grund ist es wichtig vor der Rekonstruktion des Milestoneagenten jene Befehle durch die Verarbeitung des Markierungsagenten am Client kontrollieren zu lassen, die sich zwischen dem letzten Proceed und der Absturzstelle befinden. In dem Beispiel aus Abbildung 5.8 wären dies „hire Bernd“ und „Bernd design the system“. Der Befehl „Bernd design the system“ würde als Milestonekandidat erkannt werden und müsste bei der Rekonstruktion mitberücksichtigt werden, indem er beim nächsten Proceed an die Datenbank übertragen wird.

5.5 Bereitstellen der Milestoneinformationen und Abspeichern der aufgetretenen Milestones

Um den LVLeiter die Möglichkeit zu geben in den Markierungsprozess einzugreifen, lässt sich der Markierungsagent über die angegebenen Milestones in der Datenbank steuern. Die Abbildung 5.9 zeigt die für den Markierungs-

prozess erforderlichen Teile der Datenbank.

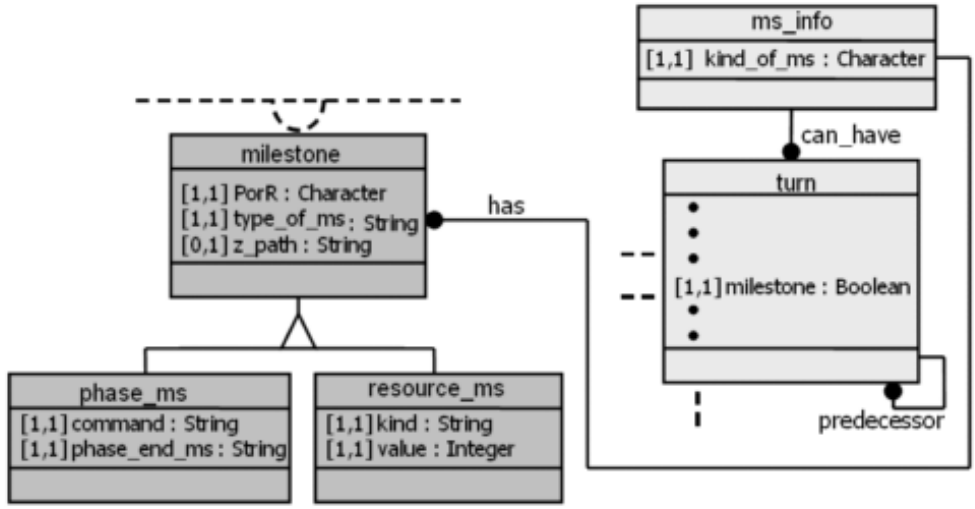

Fig. 5.9: Relevante Teile der Datenbank des Markierungsagenten

Die unterschiedlichen Färbungen dieser Tabellen beruhen darauf, dass sie für unterschiedliche Phasen der Markierung zuständig sind. Die dunkler gefärbten Tabellen beinhalten die Informationen der Milestones, die helleren Tabellen befassen sich mit der Markierung der Spielzüge und der Zuordnung der gefundenen Milestones zu den Spielzügen.

Tabelle Milestone, phase_ms, resource_ms

Bei der Tabelle Milestone handelt es sich um eine disjunkte Generalisierung. Das bedeutet, dass Milestones entweder Phasenmilestones sind, oder es sich um Ressourcenmilestones handelt. Es ist aber nicht möglich, dass ein Milestone zugleich Phasen- und Ressourcenmilestone ist.

Beispiele für Phasenmilestones:

Milestoneeintrag, der den Spezifikationsbeginn beschreibt:

PorR: P
type_of_ms: Beginn Spezifikation
z_path: PROJEKTLOGBUCH∗Projectlog∗SPEZIFIKATION_BEGINN
command: specify
phase_end_ms: Ende Spezifikation

Das Attribut „PorR“ sagt aus, ob es sich um einen Phasen- oder Ressourcenmilestone handelt, wobei „P“ für Phase und „R“ für Ressource steht. Das Attribut „type_of_ms“ bezeichnet den Namen des Milestones. Dieser Eintrag kann später für die Beschriftung des Spielverlaufs herangezogen werden, da der Eintrag in Klartext erfolgt. Das nächste Attribut „z_path“ bezeichnet jenen Ort in der Datenbank, wo sich das Datum zu diesem Eintrag befindet. Um über mehrere Tabellen iterieren zu können, werden die Suchkriterien mit einem „∗“ getrennt. Das Attribut „command“ gibt jenen Teil des Befehls wider, der mittels Textvergleich, mit dem abgesetzten Befehl des Spielers verglichen wird. Wird eine Übereinstimmung gefunden und beginnt die Phasen tatsächlich, dann werden die Informationen in das Milestonearchiv übernommen. Bei der Eingabe des „command“Eintrags muss darauf geachtet werden, dass ein Befehl eindeutig ist. Es darf nicht vorkommen, dass ein „command“Eintrag Teil eines anderen „command“Eintrages ist, da es dann zu Verwechselungen kommen kann. Um die Wichtigkeit dieser Aussage noch einmal zu betonen, sollte folgendes Beispiel dienen:

Beispiel „Milestoneeinträge der Datenbank“

type_of_ms: Beginn Spezifikation
command: specification

type_of_ms: Beginn Spezifikationsreview
command: review the specification

Beim ersten Eintrag handelt es sich um Teil eines Beginn Spezifikationseintrages. Dieser Eintrag besitzt im Commandeintrag den Eintrag „specification“. Dieser Commandeintrag ist aber auch Teil des Commandeintrages des zweiten Beispiels. Benutzt nun der Spieler den Befehl „...review the specification“ um die Spezifikationsreviewphase einzuleiten, dann kann es vorkommen, dass dieser Aufruf vom Markierungsagenten als Einleitung des Spezifikationsphase ansieht. Diese Verwechslung führt dann dazu, dass die Spezifikationsreviewphase nicht markiert wird.

Um dieses Problem zu lösen wurde der erste Commandeintrag erweitert und sieht nun wie folgt aus:

Beispiel des erweiterten Milestoneeintrags

type_of_ms: Beginn Spezifikation
command: write the specification

Durch diese Veränderung im Commandeintrag ist eine Verwechslung der Phasen nun nicht mehr möglich.

Das letzte Attribut (phase_end_ms) zeigt den Namen des Endemilestones an. Ohne diesen Eintrag wäre es nicht möglich, den Endemilestone der Phase zu finden.

Phasenmilestone, der das Spezifikationsende beschreibt
PorR: P
type_of_ms: Ende Spezifikation
z_path: PROJEKTLOGBUCH∗Projectlog∗SPEZIFIKATION_ENDE
command:
phase_end_ms:

Dieses wäre der zugehörige Endemilestone der Spezifikationsphase. Endemilestoneeinträge unterscheiden sich von Beginnmilestoneeinträge durch das Fehlen der „Command“ und „Phase_end_ms“ Einträge.

In Sesam / Ameise gibt mehrere Möglichkeiten einen Befehl abzusetzen, weshalb es auch für jeden Befehl einen eigenen Milestoneeintrag gibt. Ein anderer Commandeintrag für die Spezifikationsphase wäre „write the specification“. Dieser andere „command“Eintrag wird in einem eigenen Milestoneeintrag gespeichert, der sich vom Eintrag „Milestoneeintrag, der den Spezifikationsbeginn beschreibt“ nur durch den „command“Eintrag unterscheidet. Alle anderen Einträge bleiben unverändert.

Für den Markierungsprozess im „QS“Modell ist es wichtig, dass alle Befehlsaufrufe von Phasen, die markiert werden sollen, in der Tabelle Milestone der Datenbank enthalten sind. Anderenfalls tritt der Fall ein, dass eine Phase beginnt, diese aber nicht markiert wird, da der Befehlsaufruf nicht als Phasenaufruf angesehen wird. In zukünftigen Modellen, bei denen der Beginn einer Phase nicht mehr befehlsgesteuert ist, kann der Milestone-Teil der Datenbank nicht mehr verwendet werden, da es keinen Befehl gibt, dessen Erscheinen auf eine neue Phase schließen lässt.

Eine Schwierigkeit bei der Markierung von Phasen sind Befehle, die nicht eindeutig auf eine spezielle Phase schließen lassen. Ein Beispiel für einen solchen Befehl wäre ein Reviewaufruf. Dieser Reviewaufruf kann durch folgenden Befehl angestoßen werden: „Ask Christine and Stefanie to review“.

Nach dem Absetzten dieses Befehls wird ein Callbackfenster angezeigt. Dieses Callbackfenster zeigt dem Spieler alle Möglichkeiten auf, die er wählen

kann, um dem System mitzuteilen, welche Art von Review er gerne durchführen möchte. Für diese Art der Befehle muss der Eintrag in der Milestonetabelle leicht geändert werden.

Phasenmilestone, der den Spezifikationsreviewbeginn identifiziert
PorR: P
type_of_ms: Beginn Spezifikationsreview
z_path: REVIEWLOGBUCH∗ Reviewlog∗ SREVIEW_BEGINN
command: to review∗ Specification
phase_end_ms: Ende Spezifikationsreview

Wie bereits erwähnt wurde, benötigen diese Art von Befehlen einen leicht veränderten Eintrag. Dieser leicht verändert Eintrag ist der Commandeintrag. Der Commandeintrag besteht aus zwei Teilen. Die beiden unterschiedlichen Teile werden durch ein Trennzeichen (∗) voneinander getrennt. Der erste Teil bestimmt jenen String, der mittels Textvergleich im abgesetzten Befehl vorkommen soll. Der zweite Teil beinhaltet zwar auch einen String, nur wird dieser String nicht mit dem Befehl des Spielers, sondern mit dem Feedback des Systems verglichen. Die Antwort des Systems auf diesen eingegebenen Befehl lautet:

Spielerbefehl: „ask Christine and Stefanie to review“
Antwort: „ask Christine and Stefanie to review the Specification“.

Im Spielerbefehl wird „to review“ gefunden und in der Antwort des Systems wird „Specification“ entdeckt.

Ressourcenmilestones

Die andere große Gruppe der Milestones sind die Ressourcemilestones.

Beispiel eines Ressourcenmilestones
PorR: P
type_of_ms: 5% Ressourcen
z_path: PROJEKTZUSTAND∗Projectstatus∗KOSTEN
kind: Budget
value: 5000

Die Ressourcemilestones unterscheiden sich von den Phasenmilestones durch die beiden Attribute „kind“ und „value“. „kind“ drückt die Art des Ressourcenmilestones aus, „value“ den Vergleichswert, mit dem die aktuellen Kosten verglichen werden sollen. Die Attribute „PorR“, „type_of_ms“ und „z_path“ werden gleich wie bei den Phasenmilestones behandelt.

5.6 Informationsfluss zwischen Datenbank und Milestonearchiv

Wie vorher bereits erwähnt wurde, werden Teile der Datenbank in das Milestonearchiv überführt. Die folgende Abbildung 5.10 soll über den Informationsaustausch zwischen der Datenbank und dem Milestonearchiv Aufschluss geben.

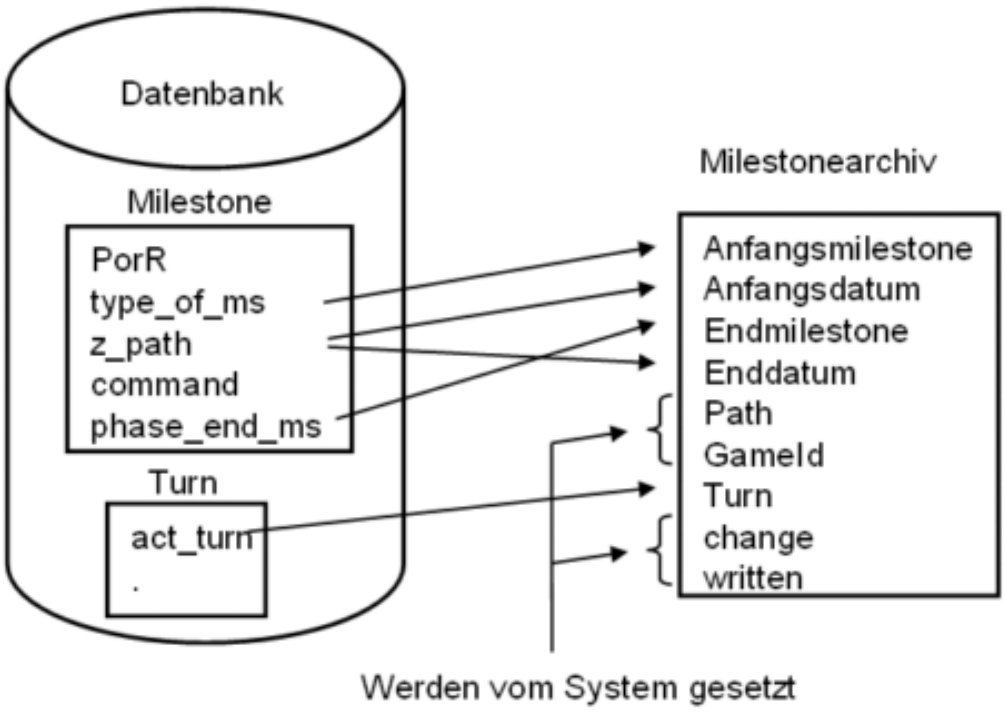

Fig. 5.10: Informationsfluss zwischen der Datenbank und dem Milestonearchiv

Wie aus der Abbildung 5.10 ersichtlich ist, stammen die Bezeichnungen des Anfangs- bzw. Endmilestons aus dem Attribut „type_of_ms“ bzw. „pha-

se_end_ms“ der Tabelle Milestone. Die Anfangs- bzw. Enddaten der Milestones werden nur indirekt aus dem Attribut „z_path“ ermittelt. Das Attribut „z_path“ gibt an, wo sich das gewünschte Datum in der Datenbank befindet. Der „Turneintrag“ des Milestonearchiv wird aus der Tabelle turn ausgelesen. Die beiden Zustandsvariablen change und written werden durch den Zustand des Milestonearchivs gesetzt. Die Einträge „path“ und „GameId“ werden durch das System gesetzt. „GameId“ repräsentiert das gerade aktive Spiel und der „Path“Eintrag ist für das Rollback eines Spiels verantwortlich.

5.7 Technische Realisierung des Markierungsprozesses

Im folgenden Abschnitt wird der Ablauf der Markierung im System geschildert. Unterstützend zur theoretischen Beschreibung des Markierungsprozesses, wird der Markierungsprozess auch bildlich dargestellt. Diese bildliche Darstellung beschreibt einen typischen Spielablauf.

Wenn das Spiel begonnen wird, muss sich der Spieler am Client anmelden. Nach der Anmeldung, wird der Spieler vor die Wahl gestellt, an welchem Spiel er teilnehmen möchte. Wurde auch dieser Schritt erfolgreich durchgeführt, wird dem Client eine Liste jener Milestone aus der Datenbank übermittelt und lokal gespeichert, die für dieses Spiel verwendet werden sollen. Diese Liste der Milestones beinhaltet alle Milestoneeinträge der Datenbank. Siehe dazu auch Abbildung 5.9 auf Seite 43. Diese Milestones werden in einem gleichnamigen Vector in der Klasse Markierung am Client gespeichert. Wird der Client beendet, so wird bei einem Neustart diese Liste wieder von der Datenbank ermittelt und wieder am Client abgelegt. Um die Initialisierung des Clients und des Simulators abzuschließen, ist bei einem erstmaligem Start des Systems ein „Proceed“ abzusetzen. Nach diesem „Proceed“ wird dem Spieler die Aufgabenstellung präsentiert und der Simulator ist für die Verarbeitung der Benutzerkommandos bereit. Bei einem erneuten Start des selben Spieles ist dieses „erste Proceed“ nicht erforderlich.

Nach der Initialisierung kann der Benutzer beginnen, Befehle abzusetzen. Diese Befehle werden an den Wrapper (Sesam Kern) gesandt. Im Sesam Kern werden die Befehle verarbeitet und dementsprechende Feedbacks werden erzeugt. Anschließend werden die Befehle und die Feedbacks an den Client und die Datenbank gesendet. Siehe dazu folgende Abbildung 5.11.

In dieser Abbildung 5.11 wurde am Client der Befehl „hire Axel“ abgesetzt. Dieser Befehl wird an den Wrapper übertragen. Im Wrapper wird dieser Befehl verarbeitet. Anschließend wird der Befehl und das Feedback an den Client und die Datenbank geschickt. Die Kommunikation zwischen dem Wrapper und der Datenbank wurde aus Gründen der Übersicht weggelassen.

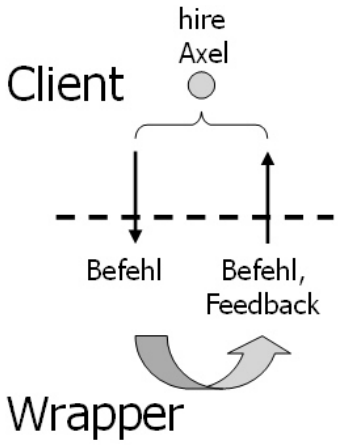

Fig. 5.11: Beschreibung des Markierungsprozesses 1

Wenn der Client Befehle und Feedbacks vom Wrapper erhält, beginnt ein Verteilungsmechanismus im Clients die erhaltenen Daten an die entsprechenden Instanzen des Clients zu überreichen. Wie dieser Verteilungsmechanismus funktioniert, lesen Sie bitte im Kapitel ... auf Seite ... nach.

Die Markierungsinstanz ist einer der ersten Instanzen, die sich mit den Daten des Wrappers beschäftigt. Sie wird durch das File Markierung.java realisiert. Dieses Objekt nimmt Befehle und Feedbacks entgegen. Dies geschieht in der Methode „processDistributedObject“. In dieser Methode wird nach Befehlen gesucht, die möglicherweise einen BeginnMilestone auslösen können. Die endgültige Kontrolle, ob dieser Befehl tatsächlich einen Milestone ausgelöst hat, findet am Wrapper statt. Diese Befehle die wahrscheinlich eine neue Phase- oder Subphase auslösen, werden dann in den Vector der MilestoneKandidaten aufgenommen.

Diese Abbildung 5.12 zeigt die Überprüfung dieses abgesetzten Befehls mit den Einträgen der Milestoneliste. Der Befehl „hire ...“ wird nicht in der Milestoneliste gefunden, weshalb dieser Befehl auch nicht in die Liste der Milestonekandidaten aufgenommen wird.

Ein weiterer wichtiger Punkt für die korrekte Zuteilung von Befehlen zu Phasen / Subphasen ist der, dass man zwischen Befehlen die Reviews aufrufen und „normalen“ Befehlen unterscheiden muss. Bei den Reviewaufrufen kann es vorkommen, dass man alleine auf Grund des abgesetzten Befehls, noch nicht auf eine Phase schließen kann. Deshalb muss bei dieser speziellen Art der Reviewaufrufen auch immer das Feedback betrachtet werden, um diesen Befehl eindeutig einer Phase zuordnen zu können. Diese Unterscheidung muss nicht nur bei der Bearbeitung sondern auch in der Datenbank

Fig. 5.12: Beschreibung des Markierungsprozesses 2

berücksichtigt werden, darum werden diese Art von Milestones in der Datenbank unterschiedlich gespeichert. Der „Commandeintrag“ dieser Art von Milestones beinhaltet deshalb neben dem Befehl auch den notwendigen Feedbackeintrag. Mit diesem Feedbackeintrag wird dann das tatsächliche Feedback verglichen. Siehe dazu auch Kapitel 5.5.

Befehle, die vom Wrapper als fehlerhaft gekennzeichnet wurden, oder Callbackbefehle die mit einem „Cancel“ bestätigt wurden, werden nicht in die Liste der MilestoneKandidaten aufgenommen.

Diese Milestonekandidatenliste wird mittels des Befehls addElement_to_Milestonecandidates in der LogicalUnit gespeichert.

Die nächste Abbildung 5.13 zeigt, die Situation nach dem Absetzten des nächsten Befehls. Bei dem Befehl handelt es sich um den Befehl „Axel specify“. Dieser Befehl wird erneut an den Client geschickt, dort bearbeitet und kommt zusammen mit dem Feedback wieder zurück zum Client.

Am Client angekommen wird dieser Befehl wieder mit den Einträgen in der Milestoneliste verglichen. Siehe Abbildung 5.14. Da der Befehl „... specify“ im „Command“ Eintrag eines Elements des Milestonearchivs ist, wird dieser neue Eintrag in die Liste der Milestonekandidaten übernommen. Dieser Eintrag aus der Milestoneliste, der den Datenbankeintrag widerspiegelt wird nun einfach in den Vector der Milestonekandidaten übernommen.

Das Objekt Markierung besitzt noch eine weitere Aufgabe: in der Methode „processDistributedObjekt“ wird auch das Clientblocking realisiert. Das Clientblocking nimmt dem Spieler (nach bestimmten Aktionen, wie die Übergabe des Systems an den Kunden) die Möglichkeit, weiterzuspielen. Dabei wird nach einem bestimmten Befehl oder eines bestimmten Feed-

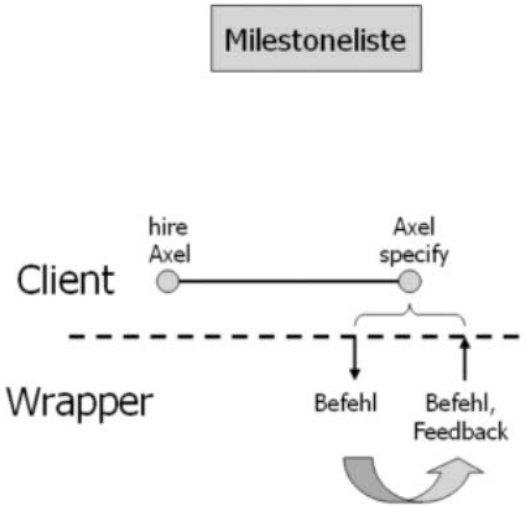

Fig. 5.13: Beschreibung des Markierungsprozesses 3

backs dem User die Möglichkeit entzogen, weitere Befehle abzusetzen. Diese auslösenden Befehle sind z.B. deliver system (dieser Befehl entspricht der Übergabe des Systems an den Kunden) oder stop project (das Projekt wird unabhängig vom aktuellen Zustand beendet). Es gibt auch Feedbacks, die vom System kommen und anzeigen, dass der Simulator nach dem Überschreiten gewisser Grenzen seine Tätigkeit einstellt. Ein Beispiel für eine solche Grenze wäre das Überschreiten der Projektdauer um das Doppelte. Danach wird dem Spieler ebenso die Möglichkeit entzogen weitere Befehle abzusetzen.

Nun da in der Milestonekandidatenliste wahrscheinliche Milestones vorhanden sind, müssen diese Milestones an den Wrapper übertragen werden. Dieses Übertragen findet bei jedem „Proceed“ statt. Beim „Proceed“ wird aber nicht nur die Milestonekandidatenliste, sondern auch das Milestonearchiv übertragen. Beide Vector werden aus der LogicalUnit ausgelesen. Natürlich kann es in der Anfangsphase oder bei einem Neustart vorkommen, dass das Milestonearchiv noch leer ist.

Die Abbildung 5.15 zeigt nun die Situation, die beim Absetzen eines „Proceed“ stattfindet. Beim Proceed werden neben dem Befehl „Proceed 1“ auch die beiden Vektoren (Milestonearchiv und Milestonekandidaten) vom Client zum Wrapper übertragen. Das Milestonearchiv, dass nun zum ersten Mal in diesem Beispiel aufscheint, ist zu diesem Zeitpunkt noch leer.

Nach dem Übertragen des Milestonearchivs und der Liste der Milestone-Kandidaten wird die Methode „update_Milestonearchiv_and_db“ aufgerufen.

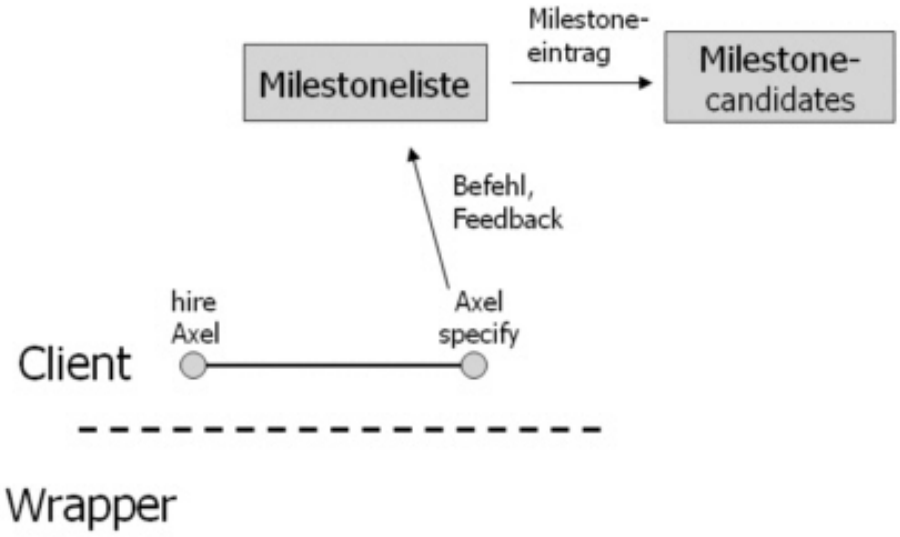

Fig. 5.14: Beschreibung des Markierungsprozesses 4

Diese Methode trägt die Hauptlast der Markierung. In ihr werden die beiden Vektoren durchlaufen und bearbeitet. Anhand dieser beiden Vektoren (Milestonearchiv, Milestonecandidates) können wird vier Fälle unterscheiden:

Fall 1: Milestonearchiv ist leer; Milestonekandidatenliste ist leer

Wenn dieser Fall eintritt, existieren noch keine Milestones und die gerade abgesetzten Befehle (alle Befehle zwischen diesem Befehl und den vorhergehenden proceed) deuten, nach dem Vergleich mit der gespeicherten Milestones, auf keine neuen Milestones hin. In diesem Fall ist für diese Methode nichts zu tun.

Fall 2: Milestonearchiv ist leer; Milestonekandidatenliste ist nicht leer

Da das Milestonearchiv leer ist, deutet diese Situation darauf hin, dass es noch keine Milestones gibt. Die Einträge in der Milestonekandidatenliste müssen überprüft werden, ob sie tatsächlich eine Phase / Subphase angestoßen haben. Wenn dies der Fall ist so wird das Milestonearchiv mit den Daten aus der Datenbank gefüllt. Siehe dazu auch folgende Abbildung 5.10. Zusätzlich zu dem Füllen des Milestonearchiv wird auch gleichzeitig die Datenbank verändert. Diese Veränderung betrifft den „turn“ Eintrag und eine neue Beziehung wird zwischen der Tabelle „ms_info“ und „turn“ realisiert.

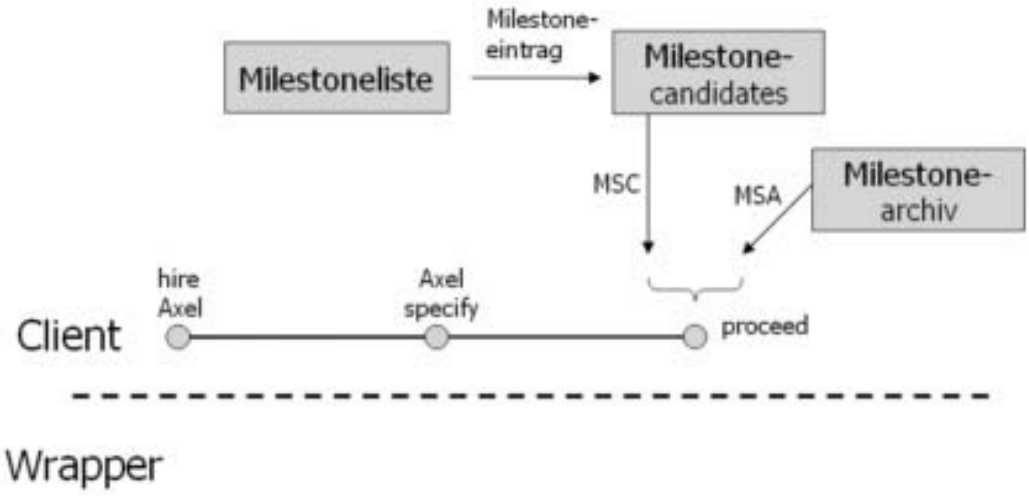

Fig. 5.15: Beschreibung des Markierungsprozesses 5

Dieser Datenbankeintrag kann gleich von statten gehen, da sich der Beginn einer Phasen / Subphasen nicht mehr verändern kann.

Fall 3: Milestonearchiv ist nicht leer; Milestonekandidatenliste ist leer

Es wurde kein Befehl abgesetzt, der auf einen Milestone schließen lässt. Da das Milestonearchiv gefüllt ist, muss dieser Vector durchlaufen werden. Zu jedem Eintrag im Milestonearchiv wird das Attribut change und written durchsucht. Je nachdem, welchen Wert das Attribut besitzt, muss die Verarbeitung unterschiedlich vonstatten gehen. Besitzt das Attribut change den Wert true so bedeutet es, dass dieser Eintrag bearbeitet werden muss, change = false deutet dann darauf hin, dass die Bearbeitung dieses Eintrages schon abgeschlossen ist. Besitzt das Attribut written den Wert true so bedeutet es, dass der Endmilestone bereits in die Datenbank überführt wurde, wird der Wert false gefunden, wurde noch nichts in die Datenbank übertragen.

Fall 4: Milestonearchiv ist nicht leer, Milestonekandidaten sind nicht leer

Dieser Fall bedeutet, dass sowohl Milestones bereits im Archiv sind , als auch Milestones in der Milestonekandidatenliste vorkommen. Er ist eine Kombination von Fall 2 und Fall 3. Es bedeutet, dass es bereits Milestones gibt und dass die letzten Befehle auch auf neue Befehle schließen lassen.

Nach der genauen Beschreibung der Tätigkeiten zeigt die Abbildung 5.16

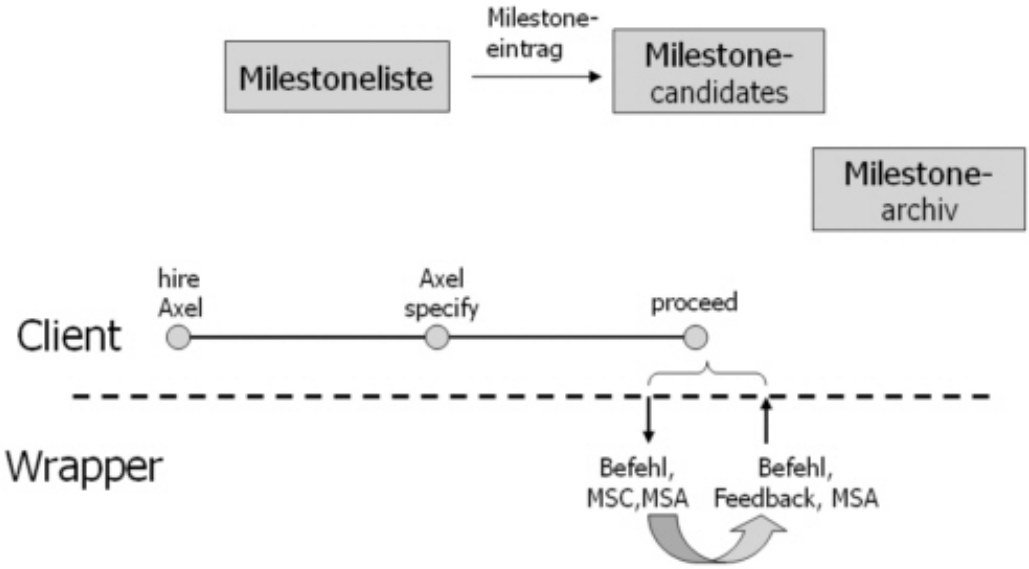

Fig. 5.16: Beschreibung des Markierungsprozesses 6

wie dies in dem Beispiel aussieht. Bei der Bearbeitung am Wrapper wird nun entdeckt, dass der abgesetzte Befehl „Axel specify“ tatsächlich eine neue Phase begonnen hat. Da es aber für die Markierung nicht relevant ist, welcher Befehl die neue Phase angestoßen hat, wird der Befehl („Axel specify“) nicht an den Wrapper übermittelt, sondern nur der Milestoneeintrag den dieser Befehl beim Vergleich am Client ausgelöst hat. Da die Spezifikationsphase begonnen hat, wird das Milestonearchiv nun mit den Informationen aus der Datenbank und aus dem System gefüllt. Welche Informationen an welche Stelle im Milestonearchiv eingetragen werden, wurde im Kapitel 5.6 auf Seite 47 bereits gezeigt. Neben dem Erstellen des Eintrages im Milestonearchiv wird auch in der Datenbank der Spielzug als Milestonespielzug gekennzeichnet.

Leider kann das Fehlen von Einträgen im Milestonearchiv auch darauf deuten, das ein Neustart erfolgt ist. Das Problem mit dem Neutstart wurde aus Seite 40 schon erwähnt. Weiters kann nicht davon ausgegangen werden, dass der Absturz direkt nach einem Proceed stattgefunden hat, was wiederum bedeutet, dass es vielleicht noch andere Befehle gibt, die noch nicht in den Milestonekandidaten aufscheinen. Die Rekonstruktion des Milestonearchiv hat immer vor den vier Fällen stattzufinden, um unnötige Arbeit zu vermeiden.

Nach der Bearbeitung des Milestonearchivs und der Milestonekandidaten

wird das veränderte oder unveränderte Milestonearchiv wieder an den Client übertragen und dort wieder gespeichert, um beim nächsten Proceed wieder an die Datenbank übertragen zu werden.

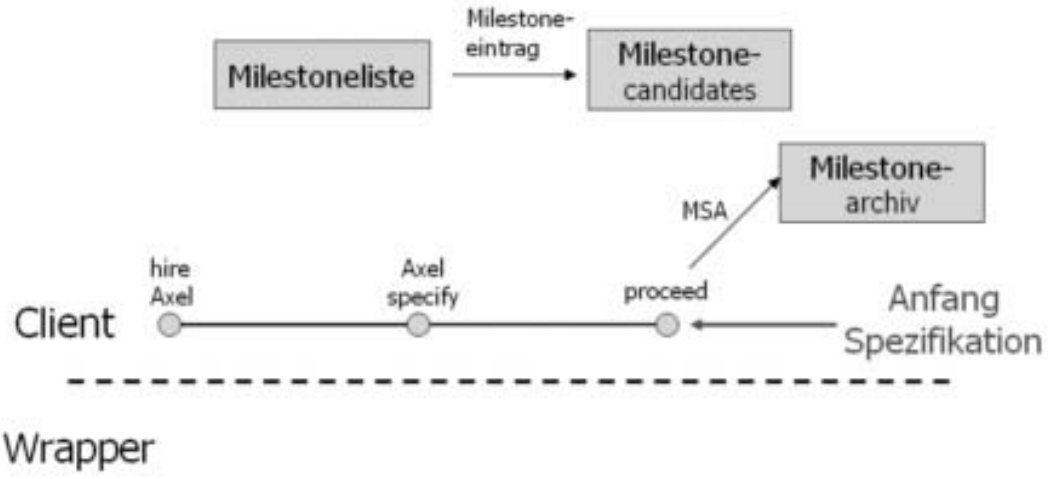

Fig. 5.17: Beschreibung des Markierungsprozesses 7

Diese Abbildung 5.17 zeigt die letzte Abbildung in diesem Beispiel. Nach der Bearbeitung wird der Befehl, das Feedback und das Milestonearchiv wieder an den Client übermittelt. Am Client wird dieses Milestonearchiv gespeichert, um beim nächsten „Proceed“ wieder an die Datenbank übertragen werden zu können.

5.7.1 Zusammenfassung

Dieses Kapitel zeigte die Tätigkeiten des Markierungsagenten, die für das gesamte Ameisespiel von Bedeutung sind. Insbesondere die Vergleichkompnente und der Treeview ist auf die Markierungen des Markierungsagenten angewiesen, aber auch das Ameisespiel (Clientblocking) ist auf die Arbeit des Markierungsagenten angewiesen. Weiters wurde in diesem Kapitel auf die Probleme der Markierung eingegangen und es wurde die aktuelle Implementierung vorgestellt.

Kapitel 6

RELEVANTE VERGLEICHSDATEN UND REGELN

Zu den relevanten Vergleichsdaten zählen Eckdaten und Projektregeln. Die Eckdaten sind für den Vergleich relevant, die Projektregeln bilden den Grundstock für den väterlichen Freund und den Ratgeber. Die Projektregeln bestimmen, auf welche Daten die Unterstützungswerkzeuge zugreifen und wie die Verarbeitung dieser Daten vorgenommen werden muss.

6.1 Eckdaten in Sesam / Ameise

In Sesam / Ameise existiert eine große Menge an Eckdaten, die während des Spiels bearbeitet und gefüllt werden, jedoch werden nicht alle Eckdaten für den Einzel-/ Gruppenvergleich benötigt.

Die Vergleichskomponenten (Einzel- / Gruppenvergleich) greifen über die Datenbank auf diese Eckdaten zu. Um zukünfigen Erweiterungen, oder gar neuen Komponenten nicht die Basis zu entziehen, werden alle Eckdaten des Sesam / Ameisespiels in der Datenbank gespeichert. Die Auswahl zwischen relevanten und nicht relevanten Eckdaten obliegt der Vergleichskomponente, die den Vergleich durchführt.

6.1.1 Eckdaten im Einzel- bzw. Gruppenvergleich

Um aus der riesigen Menge der verfügbaren Eckdaten des Sesam- / Ameisespiels jene auszuwählen, die für den Vergleich benötigt werden, wurden Auswertungen des Sesam- / Ameisespielen durchgeführt. Anhand der Ergebnisse der Auswertungen wurden die aussagekräftigsten Eckdaten der Auswertung ermittelt.

Für den Einzel- / Gruppenvergleich werden folgende Eckdaten verwendet:

1. Anzahl der erreichten AFPs. Mit Hilfe der AFPs ist es möglich festzustellen, wie umfangreich eine Phase des Projektes ist. Dieser Punkt ist auf alle Entwicklungsphasen, die während eines Spieles durchlaufen werden, anwendbar.

2. Aufwand. Mit wie viel Aufwand, angegeben in Stunden, wurde diese Phase erstellt? Dieser Punkte ist wieder auf alle Entwicklungsphasen des Projekts anwendbar.

3. Mitarbeitereinsatz. Wie viele Mitarbeiter haben bei der Erstellung dieser Entwicklungsphase mitgewirkt? Alle Phasen des Projekts werden betrachtet.

4. Kosten. Wie teuer war das gesamte Projekt oder die einzelne Entwicklungsphase? Neben den Entwicklungsphasen kann dieser Eckpunkt auch Aussagen über das gesamte Projekt treffen.

5. Kosten pro Mannmonat. Wie hoch waren die Kosten pro Mannmonat für das gesamte Projekt oder die einzelnen Entwicklungsphasen? Es kann wieder das gesamte Projekt, oder die einzelnen Entwicklungsphasen betrachtet werden.

6. Anzahl der gescheiterten Reviews. Wie viele Reviews wurden erfolglos durchgeführt? Dieser Eckpunkt ist für das gesamte Projekt und einzelne Entwicklungsphasen relevant.

7. Anzahl der erfolgreichen Reviews. Wie viele Reviews wurden erfolgreich durchgeführt? Ebenso wie der vorherige Eckpunkt ist auch dieser für das gesamte Projekt und die einzelnen Entwicklungsphasen anwendbar.

8. Revieweffektivität. Wie effektiv waren die Reviews? Diese Revieweffektivität gibt das Verhältnis zwischen den Fehler die im Dokument vorhanden sind und den gefundenen Fehlern an. Dieser Eckpunkt ist für alle Entwicklungsphasen relevant.

9. Korrektureffektivität. Wie effektiv war die Korrektur? Diese Korrektureffektivität gibt das Verhältnis zwischen den gefundenen Fehlern und den korrigierten Fehlern an. Anwendbar ist dieser Eckpunkt wieder für alle Entwicklungsphasen.

10. Restfehler. Wie viele Restfehler gibt es in den einzelnen Phasen? Dieses Eckdatum kann über alle Entwicklungsphasen Aussagen treffen.

Bei der Vergleichskomponente unterscheidet man zwei Fälle: Zwischen- und Endvergleiche. Die Unterscheidung zwischen dem Zwischen- und dem Endvergleich wird in Kapitel 3.2.1 ab Seite 14 erklärt. Beim Zwischenvergleich besitzt der Spieler die Möglichkeit auszuwählen, anhand welcher Daten sein Spiel, mit dem Spiel eines anderen Spielers verglichen werden soll. Diese möglichen Auswahlpunkte werden aus der Liste der Eckdaten generiert.

Vergleichsdaten			
Vgl.ID	Vergleichsart	interessiert an	relevant für
1	Zwischen-/Endvgl.	Spieler/Tutor	jede Phase
2	Zwischen-/Endvgl.	Spieler/Tutor	jede Phase
3	Zwischenvergleich	Spieler/Tutor	jede Phase
4	Zwischen-/Endvgl.	Spieler/Tutor	jede Phase/gesamtes Proj.
5	Zwischen-/Endvgl.	Tutor	jede Phase/gesamtes Proj.
6	Zwischen-/Endvgl.	Tutor	jede Phase/gesamtes Proj.
7	Zwischen-/Endvgl.	Tutor	jede Phase/gesamtes Proj.
8	Zwischen-/Endvgl.	Tutor	jede Phase
9	Zwischen-/Endvgl.	Tutor	jede Phase
10	Zwischenvergleich	Spieler/Tutor	jede Phase
11	Zwischen-/Endvgl.	Spieler/Tutor	jede Phase

In dieser Tabelle werden die Eckdaten erneut angeführt. Dabei wird folgendes unterschieden:

- Vgl.ID.
  Diese Abkürzung steht für Vergleichsidentifikator.

- Vergleichsart.
  Die Vergleichsart gibt an für welchen Vergleich dieses Eckdatum herangezogen werden kann. Dabei wird zwischen dem Zwischen- und dem Endvergleich unterschieden. Bei beiden Vergleichsarten muss der Spieler auswählen, welche Eckdaten er für den Vergleich heranzieht.

- interessiert an.
  Dieser Spaltenname gibt an, wer an dem Ergebnis des Vergleichs interessiert ist.

- relevant für.
  Einträge in dieser Spalte geben an, ob dieser Vergleich für Ergebnisse relevant ist, die das gesamte Projekt betreffen (wie z.B. die Kosten) oder für Phasenergebnisse.

6.2 Relevante Projektregeln in Sesam / Ameise

Alle Daten die während des Ameisespiels entstehen, werden in der Datenbank gesammelt. Diese Daten sind der Ausgangspunkt für die Unterstützungswerkzeuge (väterlicher Freund, Ratgeber). Die Daten aus der Datenbank werden für die

Berechnungen und Auswertungen herangezogen. Die Projektregeln der Unterstützungswerk-zeuge geben an, was mit diesen Daten zu geschehen hat.

spez.HMK-ID:3 Beschreibung: „Erneute Überarbeitung der Spezifikation notwendig?“ Erklärungstext: „Ihre Spezifikation enthält noch zu wenige AFPs“		
Instanz	Abfrage	Regel
Instanz-ID:2000 spez.HMK-ID:3 Vorgänger-ID: null Abfrage-ID: 5 Regel-ID: 4	Abfrage-ID:5 Attribut:ANZ_AFPs Spez Statement:“Select ...“	Regel-ID:4 Operator:“¡“ Richtwert: 195

Diese Beispiel stellt eine Projektregel des Ratgebers dar. Wählt der Spieler aus der Liste der Möglichen Ratschläge die „Erneute Überarbeitung der Spezifikations notwendig?“ aus, so geschieht folgendes:

- Die Instanz der speziellen Hilfsmittelkomponente wird ausgewählt. Die Auswahl erfolgt über die gleiche spez.HMK-ID.
- Über die Instanz gelangt man, wieder mittels ID zur Abfrage. In dieser Abfrage befindet sich das SQL-Statement, das abgesetzt wird.
- Die Instanz zeigt weiters auch auf die Regel dieser speziellen Hilfsmittelkomponente. Mit dem Richtwerteintrag der Regel wird das Ergebnis des SQL-Statements mittels des Operators verglichen.
- Ist dieser Vergleich positiv, so kann der Erklärungstext ausgegeben werden. Bei einem negativen Vergleichsergebnis, muss nach einer anderen speziellen Hilfsmittelkomponente gesucht werden, die die gleiche Beschreibung besitzt.

Die Entwicklung dieses Verfahrens wird in der Diplomarbeit von Fr. Susanne Jäger [Jaeger 2003] genauer erklärt.

Mit Hilfe dieses Verfahrens arbeitet sowohl der väterliche Freund, als auch der Ratgeber. Die Datenbankeinträge für den väterlichen Freund und den Ratgeber müssen vor dem Spiel vom Modellbauer eingetragen werden. Über diese Einträge ist es auch möglich den Umfang der Unterstützung des Spielers einzustellen.

6.3 Relevante Literatur

[Jaeger 2003]

Kapitel 7

AMEISE PROJEKTBEGLEITER

Die Analyse der Sesam- /Ameisespiel hat gezeigt, dass die Spieler immer die gleichen Fehler begehen. Um den Spieler zu unterstützen, werden neue Komponenten eingefügt, damit diese Fehler nicht mehr auftreten. Diese neuen Komponenten konnten aber nicht ohne Eingriffe in den Client von statten gehen. In diesem Kapitel werden die notwendig gewordenen Eingriffe in das System vorgestellt. Sie umfassen das Erweitern der Architektur und das Erstellen einer GUI, welche die Erweiterung aufnehmen kann. Zu den Erweiterungen des Ameisesystems zählen der Ratgeber, das Auswertungstool, der väterliche Freund, der Markierungsagent, die Vergleichskomponente und die Treeviewkomponente.

7.1 Eingriffe in das System

In diesem Kapitel werden neben der Architektur des „Urclients“ auch jene Erweiterungen vorgestellt, die für das Integrieren der neu entstandenen Komponenten notwendig waren.

7.1.1 Der „Urclient“

Der „Urclient“ wurde von einer Praktikumsgruppe im Wintersemester 2001/2002 an der Universität Klagenfurt entwickelt und erstellt. Die Entwicklung des Clients war nur ein Teil der Aufgabe, die die Praktikumsgruppe zu erledigen hatte. Der andere Teil der Aufgabe war das Entwickeln eines Client / Server Systems [1].

Der Client sollte folgende Aufgaben erledigen können:

- Eine stabile und zustandlose Kommunikation zwischen dem Client und dem Server soll erstellt werden.

[1] http:\\ameise.uni-klu.ac.at

- Um mit dem Client spielen zu können, muss sich der Benutzer am Client anmelden.
- Ein Fenster soll das Spielen am Client ermöglichen (Simulationsfenster).
- Spielständen sollen gespeichert und geladen werden können.
- Ein Spielbaum soll erzeugt werden, der den Spielverlauf des Spielers ausdrückt.

Neben den hier angeführten Punkten, die den Client betreffen, war es auch notwendig den Kern der Simulationsmaschine (Sesam) in einem Wrapper einzupacken, um damit das Gesamtsystem in ein Client- / Serversystem zu verwandeln. Weiters musste eine geeignete Kommunikation zwischen dem Client und dem Wrapper erstellt werden.

All diese Punkte wurden von der Praktikumsgruppe realisiert. Durch die Erstellung der zusätzlichen Komponenten (väterlicher Freund, Erklärungstool, Ratgeber, Markierungsagent) haben sich die Anforderungen geändert, wodurch der Client, Wrapper und die Kommunikation verändert werden mussten. Die Veränderungen an der Kommunikation zwischen den Komponenten (Client, Wrapper) sind sehr gering und werden in diesem Kapitel näher beschrieben. Es wurden zusätzliche Transportobjekte erzeugt, die in den bereits vorhandenen Kommunikationsobjekten verpackt wurden. Weitere Veränderungen am Gesamtsystem sind nicht Teil dieser Diplomarbeit und können auf der Ameisehomepage [2] nachgelesen werden.

7.1.2 Der „neue“ Client

Um die notwendig gewordenen Erweiterungen in die Tat umzusetzen, wurde der „Urclient“ herangezogen. Diese Erweiterungen bezogen sich hauptsächlich auf das graphische Benutzerinterface (GUI), das höheren Anforderungen genügen musste.

Die Aufgabe der Weiterentwicklung bestand darin, die Architektur des „Urclients“ zu übernehmen, an manchen Stellen zu erweitern und in das neu entwickelte Benutzerinterface einzufügen. Es wurde keine Neuentwicklung des „Urclients“ durchgeführt. Das grundlegende Konzept der Datenübertragung wurden übernommen. Jene Klassen und Methoden des alten Clients die im neuen Client nicht mehr benötigt werden, wurden entfernt.

Die Abbildung 7.1 zeigt die Architektur des Clients. Der Client besteht aus fünf Hauptklassen. Diese Hauptklassen sind:

[2] http:\\ameise.uni-klu.ac.at

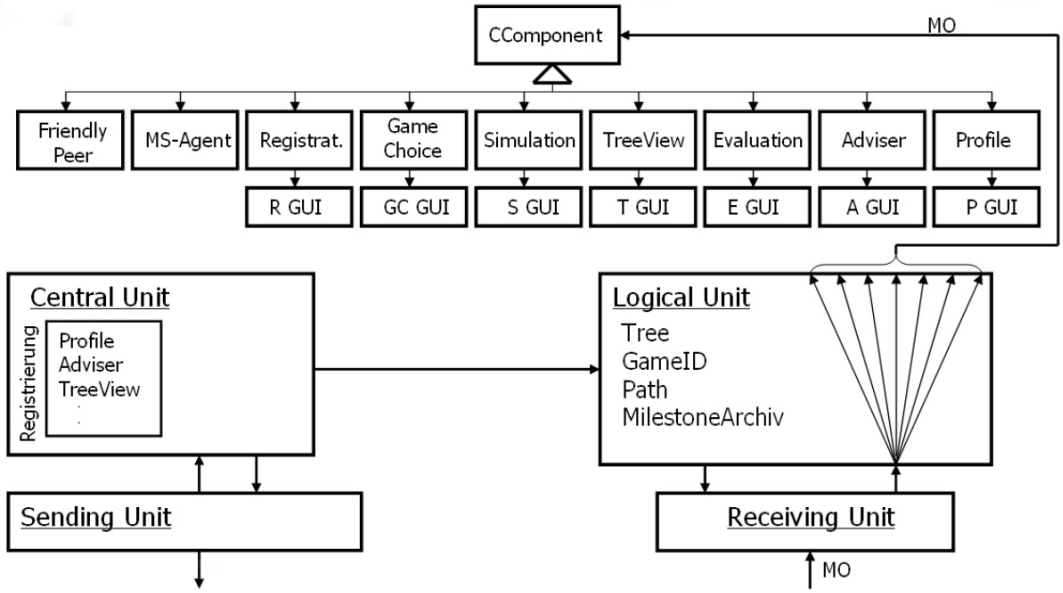

Fig. 7.1: Architektur des Clients

- CentralUnit
- LogicalUnit
- SendingUnit
- ReceivingUnit
- Client

Die CentralUnit dient als Container für die einzelnen Teile des Clients. Sie enthält die registrierten Komponenten, die LogicalUnit und die SendingUnit. Die einzelnen Komponenten nützen diese Klasse, um auf die SendingUnit und die LogicalUnit zuzugreifen.

Die LogicalUnit speichert alle relevanten Informationen und Datenstrukturen des Clients. Sie verarbeitet weiters alle zu sendenden und zu empfangenen Datenobjekte. In dieser Klasse wird die Verteilung dieser Datenobjekte an die interessierten Komponenten realisiert. Die Information darüber, welche Komponenten registriert sind, erhält sie durch die CentralUnit, in der sich alle Komponenten anmelden müssen.

Dabei gilt:

- Die SendingUnit ist für das Versenden der Datenobjekte an den Wrapper verantwortlich.
- Die ReceivingUnit ist für das Empfangen der Datenobjekte zuständig.
- Der Client ist die Hauptklasse des Clients. Neben den Informationen über das GUI, wird hier auch die Mehrsprachigkeit des Clients implementiert.

Wie aus der Abbildung 7.1 ersichtlich ist, wird die Verteilung der Datenobjekte über die Klasse „CComponent“ realisiert. Sie stellt die Elternklasse der spezialisierten Komponenten dar und beinhaltet abstrakte Methoden, die von den Komponenten überschrieben werden müssen, um die gewünschte Funktionalität zu erhalten.

Die Vielzahl der Komponenten sind in einen Logikteil und in einen Grafikteil eingeteilt. Das Zusammenspiel geht dabei wie folgt von statten: Der Logikteil besitzt die Möglichkeit Datenobjekte zu empfangen, zu bearbeiten und wieder zu senden, die „GUI“ Klassen der Komponenten besitzen dabei nur die Aufgabe die Inhalte der Datenobjekte zu repräsentieren oder die Ergebnisse der Bearbeitung darzustellen.

Zu dem Komponenten des Clients zählen:

- Profile (ProfileGUI)
- Advice (AdviceGUI)
- Evaluation (EvaluationGUI)
- Treeview / TreeviewGUI
- Simulation / SimulationGUI
- Registration / RegistrationGU
- Gamechoice / GamechoiceGUI
- MS Agent
- Friendly Peer

In den folgenden Zeilen werden die einzelnen Komponenten kurz beschrieben. Zu allen Komponenten, die eine grafische Benutzeroberfläche besitzen, wird auch ein Screenshot das Benutzerinterface.

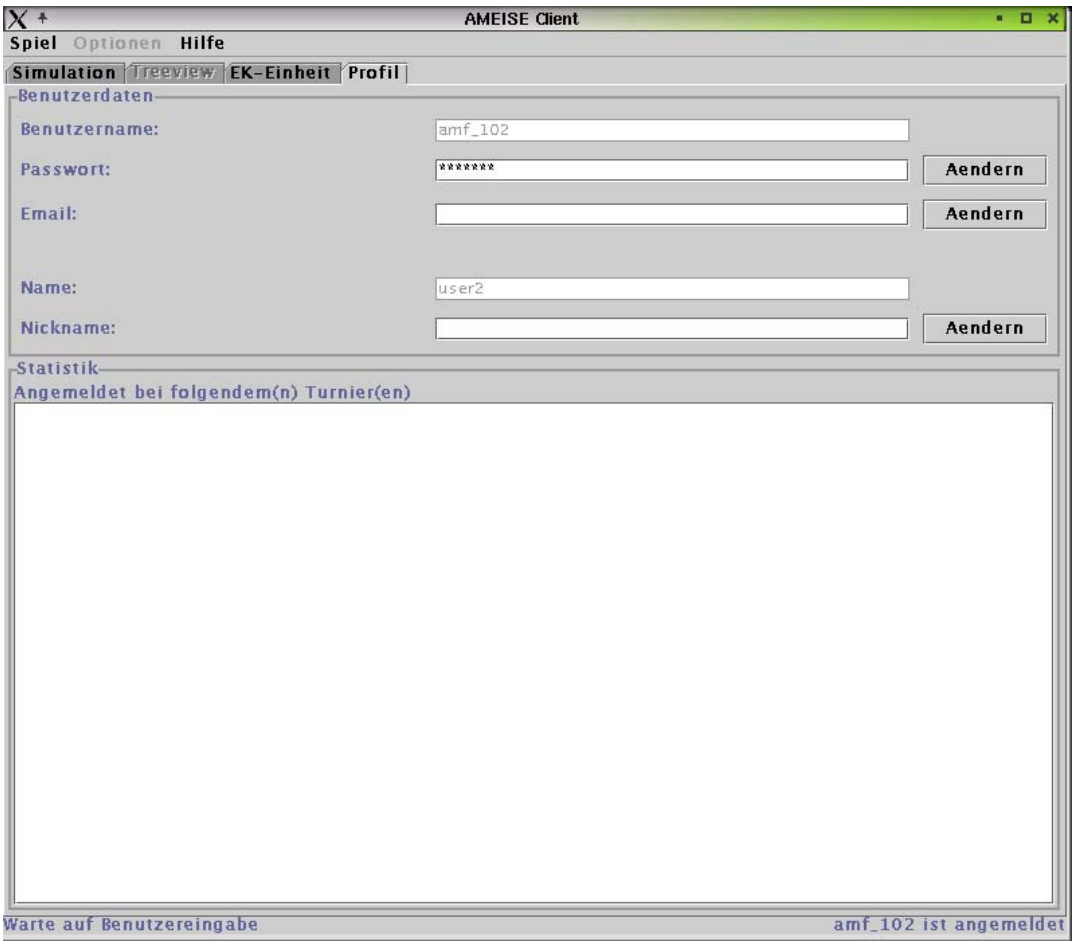

Fig. 7.2: Das ProfileGUI

Die Spielerprofilkomponente („Profile“) besitzt die Aufgabe, die Daten der Spieler anzuzeigen und Veränderungen des Spielerprofils zu ermöglichen.

Das ProfileGUI ist der Abbildung 7.2 ersichtlich. Zu diesen Spielerdaten zählt der Benutzername des Spielers, das Passwort, die Emailadresse, der tatsächliche Name des Spieler und der Spitzname (Nickname). Dem Spieler wurde in dieser Komponente die Möglichkeit gegeben, Passwort, Emailadresse und Spitzname zu verändern. Die anderen Benutzerdaten können vom Spieler nicht verändert werden. Ebenso soll in Zukunft ein Statistikfenster angezeigt werden, dass Informationen, über die bereits gespielten oder begonnenen Spiele des Spielers anzeigt. Diese Komponente wurde bereits entwickelt.

Die Auswertungskomponente („Evaluation“) besitzt die Fähigkeit, dem Spieler Informationen über das Spiel zu vermittelt. Die gewünschten Auswertungen werden aus der Datenbank ausgelesen und als Liste von möglichen Auswertungen, dem Spieler präsentiert. Die Bearbeitung erfolgt dann ebenfalls mit Hilfe der Datenbank. Diese Auswertungsinformationen werden in Textform oder anhand von Grafiken dargestellt. Der Spieler hat während und nach dem Spiel die Möglichkeit zu diesen Informationen zu gelangen. Werden die Spielinformationen nach dem Spiel abgefragt, spricht man von Endauswertung, erfolgt die Informationsabfrage während des Spiels, so spricht man von der Zwischenauswertung. Bei der Zwischenauswertung kann es vorkommen, dass nicht alle relevanten Daten für diese Auswertung zu Verfügung stehen. In solchen Fällen kann dann keine Auswertung oder nur eine Teilauswertung (Auswertung nicht im vollen Umfang) gemacht werden.

Die Auswertungskomponente wird in der Abbildung 7.3 dargestellt. Diese Abbildung zeigt neben der Auswertungsoberfläche auch die Antwort des Systems. Die Auswertungsoberfläche ist in zwei Teile eingeteilt. Der obere Teil nimmt nach der Beendigung des Spiels, die Reaktionen des Kunden entgegen. Bei einer Zwischenauswertung bleibt dieses Feld leer. Im unteren Teil des Auswertungsbildschirms, kann der Spieler ein Bewertungskriterium auswählen. In der Abbildung 7.3 wurde das Budget als Auswertungskriterium ausgewählt. Durch Drücken des „Anzeigen“ Knopfes wird die Auswertung gestartet und der Spieler erhält die Antwort des Systems. Neben der hier gezeigten textuellen Antwort, gibt es auch grafisch / textuelle Antworten der Auswertung.

Die Abbildung 7.4 zeigt die grafische / textuelle Antwort der Auswertungskomponente.

Das Antwortfenster ist vertikal in zwei Teile unterteilt. Der obere Teil des Fensters nimmt die erzeugte Grafik auf, im unteren Fenster wird der Erklärungstext ausgegeben.

Die Auswertungskomponente ist Teil der Diplomarbeit von Fr. Susanne Jäger.

Die Ratgeberkomponente („Advice“) hat die Aufgabe dem Spieler während des Spieles Ratschläge zu geben. Die Auswahlliste mit möglichen Ratschlägen, anhand derer ein Ratschlag erteilt werden kann, wird aus der Datenbank ausgelesen. Der Spieler wählt dann einen dieser Punkte aus, und das System ermittelt den Ratschlag. Dieser Ratschlag wird ebenfalls aus Daten der Datenbank erzeugt. Die Ratschläge dieser Komponente werden dem Spieler in Textform präsentiert. Die Ratgeberkomponente wurde bereits implementiert. Das RatgeberGUI wird in Kapitel 8.1.3 auf Seite 80 dargestellt.

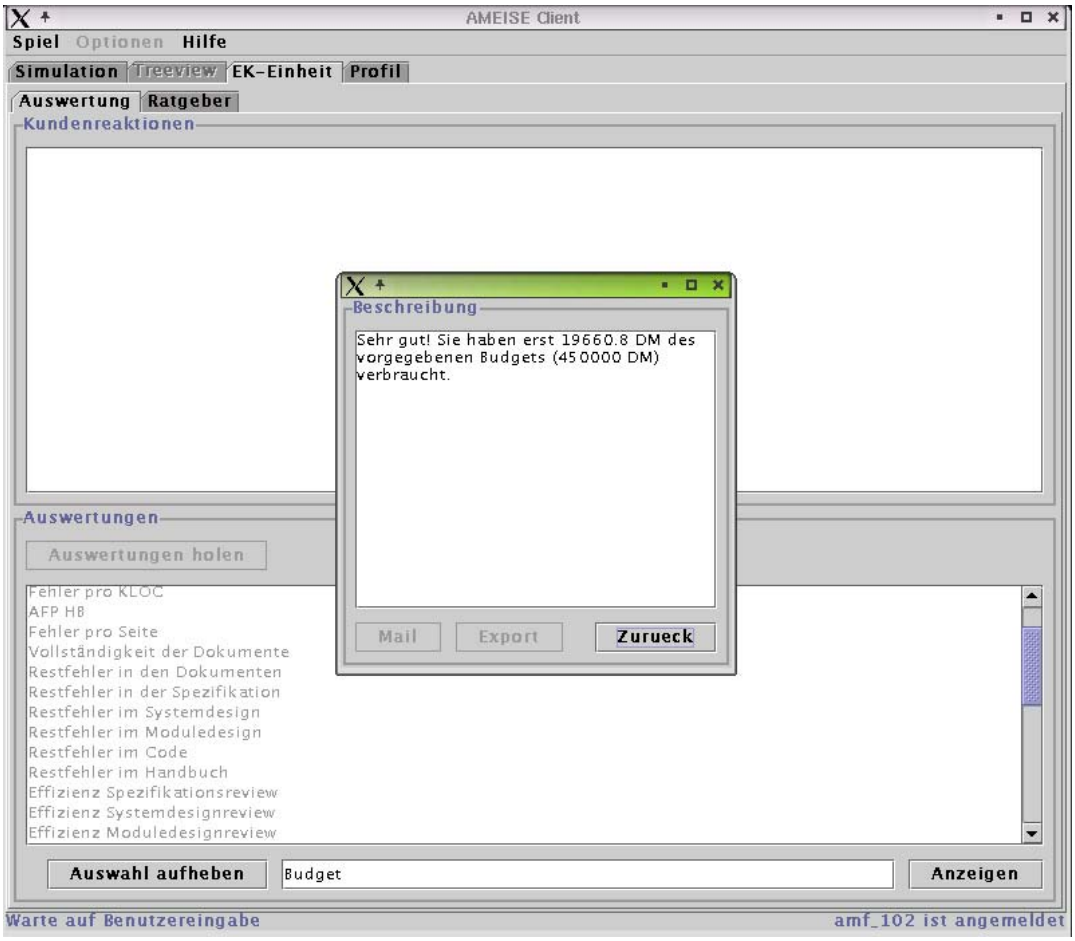

Fig. 7.3: Die Auswertungskomponente

Die Treeviewkomponente besitzt die Aufgabe den Spielbaum eines Spielers darzustellen. Dabei soll die Möglichkeit bestehen, die Ansicht auf diesen Spielbaum zu verändern. Jene Knoten des Spielbaumes, die Milestones repräsentieren, werden extra markiert dargestellt. An diesen Punkten besitzt der Spieler die Möglichkeit, sein Spiel zurückzusetzen. Das bedeutet, dass der Spieler die Möglichkeit bekommt, begangene Fehler auszubessern, da es möglich ist, jene Stellen an denen die Fehler passiert sind, noch einmal zu spielen. Durch dieses Rücksetzen wird ein neuer Ast im Spielbaum erzeugt. Weiters wird dem Spieler die Möglichkeit gegeben, einzelne Knoten im Spielbaum zu beschriften. Der Treeviewkomponente wird eine weitere wichtige Aufgabe zu teil: über diese Komponente wird der Vergleich dieses Spieles mit anderen Spielen angestoßen. Diese Komponente ist zurzeit noch nicht

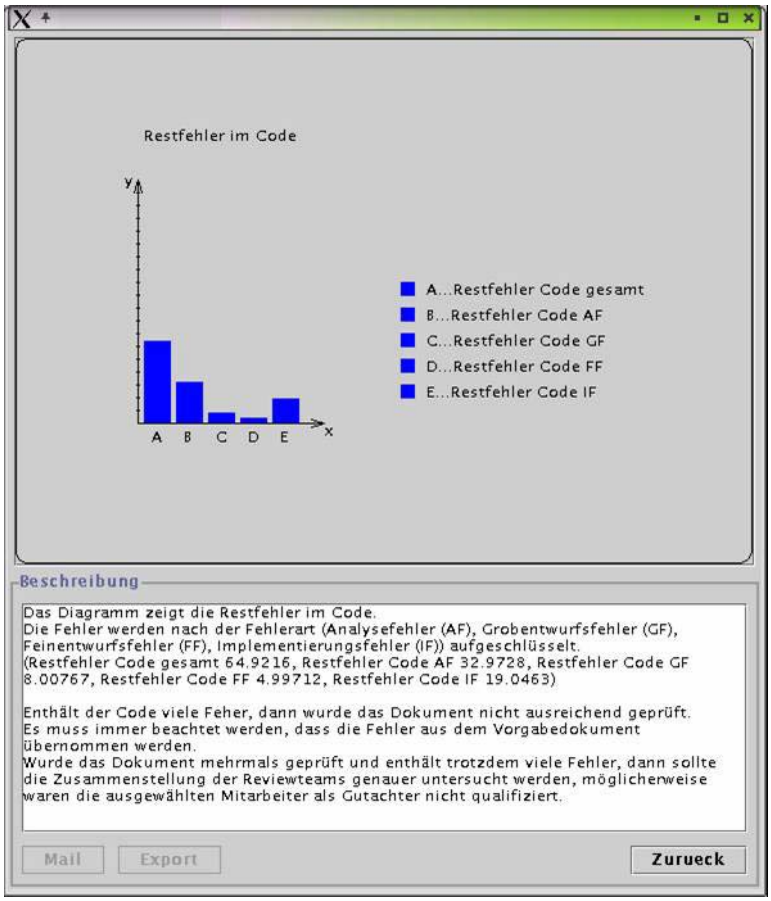

Fig. 7.4: Grafische Auswertungsergebnis

implementiert.

Die Simulationskomponente („Simulation“) ist die wichtigste grafische Komponente des Clients. In dieser Komponente wird das Spiel gespielt. Sie besitzt die Aufgabe dem Spieler die Interaktion mit dem Simulator zu ermöglichen. Alle Spieleingaben bzw. Antworten des Systems werden von dieser Komponente behandelt.

Die Simulationsoberfläche wird in Abbildung 7.5 dargestellt. Die Befehlseingabe erfolgt über das Eingabefeld. Bestätigt wird ein Befehl durch „Return“. Danach wird der Befehl im System verarbeitet und eine Antwort wird zurück an den Client geschickt. Diese Antwort wird zusammen mit dem Befehl in der Simulationsumgebung angezeigt. In der Abbildung 7.5 kann man mehrere abgesetzte Befehle und erhaltene Antworten sehen.

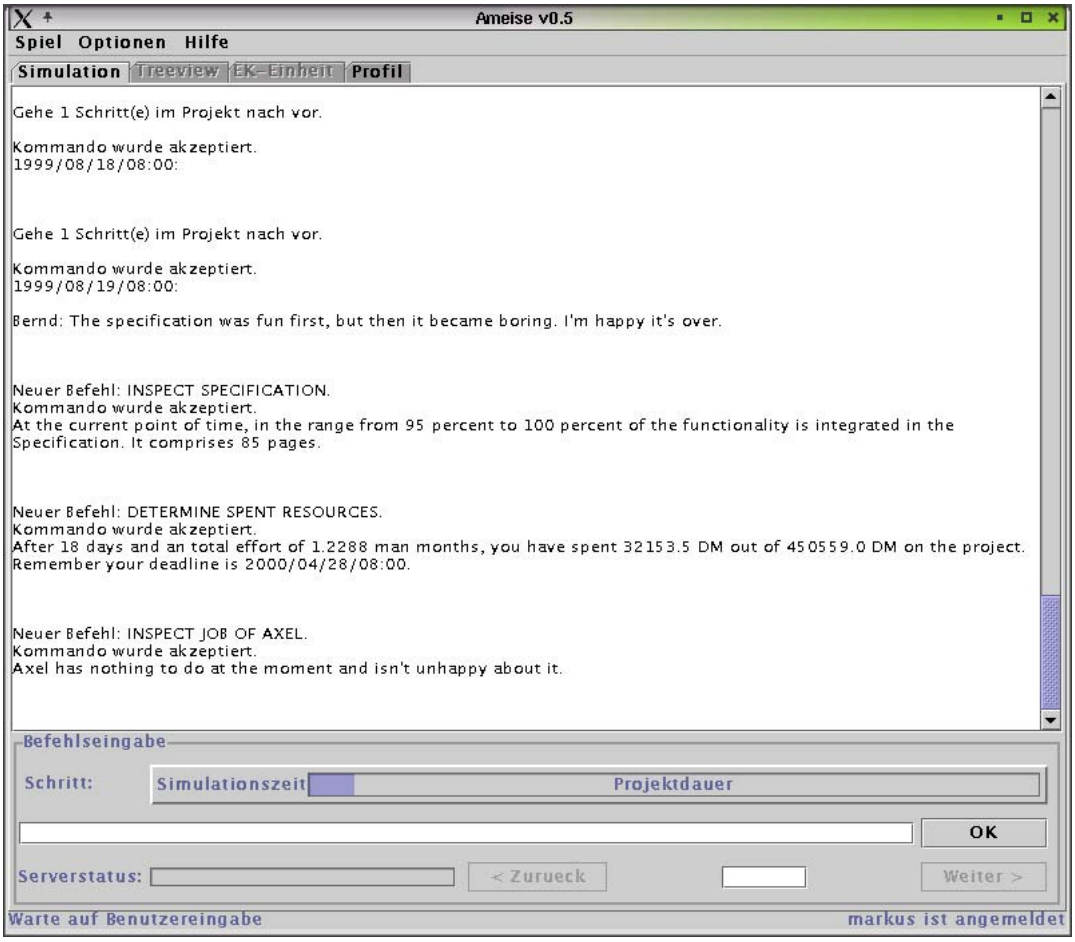

Fig. 7.5: Simulationsoberfläche

Die Anmeldekomponente („Registration“) ist die erste Komponente mit der sich der Spieler befassen muss. Nach dem Start des Systems muss sich der Spieler anmelden. Dieses Anmelden wird über die Befehlsleiste aufgerufen.

Die Abbildung 7.6 zeigt das Anmeldefenster, wie es der Spieler vorfindet. Nachdem der Benutzer den Namen und das Passwort eingetragen hat und dies mit „OK“ bestätigt, wird die Prüfung der Daten in der Datenbank angestoßen. Bei dieser Prüfung wird neben der Prüfung der Spielereingaben auch kontrolliert, ob der Spieler noch berechtigt ist, sich am Client anzumelden. Nach erfolgreicher Anmeldung, muss der Spieler noch ein Spiel auswählen, bevor er tatsächlich mit einem Spiel beginnen kann.

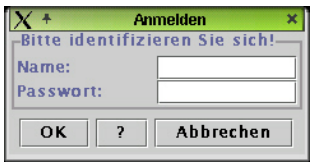

Fig. 7.6: Anmeldefenster

Die Spielauswahlkomponente („Game Choice“) stellt die Spiel- / Turnierauswahl dar. Nach der erfolgreichen Anmeldung am System, muss sich der Spieler zu einem Spiel anmelden. Dieses Anmelden am Spiel wird von dieser Komponente durchgeführt. Die Komponente holt sich aus der Datenbank alle möglichen Spiele, an denen dieser Spieler teilnehmen darf. Diese verschachtelte Informationen werden dann in einem Baum abgelegt.

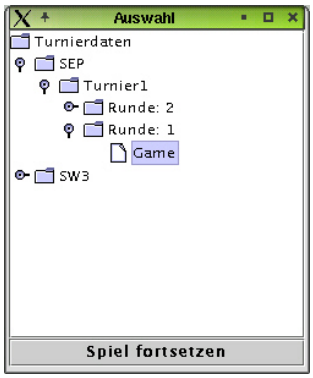

Fig. 7.7: Spielauswahlfenster

Die Abbildung 7.7 zeigt das Auswahlfenster. Die Verschachtelung beginnt mit dem Kurs, gefolgt vom Turnier, der Runde und die letzte Stufe der Verschachtelung ist das Spiel. In einem Kurs kann es mehrere Turniere geben, pro Turnier können mehrere Runden gespielt werden und in jeder Runde existiert ein Spiel. Nach der Spielauswahl ist der Client zum Spielen bereit. Diese Komponente wurde bereits implementiert.

Die Markierungsagent („MS Agent“) und die väterliche Freund Komponente („friendly peer“) unterscheiden sich von den anderen Komponenten dadurch, dass sie keine „GUI“ Klasse besitzen. Die väterliche Freund Komponente besitzt keine „GUI“ Klasse, weil die Ergebnisse der Bearbeitung keines eigenen GUIs bedürfen. Diese Ergebnisse werden dem Spieler über das GUI der Simulationskomponente angezeigt. Die zweite Komponente, die kein GUI

besitzt ist der Markierungsagent. Diese Komponente benötigt kein GUI, weil die Ergebnisse der Bearbeitung der Datenobjekte durch diese Komponente dem Benutzer verborgen bleiben sollen. Ihre Ergebnisse werden von anderen Komponenten ermöglicht. Siehe dazu auch Kapitel 5.

7.2 Erweiterungsmöglichkeiten

Die Architektur des Clients ist für das Hinzufügen neuer Komponenten ausgelegt worden und es ist ein leichtes, neue Komponenten in die bestehende Architektur einzubinden. An dem Sende- und Empfangseinheiten muss bei einer solchen Erweiterung nichts geändert werden.

Wenn neue Komponenten zum System hinzugefügt werden, müssen am Client folgende Veränderungen durchgeführt werden:

- Neuen Komponenten müssen in der CentralUnit registriert werden.
- Die abstrakten Methoden der Klasse „CComponent“ müssen in den neuen Komponentklassen angepasst werden.
- Wenn neue Transportobjekte benötigt werden, dann müssen diese mit Identifieren versehen werden. Diese Identifier müssen dann in den Klassen eingetragen werden, die sich mit den Transportkonstanten beschäftigen. Ebenso muss die Verarbeitung dieser Transportobjekte auf die neuen Komponenten angepasst werden. Die Verarbeitung der Transportobjekte erfolgt in der Klasse „LogicalUnit“.
- Besitzt die Komponente ein eigenes GUI, so muss es an der richtigen Stelle eingebettet werden. Da das ClientGUI generisch gehalten wurde, ist das Hinzufügen neuer KomponentenGUI kein Problem. Größere GUIs besitzen am Client einen eigenen Reiter. Derzeit gibt es im Client vier Reiter. Diese Reiter beinhalten die Simulation, den Treeview, die EK-Einheit und das Spielerprofil. Soll die neue Komponente auch ein eingeständiges GUI besitzen, dann kann einfach ein neuer Reiter eingefügt werden.
- Auch für die rein textuelle Interaktion der Komponenten mit dem Spieler wurden bereits Klassen zur Verfügung gestellt, die diese Art der Interaktion ermöglichen.
- Der Aufruf dieser neuen Komponenten muss an der richtigen Stelle stattfinden. Am Client gibt es die Möglichkeit, Komponenten an den

unterschiedlichsten Stellen aufzurufen. Beispiele für solche Stellen sind das Menü des Clients, vom einem der GUIs aus, oder der Aufruf erfolgt durch das Eintreten gewisser Situationen im Spielverlauf (dynamischer Aufruf).

Nicht nur das Hinzufügen neuer Komponenten ist einfach, auch das Erweitern der Mehrsprachigkeit ist sehr einfach. Die Mehrsprachigkeit wurde durch die Klasse ResourceBundle realisiert, die Teil der Java Programmiersprache ist. Das ResourceBundle verwendet für das Auslesen der unterschiedlichen Sprachen Dateien, die die verschiedenen Sprachen beinhalten. Diese Dateien beinhalten Schlüssel und den entsprechenden Spracheintrag. Diese Schlüssel werden im Code verwendet und in Abhängigkeit der gerade aktuellen Sprache wird der entsprechende Schlüsseleintrag aus der Sprachdatei ausgelesen. Für eine Erweiterung der Mehrsprachigkeit um eine Sprache muss einfach eine neue Datei angelegt werden.

Kapitel 8

IMPLEMENTIERUNG UND EINSATZ

Diese Kapitel beschreibt die Implementierung des Ratgebers, des väterlichen Freundes und beinhaltet Anwendungsbeispiele der beiden Komponenten.

8.1 Beschreibung der Komponenten und Anwendungsbeispiele

Das folgende Kapitel befasst sich mit der Beschreibung der Komponenten. Zu diesen zählen: der Markierungsagent, der Ratgeber und der väterliche Freund. Ebenso soll die verwendete Architektur kritisch betrachtet werden.

8.1.1 Die Clientarchitektur

In diesem Abschnitt wird die bestehende Architektur des Clients, die bereits im Kapitel 7.1.2 auf Seite 62 beschrieben wurde, genauer behandelt. Trifft die Architektur die Anforderungen, die an interaktive Anwendungen gestellt werden. Das am meisten verwendete Designparadigma für interaktive Anwendungen ist das Modell-View-Controller (MVC) Framework. Dieses Framework wird in dem folgenden Abschnitt kurz erklärt.

Das MVC Framework

Das Model-View-Controll Konzept beschreibt, wie interaktive Benutzerschnittstellen zu designen sind. Bei vielen Entwicklern hat es sich als eine sinnvolle Struktur herausgestellt, mit dessen Hilfe es möglich ist, einfach objektorientierten Software zu schreiben. Die Wartbarkeit von Software wird dadurch ebenso verbessert, wie die Möglichkeit, Teile der Anwendungssoftware wiederzuverwenden.

Es besteht aus drei Teilen: dem Modell (Model), der Sicht (View) und dem Kontroller (Controller). Das Modell repräsentiert die unterliegende, logische Struktur der Daten einer Applikation und beinhaltet Funktionen, die mit dem Modell assoziiert sind. Die Sicht beinhaltet eine Sammlung von Klassen, die

die grafischen Elemente des Benutzerinterfaces repräsentieren. Der Kontroller (Controller) verbindet die Sichten mit den Modellen und beschäftigt sich mit der Benutzereingabe.

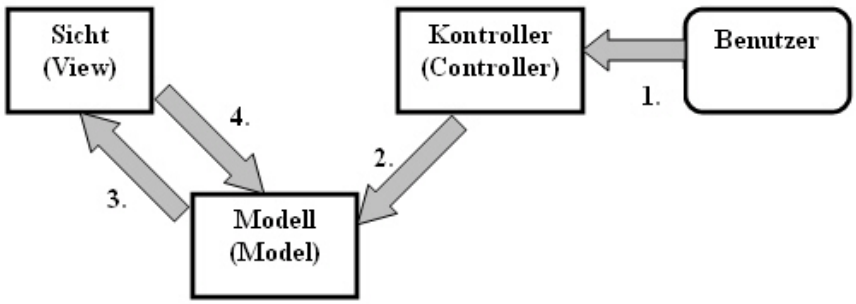

Fig. 8.1: Das MVC Kommunikationsmodell

Die Abbildung 8.1 zeigt das MVC Kommunikationsmodell. Anhand der Abbildung wird eine typische Kommunikation zwischen den Komponenten gezeigt. Um den Ablauf auch zeitlich korrekt darzustellen, wurden die Verbindungen zwischen den Komponenten nummeriert.

Kommunikationsablauf:

1. Der Benutzer führt eine Tätigkeit durch.
2. Der Kontroller interpretiert diese Tätigkeit, als Zustandsänderung des Modells und informiert es darüber.
3. Das Modell ändert seinen Zustand und informiert die Sicht, über diese Veränderung.
4. Die Sicht fragt das Modell nach dessen derzeitigen Zustand und verändert sich dementsprechend.

Vorteile des MVC Paradigmas:

- Verschiedene Sichten auf dasselbe Modell können realisiert werden.
  Die Trennung des Modells von den Sichten erlaubt es, auf dasselbe Modell verschiedene Sichten zu haben. So besitzt zum Beispiel der Buchhalter eine andere Sicht auf das System, als die Sekretärin.

- Einfache Unterstützung neuer Clienttypen.
  Um einen neuen Typ von Clienten zu erstellen, benötigt man eine neue Sicht und einen neuen Kontroller und verbindet diese zum bestehenden Modell. Am Modell müssen keine Veränderungen durchgeführt werden.

- Das Design wird übersichtlich.
  Die Entwicklungen der Applikationen werden einfacher und leichter wartbarer.

- Effiziente Modularisierung.
  Da die Applikation in unabhängige Teile aufgeteilt ist, können diese Teile ohne Aufwand ausgetauscht werden. Es ist sogar möglich, verschiedene Komponenten parallel zu entwickeln, da die Schnittstellen zwischen ihnen klar definiert sind.

- Die Applikation kann einfach erweitert werden.
  Diverse Erweiterungen des Systems sind jederzeit möglich, solange die gemeinsamen Schnittstellen aufrecht erhalten bleiben.

- Applikationen können verteilt werden.
  Mit einer einfachen Änderung der Startmethoden können MVC Anwendungen auf eine Reihe von Proxies verteilt werden. Es entsteht kein zusätzlicher Aufwand durch diese Verteilung.

Die Architektur des Ameiseclients ist dem MVC Konzept ähnlich, unterscheidet sich aber an einigen Stellen. In folgenden werden die Unterschiede genauer beschrieben.

Beim MVC Modell erfolgt eine strikte Dreiteilung, in die Verarbeitung, die Benutzeroberfläche und die Benutzerinteraktion. Diese rigorose Dreiteilung findet am Ameiseclient nicht statt. Am Ameiseclient ist die Benutzeroberfläche und die Benutzerinteraktion in einer Klasse vereint. Dieses Zusammenziehen der Sicht und des Kontrollers wird auch in der Literatur [Prosise 1999], [Althammer und Pree] beschrieben. In diesen beiden Arbeiten wird die Tatsache erwähnt, dass die Benutzerinteraktionskomponenten nicht voll von den Benutzeroberflächen unabhängig sein können, weil sie die Events verstehen müssen, die von den Benutzeroberflächen getriggert werden. Deshalb benötigt jede Sicht einen eigenen Kontroller. Auf Grund dieser Tatsache sind die Autoren der Meinung, dass die strikte Trennung der Kontroller und der Sichten aufgehoben werden sollen. Moderne, auf der MVC Architektur basierende Bibliotheken wie Swing [R.Eckstein 1998], machen dies.

Durch die Verbindung geht aber die Austauschbarkeit und Widerverwendbarkeit der Benutzerinteraktionkomponente verloren. Im Ameiseclient ist die

Widerverwendbarkeit und Austauschbarkeit der Benutzerinterakionskomponenten nicht möglich, da sie zu stark an die Darstellung gebunden sind.

Die Architektur des Clients wird in Kapitel 7.1.2 auf Seite 7.1.2 gezeigt.

Ein weiterer Unterschied zwischen der MVC Architektur und der Ameisearchitektur findet man beim Modell. Dieses Modell soll laut MVC neben der logischen Struktur auch noch die Aufgabe besitzen, einen Benachrichtigungsmechanismus zu implementieren. Dieser Benachrichtigungsmechanismus hat die Aufgabe, die Sichten und die Kontroller von eventuellen Veränderungen des Modells zu informieren. Diese Veränder-ungen werden durch die Anwendungssoftware über den Kontroller durchgeführt. Die Möglichkeit Veränderungen durchzuführen ist aber nicht Teil des Ameisesystems. Der Spieler hat im Ameisesystem nicht die Möglichkeit das Modell zu verändern, weshalb das Modell des Ameiseclients keinen Benachrichtigungsmechanismus implementiert.

Die Darstellung von unterschiedlichen Sichten auf gleiche Inhalte ist einer der angepriesenen Vorteile der MVC Architektur. Der Ameiseclient ist für Spieler entwickelt worden. Jeder Spieler hat die gleiche Sicht auf das System. Die Oberfläche von Spieler A unterscheidet sich nicht von der Oberfläche von Spieler B. In Zukunft bekommen die Spieler die Möglichkeit (siehe Kapitel9.1 auf Seite 90) die Art der Interaktion mit dem System selbst zu bestimmen. Alle Spieler sollen in Zukunft diese Möglichkeit besitzen. Es gibt keine besonderen Personen, die andere Rollen besitzen und deshalb andere Sichten auf die gleichen Daten besitzen.

Da sich das Ameisesystem nur leicht von der MVC Architektur unterscheidet, entstehen bei der Wartung und der Wiederverwendung des Ameisesystems keine Probleme, die dazu führen, dass zusätzliche Veränderungen des Clients durchgeführt werden müssen.

8.1.2 Der Markierungsagent

In diesem Kapitel werden mittels Sequenzdiagramme die Abläufe des Markierungsagenten dargestellt. Generelle Informationen über den Markierungsagenten und Markierungsbeispiele wurde bereits im Kapitel 5.7 auf Seite 48 ausführlich dargestellt.

Da der Markierungsprozess über mehrere Stufen verläuft, repräsentieren auch mehrere Diagramme diesen Ablauf.

Die Abbildung 8.2 zeigt das Anfordern der Milestones aus der Datenbank.

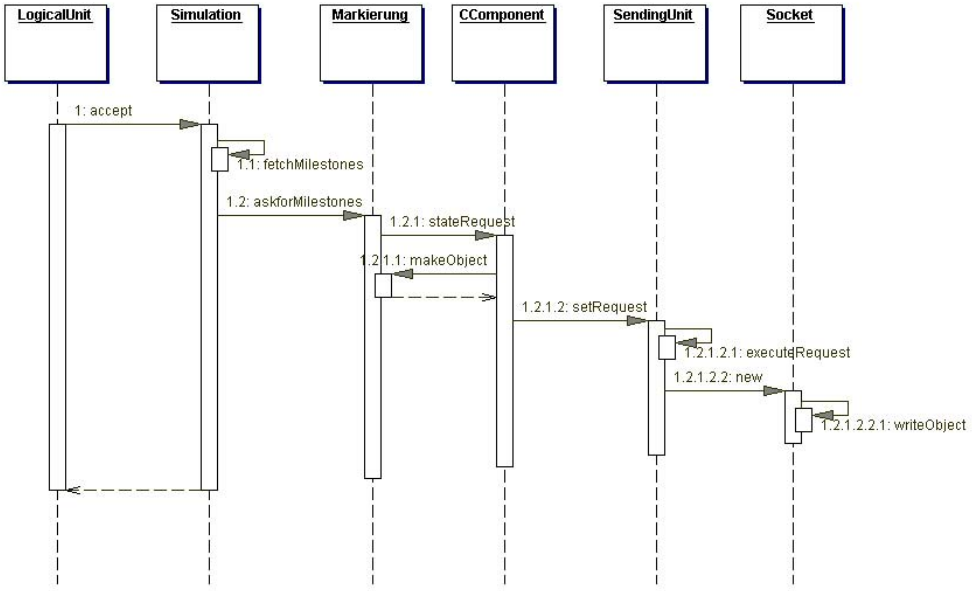

Fig. 8.2: Aufforderung an den DB Server, die Milestones zu schicken

Auf Grund dieser Anforderung werden die Milestonedaten ausgelesen und am Client gespeichert. Die Milestonedaten, die für die korrekte Markierung von Spielzügen zuständig sind, werden ab dem Beginn des Spieles benötigt. Aus diesem Grund werden sie sofort nach dem Start des Clients aus der Datenbank ermittelt.
Anhand dieser Abbildung kann der Kommunikationsprozess des Clients anschaulich dargestellt werden. Will eine Komponente eine Anfrage an die Datenbank oder den Wrapper schicken, so setzt diese den Befehl „stateRequest“ ab. Mittels der Methode „makeObject“ wird der Komponente die Möglichkeit gegeben, ihre Anfrage zu stellen und in ein geeignetes MessageObjekt zu verpacken. Dieses MessageObjekt wird an die Klasse „CComponent“ übergeben. In dieser Klasse wird mittels des Befehls „setRequest“ die Klasse „SendingUnit“ aufgerufen und das erstellte MessageObjekt wird übergeben. Die Aufgabe der „SendingUnit“ besteht nun darin, einen Verbindung zum LBM herzustellen und die Abfrage abzuschicken. Bei jeder Abfrage einer Komponente an die Datenbank oder den Wrapper wird die gleiche Aufrufsstruktur verwendet.

Nach dem die Milestonedaten nun am Client gespeichert wurden, zeigt die nächste Abbildung den nächsten Schritt des Markierungsablaufs.

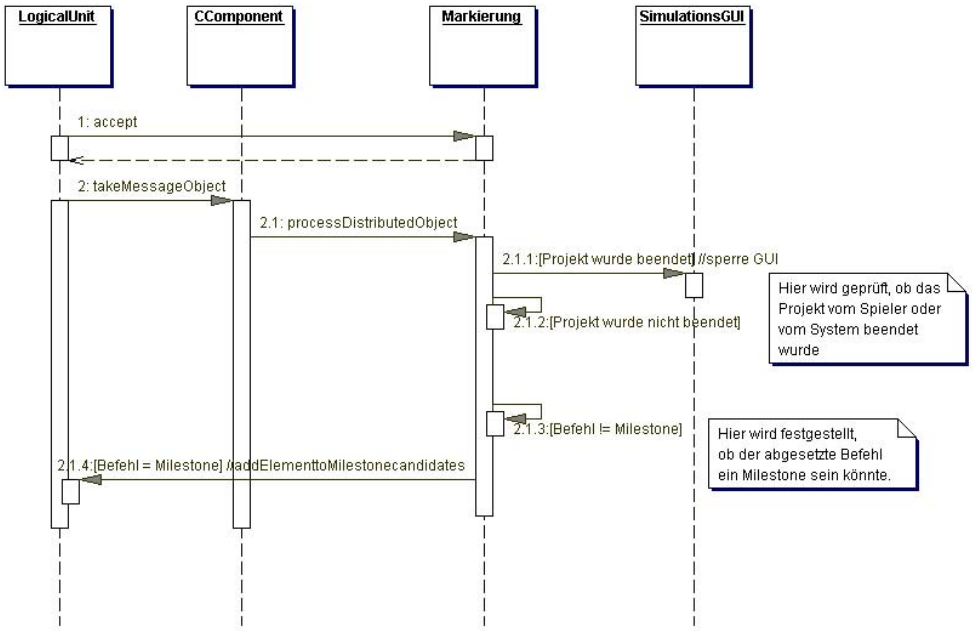

Fig. 8.3: Identifizieren und Abspeichern der Milestonekandidaten

Das Setzen der Milestonekandidaten stellt den nächsten Schritt im Markierungsprozess dar. Die Abbildung 8.3 zeigt diesen Ablauf. Der hier dargestellte Ablauf, stellt einen Teil des Verteilungsmechanismus des Clients dar. Die vollständige Verarbeitung eines erhaltenen MessageObjekts ist in Kapitel ... ab Seite ... ersichtlich. Die „LogicalUnit“ fragt die Markierungsklasse, ob sie an diesem MessageObjekt interessiert ist. Ist dies der Fall, so wird der Komponente „Markierung“ das MessageObjekt übergeben. Diese Übergabe wird durch die Klasse „CComponent“ realisiert. Ist das MessageObjekt in der Klasse Markierung angekommen, wird es verarbeitet. Wird anhand des Feedbacks des Systems erkannt, dass der Simulator beendet wurde, so wird auch dem Benutzer die möglichkeit entzogen weitere Befehle abzusetzen. Um dies zu Realisieren, werden Teile der SimulationsGUI gesperrt. In einer anschließenden Überprüfung wird kontrolliert, ob der abgesetzte Befehl Teil eines Milestoneeintrages ist. Ist dies der Fall, bedeutet es für den Markierungsprozess, dass dieser Befehl ein möglicher Milestonekandidat ist. Deshalb wird dieser Milestoneeintrag der mit dem Befehl verglichen wurde, in die Liste der Milestonekandidaten aufgenommen.

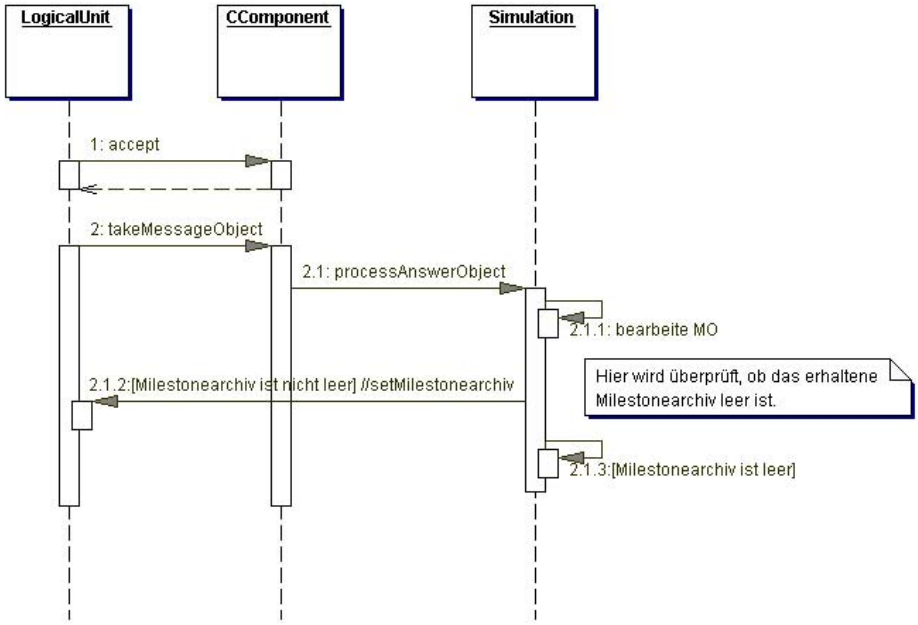

Fig. 8.4: Setzen des Milestonearchivs

In der Abbildung 8.4 ist ersichtlich, wie das Milestonearchiv am Client gesetzt wird. Das Setzen des Milestonearchivs wird in der Klasse „Simulation“ durchgeführt. Da das Milestonearchiv und die Milestonekandidaten bei jedem „Proceed“ zur Verarbeitung an den LBM weitergeleitet werden, beinhaltet das Ergebnis dieser Verarbeitung das aktuelle Milestonearchiv. Das erhaltene Milestonearchiv wird dann in der Klasse „LogicalUnit“ abgespeichert.

Die Abbildung 8.5 zeigt den letzten Schritt des Markierungsprozesses, der am Client stattfindet. Sind das Milestonearchiv und die Milestonekandidaten gefüllt, so werden diese Einträge an die Verarbeitung übermittelt. Die Übermittlung dieser Daten an den LBM findet bei jedem „Proceed“ statt. Wie aus der Abbildung ersichtlich ist, werden die Milestonekandidaten und das Milestonearchiv in der Klasse „Simulation“ in das MessageObjekt verpackt.

Nachdem das Milestonearchiv und die Milestonekandidaten an den LBM gesandt werden, werden sie verarbeitet. Diese Verarbeitung findet am LBM

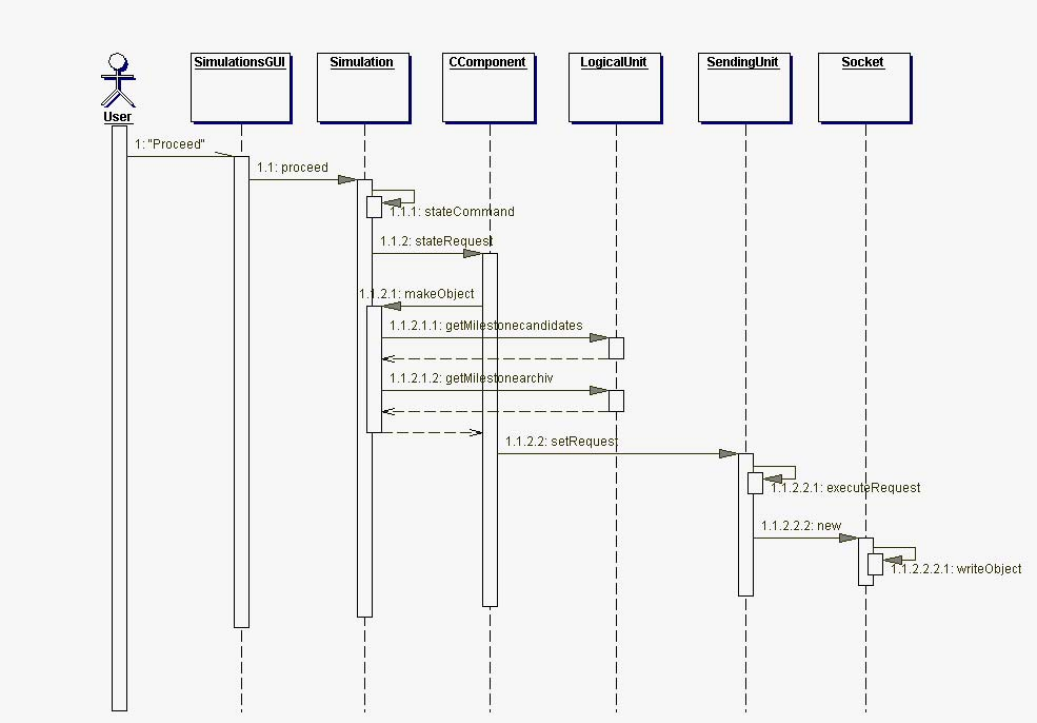

Fig. 8.5: Übertragen des Milestonearchivs und der Milestonekandidaten an die Verarbeitung

statt und wird im Kapitel 5 erklärt.

8.1.3 Der Ratgeber

Der Ratgeber besteht aus der Klasse Ratgeber und der Klasse RatgeberGUI. Die Klasse Ratgeber beinhaltet die Verarbeitungslogik, die Klasse RatgeberGUI beinhaltet das Benutzerinterface des Ratgebers.

Da der Ratgeber ein eigenes GUI besitzt, ist dieses GUI Teil der Benutzeroberfläche. Die Komponente Ratgeber und die Komponente Auswertung wurden in der grafischen Benutzeroberfläche in einer Einheit zusammengefasst. Will der Spieler im Spiel den Ratgeber aufrufen, muss er den Simulationsreiter verlassen und zum Reiter der EK-Einheit wechseln. Der Wechsel zur EK-Einheit geschieht durch das Auswählen des EK-Einheit Reiters. Da zwei Komponenten in diesem Reiter zu finden sind, muss der Spieler eine erneute Auswahl treffen, um zum Ratgeber zu gelangen.

Das Ratgeberbenutzerinterface wird in Abbildung 8.6 dargestellt.

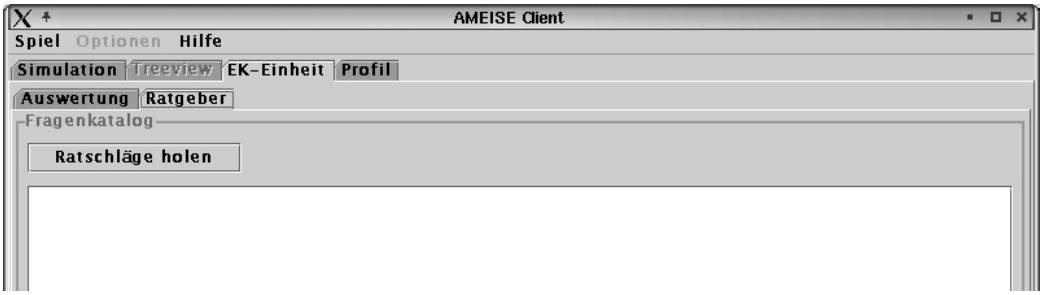

Fig. 8.6: Ratgeberbenutzerinterface (Startsituation)

Das Benutzerinterface ist horizontal in zwei Teile unterteilt. Der obere Teil beinhaltet den Fragenkatalog, der untere Teil des Benutzerinterfaces beinhaltet die Ratschläge, die den Benutzer vom System erteilt werden. Der Fragenkatalog wurde deshalb implementiert, da der Benutzer die Möglichkeit hat, zwischen mehreren Fragen des Ratgeber auszuwählen. Diese Fragen werden in der Struktur der speziellen Hilfsmittelkomponente in der Datenbank abgelegt. Wenn der Spieler den Button „Ratschläge holen“ drückt, werden die möglichen Ratschläge, die der Ratgeber geben kann, aus der Datenbank ermittelt. Der Benutzer bekommt eine Reihe von Fragen präsentiert. Diese Fragen werden exemplarisch in Abbildung 8.7 dargestellt.

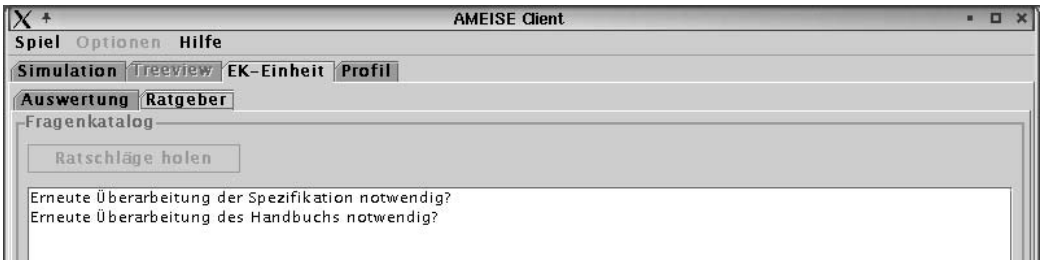

Fig. 8.7: Ratgeberbenutzerinterface (mit gefülltem Fragenkatalog)

Wird nun eine Frage vom Spieler ausgewählt, so wird die entsprechende spezielle Hilfsmittelkomponente ausgeführt und das Ergebnis der Auswertung wird dem Spieler präsentiert. Bei diesen hier angeführten Beispielen, besteht die Aufgabe der spez. Hilfsmittelkomponente darin, die jeweiligen aktuellen Werte des Spiels (AFPs der Spezifikation und AFPs des Handbuchs) aus der Datenbank auszulesen und mit vorher, in der Datenbank eingegebenen Richtwerten zu vergleichen. Die Entscheidung, welches Ergebnis nach dem Vergleich (zwischen dem Ergebnis der Auswertung und dem Richtwert) aus-

gegeben wird, liegt bei der speziellen Hilfsmittelkomponente und hängt von der ausgewählten Frage ab. Diese Antworten müssen bei der Erstellung dieser Ratgeberfragen bereits in die speziellen Hilfsmittelkomponente eingetragen werden. Ebenso ist es möglich, diese Ausgaben (und Fragen) in verschiedenen Sprachen anzuzeigen. Die Funktionsweise der speziellen Hilfsmittelkomponente, die von Frau Susanne Jäger entwickelt wurde, ist in ihre Diplomarbeit [Jaeger 2003] nachzulesen.

In der Abbildung 8.7 wurde die Frage: „Erneute Überarbeitung der Spezifikation notwendig?“ ausgewählt. Das Ergebnis der Auswertung ist in Abbildung 8.8 ersichtlich.

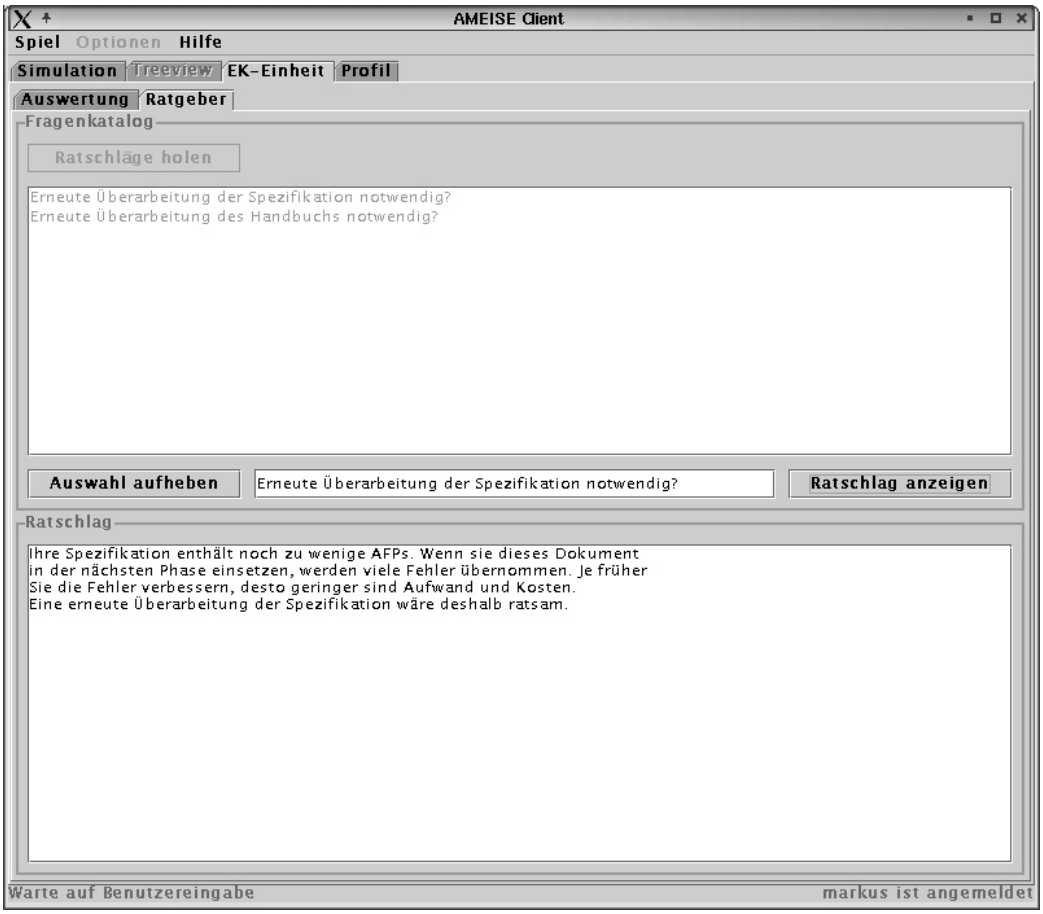

Fig. 8.8: Ratgeberbenutzerinterface (mit Ratschlag)

Dieses Ergebnis wurde auf Grund der aktuellen Situtation im Spiel ermittelt. Die Auswertung in der Abbildung 8.8 repräsentiert den negativen Fall, d.h. der Spieler besitzt zu diesem Zeitpunkt noch zu wenige AFPs in

der Spezifikationsphase. Bei dieser Antwort wird nicht nur der Ratschlag, der zu einer Verbesserung führen soll, angezeigt, sondern es werden auch die daraus resultierenden Probleme angeführt. Diese Probleme treten ein, wenn der Spieler sein Spiel, ohne einer Überarbeitung dieser, in der gewählten Frage angeführten, Phase vorsetzt.

Damit der Ratgeber keine falschen Ratschläge liefert, sollte der Ratgeber nur über bereits abgeschlossene Tätigkeiten befragt werden.

Beispiel:

Der Spieler beginnt mit der Spezifikationsphase. Ein paar Spielzüge später will er, wie in dem Beispiel von vorhin, einen Ratschlag über die Spezifikationsphase bekommen. Diese Phase ist aber noch nicht abgeschlossen. Der Ratgeber kommt bei seiner Auswertung der aktuellen Situation zum Schluss, dass die Spezifikationsphase noch nicht genügend AFPs besitzt. Da aber diese Antwort nur eine Momentaufnahme war, d.h. sie betrachtet nur die aktuellen Daten der jeweiligen Phase, weiß der Ratgeber nicht, dass diese Phase noch nicht abgeschlossen ist. Wird nun die Phase beendet, glaube der Spieler, dass sie noch nicht korrekt genug ist und überarbeitet sie noch einmal. Diese erneute Überarbeitung, die vom Ratgeber vorgeschlagen wurde, kann aber nach der Beendigung der Phase nicht mehr stimmen, da die Phase bereits den Anforderungen genügt.

Aus diesem Beispiel soll hervorgehen, dass der Ratgeber eine sehr nützliche Komponente ist, die den Spieler unterstützen kann. Wird er jedoch falsch verwendet, kann er den Spieler zu falschen Schlüssen führen. Diese Ratschläge sind nur Momentaufnahmen, die in diesem Spielzug ihre Gültigkeit besitzen, aber im nächsten Spielzug kann es vorkommen, dass sie nicht mehr zutreffen.

Wie im Kapitel des Markierungsagenten, wird auch die Funktionsweise des Ratgebers anhand von Sequenzdiagrammen dargestellt.

Die Abbildung 8.9 zeigt den Aufruf des Ratgebers. Der Spieler stößt diesen Ablauf im „RatgeberGUI“ an. Durch die Methode „send“ wird die Auswahl des Ratschlags an die „Ratgeber“ Komponente übergeben. Durch den Methodenaufruf „makeObject“ wird der ausgewählte Ratschlag in das MessageObjekt eingebunden. Der weitere Ablauf der Übertragung ist bereits aus der vorigen Kapitel bekannt. Anhand dieses ausgewählten Ratschlags wird durch die EK-Einheit am Wrapper die entsprechenden Antwort generiert und im nächsten Schritt an den Client übermittelt.

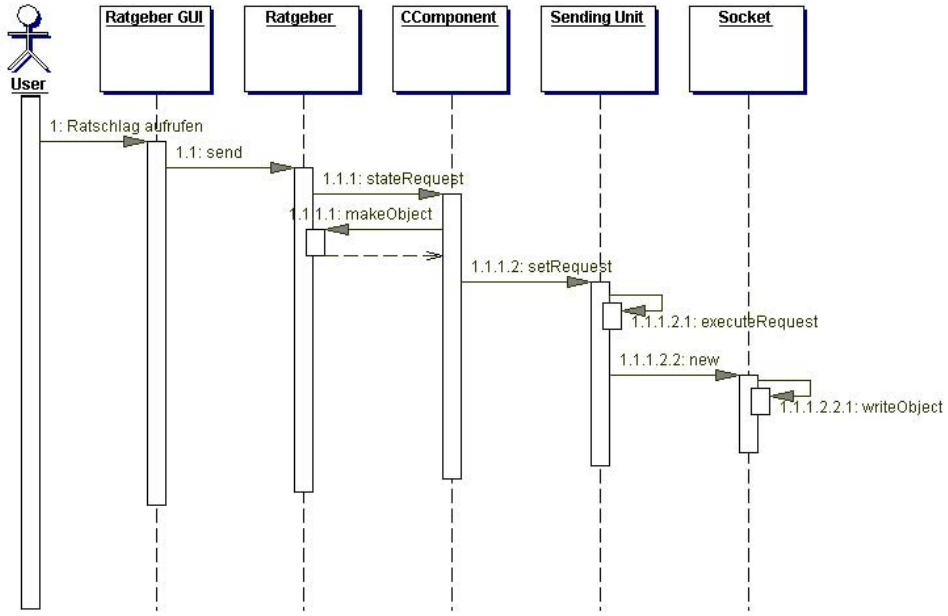

Fig. 8.9: Abschicken eines ausgewählten Ratschlags

Die Abbildung 8.10 beschreibt das Empfangen eines MessageObjekts.

Anhand dieser Abbildung 8.10 wird der vollständige Ablauf des Empfangs und die Verarbeitung von Nachrichten dargestellt. Dieser Ablauf ist, bis auf die unterschiedliche Verteilung, bei allen Komponente der Gleiche. Die „ReceivingUnit“ versucht periodisch Daten vom LBM zu erhalten. Sind Daten vorhanden, so werden diese Daten in die „LogicalUnit“ übermittelt. In der „LogicalUnit“ werden die erhaltenen MessageObjekte auf deren Korrektheit überprüft. Nach der Überprüfung werden alle Komponenten befragt, ob sie interesse an diesem MessageObjekt besitzen. Da es sich bei dem Beispiel um eine MessageObjekt handelt, dass vom Ratgeber angefordert wurde, übermittelt der Ratgeber der LogicalUnit sein Interesse an diesem MessageObjekt. Das MessageObjekt wird zuerst an die „CComponent“ übergeben, die es dann an den Ratgeber weiterleitet. Nach dem die interessierte Komponente das MessageObjekt erhalten hat, wird es entpackt und der Inhalt wird bearbeitet.

Die Verarbeitung am Ratgeber wird durch die Methode „processAnswerObject“ durchgeführt. Danach wird der erhaltene Ratgebertext zur Kompo-

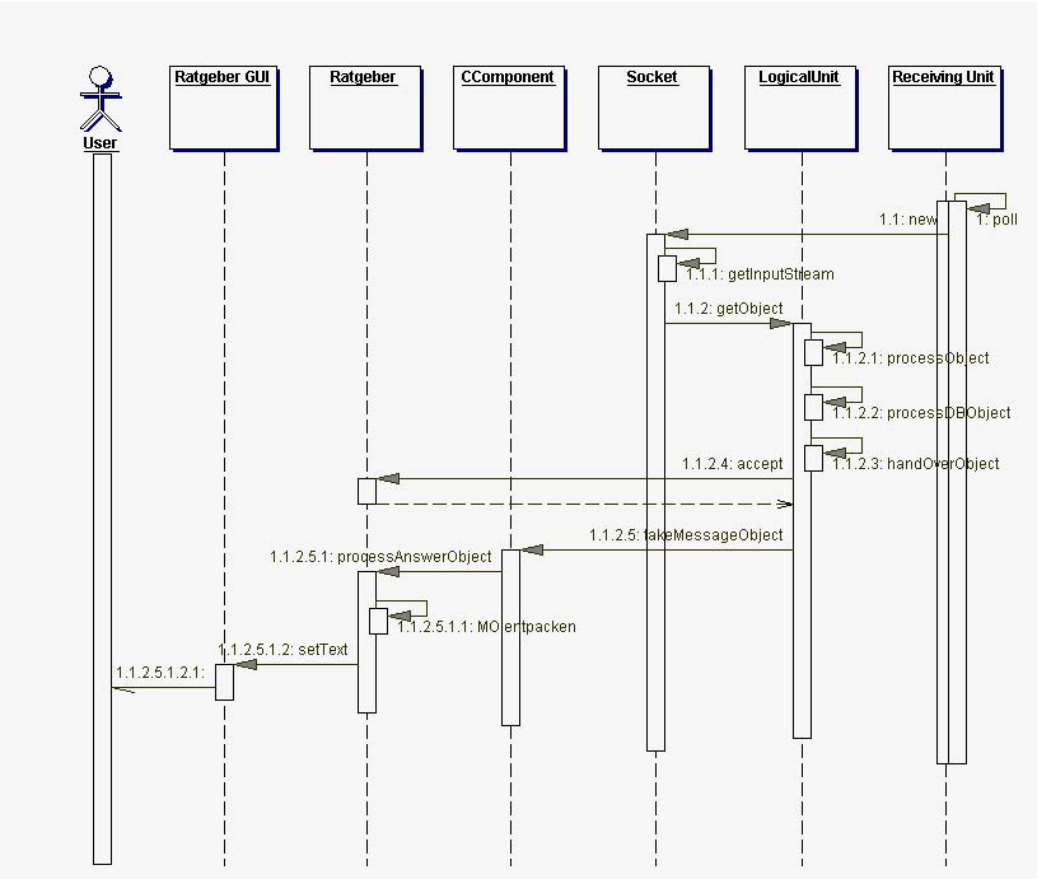

Fig. 8.10: Der Ratschlag wird von der Datenbank an den Client übermittelt

nente „RatgeberGUI“ übermittelt, die dem Spieler diesen Text anzeigt.

8.1.4 Der väterliche Freund

Der väterliche Freund unterscheidet sich vom Ratgeber dadurch, dass er vom Spieler nicht explizit aufgerufen werden kann. Der väterliche Freund ist ein Agent, d.h. er läuft selbständig im Hintergrund. Er besteht aus der Klasse VFreund und er besitzt kein eigenes GUI. Er interagiert mit dem Spieler nur mittels „Popup“Fenster, die die Ergebnisse seiner Auswertung darstellen.

Eigenschaften des väterlichen Freundes:

1. Der Spieler kann den väterlichen Freund nicht aus- oder einschalten.

2. Er kann nicht bestimmen, auf welche Art von Situationen er ansprechen soll.

3. Er kann den Zeitpunkt nicht bestimmen, wann er den Hinweis des väterlichen Freundes erhält.

In diesen Eigenschaften unterscheiden sich väterlicher Freund und Ratgeber.

Der väterliche Freund meldet sich tatsächlich nur dann, wenn er auf eine Situation aufmerksam geworden ist, die ihm zu einem Anzeigen einer Hinweises veranlasst. Wird keine solcher Situationen erkannt, bekommt der Spieler während des gesamten Spiels keine Meldung vom väterliche Freund. Auch in diesem Punkt unterscheiden sich Ratgeber und väterlicher Freund.

Der väterliche Freund bezieht ebenso wie der Ratgeber seine Daten aus der Datenstruktur der speziellen Hilfsmittelkomponente der Datenbank.

Bei jedem „Proceed“ werden alle speziellen Hilfsmittelkomponenten des väterlichen Freundes ausgewertet. Trifft eine oder mehrere Auswertungen zu, so wird in diesen speziellen Hilfsmittelkomponenten der anzuzeigende Text ausgelesen. Diese ausgelesenen Texte werden dem Spieler präsentiert. Es kann auch möglich sein, dass zu einem Zeitpunkt mehrere Auswertungen zutreffen. In diesem Fall werden mehrere Hinweise an den Spieler weitergeleitet. Auf Grund der gleichen Realisierung in der Datenbank, ist es auch beim väterlichen Freund möglich, die Hinweise des väterlichen Freundes in mehreren Sprachen anzuzeigen.

Meldungen, die bereits angezeigt wurden, werden in einem Vector mit dem Namen „vFreund“ in der Klasse „LogicalUnit“ gespeichert. Dies ist notwendig, damit bereits angezeigt Meldungen nicht noch einmal angezeigt werden.

Bei einem Ameisespiel ist es möglich, das Spiel vorzeitig zu beenden und zu einem späteren Zeitpunkt wieder aufzunehmen. Der Vector, der bereits alle gezeigten Hinweise des väterlichen Freundes gespeichert hat, ging verloren. Wenn Situationen, die dem väterlichen Freund bereits einmal veranlassten, einen Hinweis anzuzeigen, noch existieren, dann wird dieser Hinweis erneut angezeigt. Dieses wiederholte Anzeigen der Hinweise wird aber nicht als störend empfunden, da der Neustart in den meisten Fällen erst zu einem späteren Zeitpunkt stattfindet und der Spieler diesen Hinweis bereits vergessen hat.

Die Abbildung 8.11 zeigt beispielhaft eine Meldung des väterlichen Freundes. Diese Meldung erscheint auf der gerade aktive Anzeige. Wenn dieser Hinweis vom Spieler befolgt wird, wirkt es sich positiv auf seinen weiteren Spielverlauf aus.

Fig. 8.11: Hinweis des väterlichen Freundes

Wie bei den vorangegangenen Kapiteln, wird auch an dieser Stelle die Arbeitsweise des väterlichen Freunds mittels Sequenzdiagrammen dargestellt.

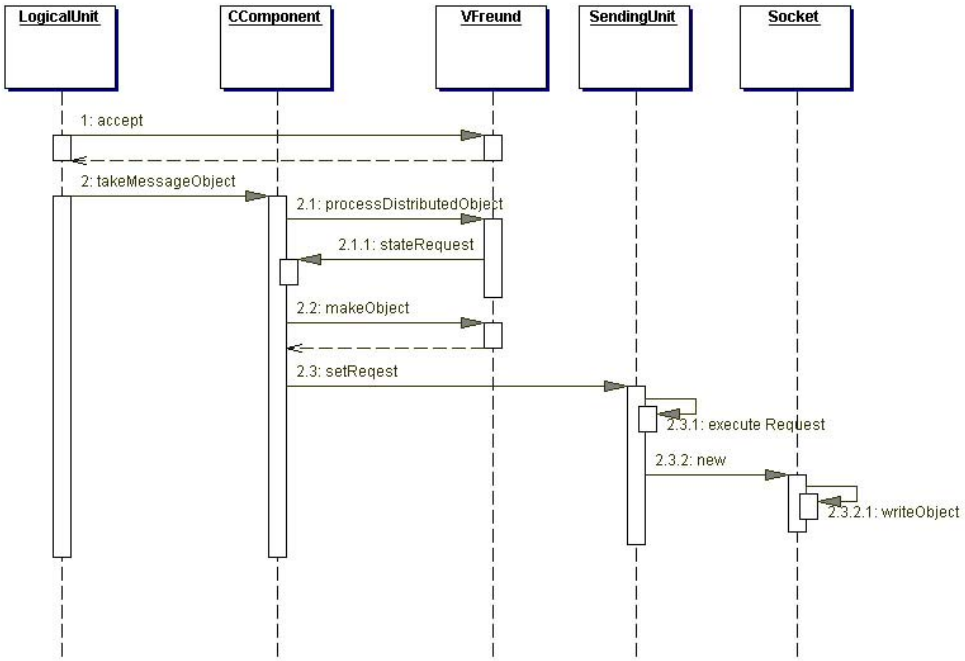

Fig. 8.12: Senden einer Abfrage des väterlichen Freundes

Die Abbildung 8.12 zeigt das Senden einer Anfrage des väterlichen Freundes an den LBM. Die Übermittlung der Anfrage wird nicht durch den Spieler angestoßen, wie es beim Ratgeber der Fall war. Auch die Art des Hinweises wird beim Senden der Anfrage nicht spezifiziert. Der eigentliche Ablauf der Übermittlung ist, aus den bereits gezeigten Beispielen, schon bestens bekannt.

Nachdem die Daten an den LBM übertragen wurden, erfolgt dort deren Verarbeitung. Alle väterlichen Freund Einträge der Datenbank werden

mit den aktuellen Spieldaten verglichen. Bei einer Übereinstimmung eines väterlichen Freund Eintrags mit einem aktuellen Spieldatum wird der entsprechende, bereits in der Datenbank befindliche, Hinweistext an den Client übermittelt. Wird keine Übereinstimmung gefunden, so wird das Message-Objekt ohne Hinweistext an den väterlichen Freund übertragen.

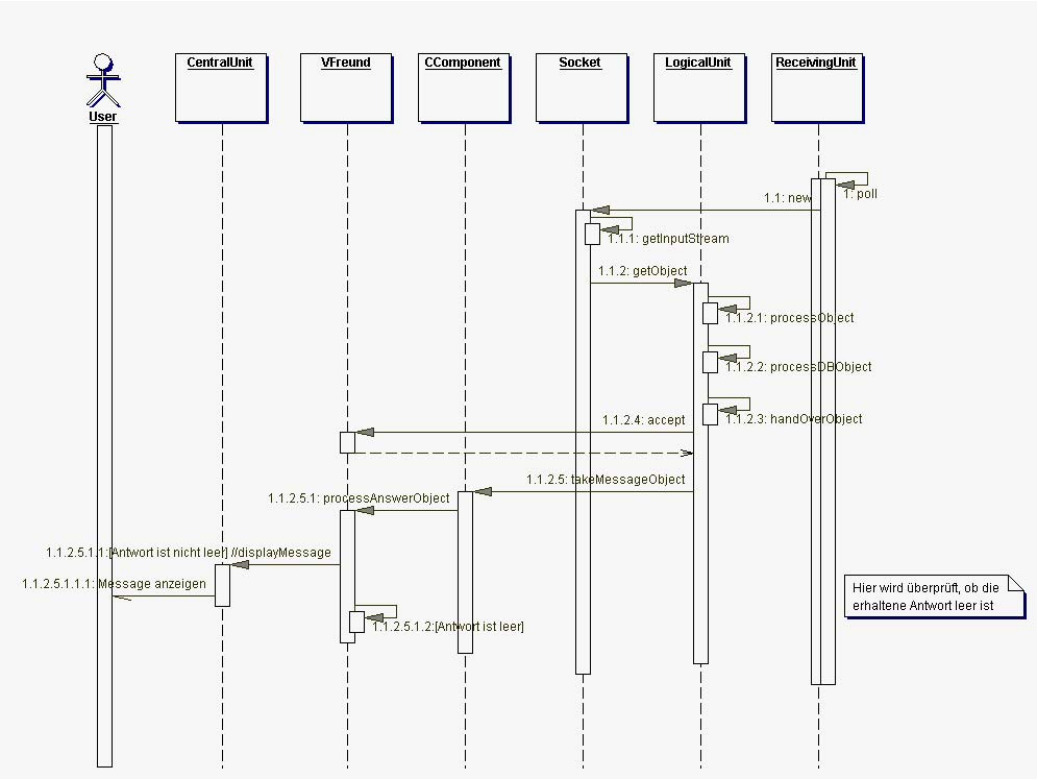

Fig. 8.13: Empfangen einer Antwort für den väterlichen Freund

Die Abbildung 8.13 zeigt das Empfangen einer väterlichen Freund Nachricht. Nach der Verarbeitung in der „LogicalUnit“ wird das MessageObjekt der „VFreund“ Klasse übergeben. In dieser Klasse wird ermittelt, ob das erhaltene MessageObjekt einen Text für den väterlichen Freund beinhaltet. Ist dies der Fall, wird dieser Text der „CentralUnit“ übergeben. Die in der „CentralUnit“ befindliche Methode „displayMessage“ zeigt den Hinweis des väterlichen Freundes auf dem gerade aktuellen Benutzerinterface an.

8.2 Relevante Literatur

[Yin], [Prosise 1999], [Althammer und Pree] [Krasner und Pope 1988] [R.Eckstein 1998]

Kapitel 9

ZUSAMMENFASSUNG

Diese Arbeit befasst sich mit dem Ameise Projekt und dessen Erweiterungen. Mit Hilfe des Ameisesystems wird dem Studierenden die Möglichkeit geboten, grössere Softwareprojekte zu leiten. Dabei soll das theoretisch erlernte Wissen in die Praxis umgesetzt werden. Fehler, die beim Führen des Projektes entstehen, haben durch diese „Simulator“ nicht so gravierende Folgen, wie in der Praxis. Dort kosten begangene Fehler eine Menge an Geld und führen oft zum Scheitern des Softwareprojektes. Zukünftige Projektleiter sollen die Probleme des Softwareprojektmanagements kennen, um darauf entsprechend reagieren zu können.

Um den Spieler während eines Spieles zu unterstützen, wurden im Ameisesystem Werkzeuge integriert. Diese Werkzeuge sind der väterliche Freund, der Ratgeber, das Auswertungswerkzeug und der Markierungsagent. Mit ihnen kann der Spieler durch schwierige Situationen geführt werden und es bedarf einer geringeren Betreuung durch den Spielbetreuer. Um die Ergebnisse der Spiele durch die Werkzeuge nicht zu sehr zu verfälschen, besteht die Möglichkeit sie je nach Bedarf ein- oder auszuschalten.

9.1 Erweiterungsmöglichkeiten und Ausblick

Da das Ameisesystem sehr umfangreich ist und alle Bereiche des Softwareprojektmanagements abdeckt, sind den Erweiterungen keine Schranken gesetzt. Um zukünftige Erweiterungen zu ermöglichen, wurden alle Werkzeuge im Ameisesystem generisch gehalten. Durch diese Vorgehensweise kann z.B. die darunterliegende Wissensbasis (die in Form des QS Modells vorliegt) ausgetauscht werden und die Werkzeuge können durch oberflächliche Änderungen an die neue Wissensbasis angepasst werden. Diese oberflächen Änderungen müssen vor allem in der Datenbank gemacht werden, da diese die modellspezifischen Daten beinhaltet, mit deren Hilfe die Werkzeuge arbeiten können.

Da die textuelle Eingabe und das damit verbunden Kennen der exakten Syntax den meisten Spielern bei den diversen Spielen nicht zusagte, wurde ein „AddOn“ zum aktuellen Client entwickelt. Diese Entwicklung wurde von der Fachhochschule für Telematik in Klagenfurt durchgeführt und besitzt den Namen „Client Plus“. Mit Hilfe dieser Erweiterung ist es möglich, Befehle aus einer erstellten Liste von Befehlen auszuwählen. Die exakte Syntax der Befehle ist dabei nicht mehr von Bedeutung. Diese Version des Clients existiert bereits und wird bald in das bestehende System eingebunden. Diese Erweiterung bezieht sich nur auf die Eingabe der Benutzerkommandos. Die Architektur des Clients bleibt von der Veränderung ausgenommen.

Da die „Client Plus“- Erweiterung einen Zusatz zum bestehenden System darstellt, ist es möglich, dass die alte textuelle Eingabe ebenfalls noch benutzt werden kann. Da jedoch die Syntax der Befehle sehr umständlich ist und der Aufbau der Befehle nicht konsistent gehalten wurde, wird es kaum noch Spieler geben, die diese verwenden.

Anhang A

BESCHREIBUNG DER BENUTZEROBERFLÄCHE

In diesem Kapitel wird die Benutzeroberfläche des Clients erklärt.

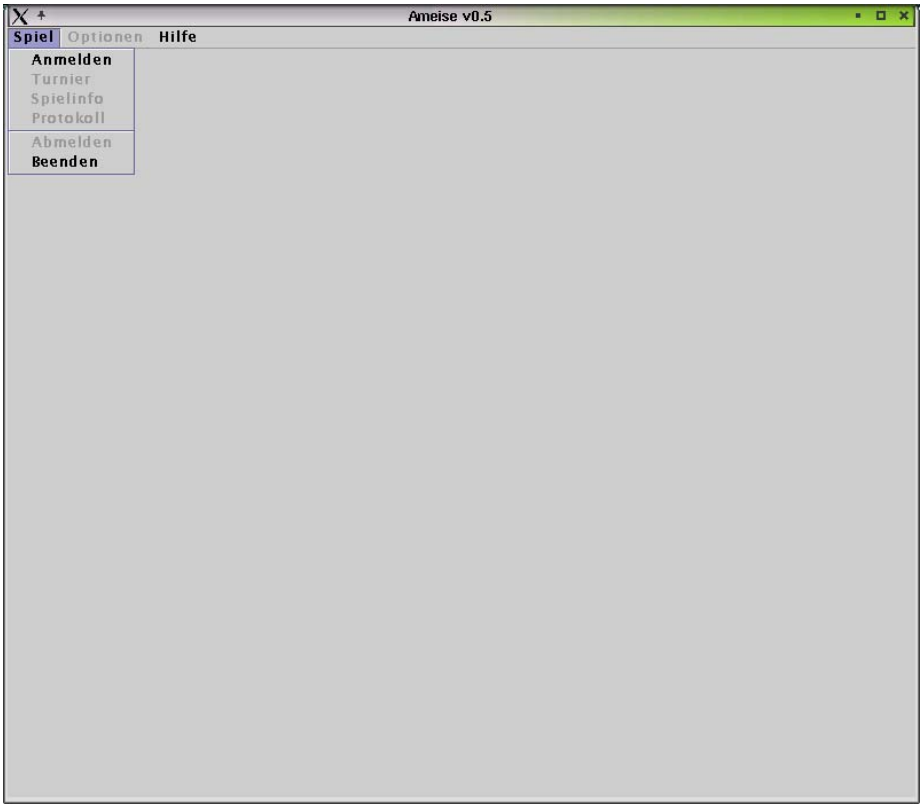

Fig. A.1: Einstiegsbildschirm

Wenn der Spieler den Client aktiviert, wird ihm der Bildschirm aus Abbildung A.1 gezeigt. Um den Client vollständig zu aktivieren, muss der Spieler sich anmelden. Um zum Anmeldefenster gelangen zu können, muss der Spieler aus der bereits sichtbaren Menüleiste den Eintrag „Anmelden“ auswählen.

Dieser Eintrag findet sich in dem Menüpunkt Spiel. Nach dem der Eintrag ausgewählt wurde, erscheint das Anmeldefenster.

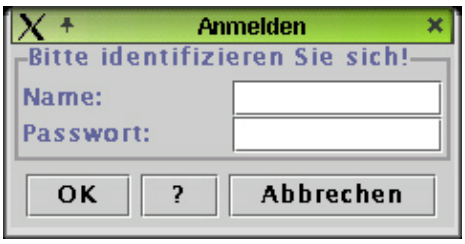

Fig. A.2: Anmeldefenster

Das Anmeldefenster ist in Abbildung A.2 ersichtlich. In ihm wird der Spieler aufgefordert, dessen Spielername und das Spielerpasswort einzugeben. Der Spielername und das Spielerpasswort muss dem Spieler vor Spielbeginn vom Ameise- Betreuer gegeben werden. Nach der Eingabe des Passwortes muss der „OK“ Button gedrückt werden, um die Überprüfung auf der Datenbank zu starten. Der „?“ Button erklärt dem Spieler die Notwendigkeit der Anmeldung und der „Abbrechen“ Button beendet den Anmeldebildschirm.

Nach der erfolgreichen Anmeldung wird dem Spieler der Simulationsbildschirm gezeigt. Siehe Abbildung A.3. Die Statuszeile ist vertikal in zwei Teile unterteilt. Der rechte Teil gibt an, welcher Spieler gerade am Client angemeldet ist, die linke Seite gibt den Systemzustand des Clients an. Da der Spieler noch keine Spielauswahl getroffen hat, zeigt der Eintrag an, dass der Client auf die Spielauswahl wartet. Dieses Fehlen der Spielauswahl führt dazu, dass der Spieler noch nicht ermächtigt ist, Befehle abzusetzen. Dies zeichnet sich durch das ausgegraute Eingabefeld aus, dass in Abbildung A.3 ersichtlich ist.

Um die Spielauswahl durchzuführen, muss der Spieler im Menu Spiel den Eintrag Turnier auswählen. Nun besitzt der Spieler die Möglichkeit den Eintrag Turnier auszuwählen. Dies resultiert daraus, dass der Spieler dem System bereits bekannt ist. Nach dem Drücken der Turnierauswahl erscheint das Turnierauswahlfeld.

Die Abbildung A.5 zeigt das Turnierauswahlfenster. Dieses Fenster beinhaltet beim Öffnen nur die Wurzel des Auswahlbaumes. Um ein Spiel auszuwählen, muss der Spieler den Baum öffnen. Wurde das Spiel ausgewählt und der Button Spiel fortsetzen gedrückt, so wird das gewählte Spiel in den Wrapper geladen, damit der Spieler spielen kann. Diese Schritt muss sowohl bei einem Neustart, als auch beim Beginn eines neuen Spieles durchgeführt

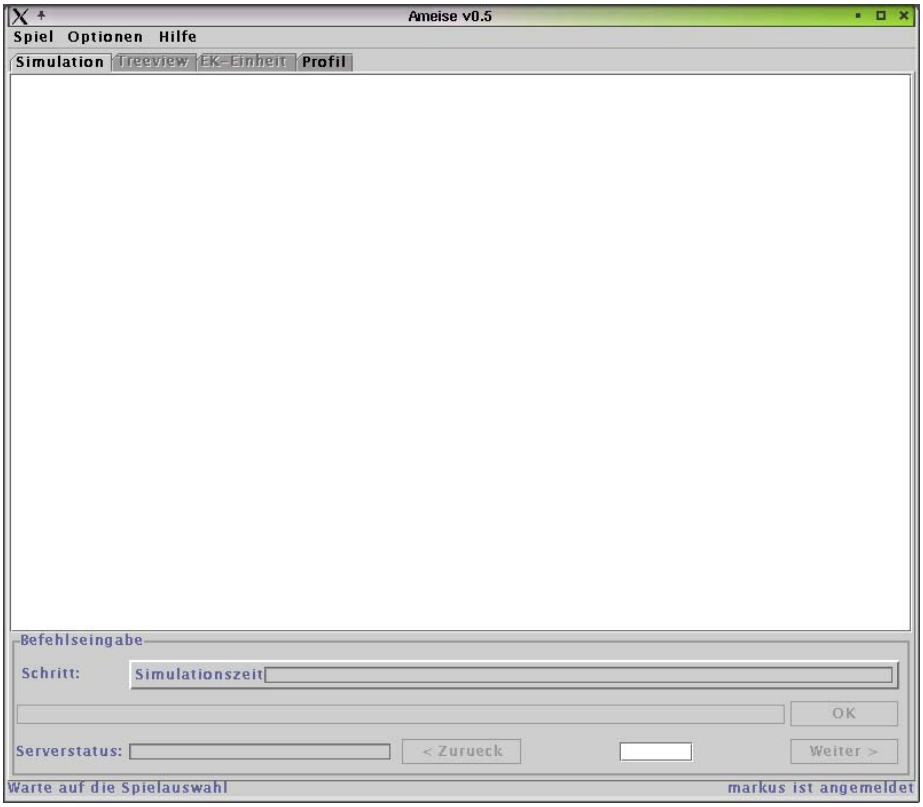

Fig. A.3: Simulationsbildschirm (vor der Spielauswahl)

werden.

Nach dem der Spieler auch diese Hürde überwunden hat, kann er mit dem Spiel beginnen.

Die Abbildung A.6 zeigt die Simulationsoberfläche, die auf Benutzereingaben wartet. Der linke Eintrag der Statuszeile drückt diese Bereitschaft der Simulation aus. Obwohl der Client für das Spielen bereits ist, muss der Wrapper zuerst initialisiert werden. Dieses erste Initialisieren wird durch das einmalige Drücken des „OK“ Buttons (rechts neben dem Eingabefeld) ermöglicht. Jede Betätigung dieses Buttons führt dazu, ein „Proceed“ (also ein Fortschalten des Simulators, um eine Tag) durchzuführen. Nach dem ersten „Proceed“ wird dem Spieler die Aufgabenstellung präsentiert. Diese Aufgabenstellung wurde vom Kunden erstellt und beinhaltet dessen Anforderungen. Mit diesen Anforderungen werden die erzielten Resultate nach Fertigstellung des Projektes verglichen. Ebenso wird das Spiel des Spielers anhand dieser Anforderungen bewertet.

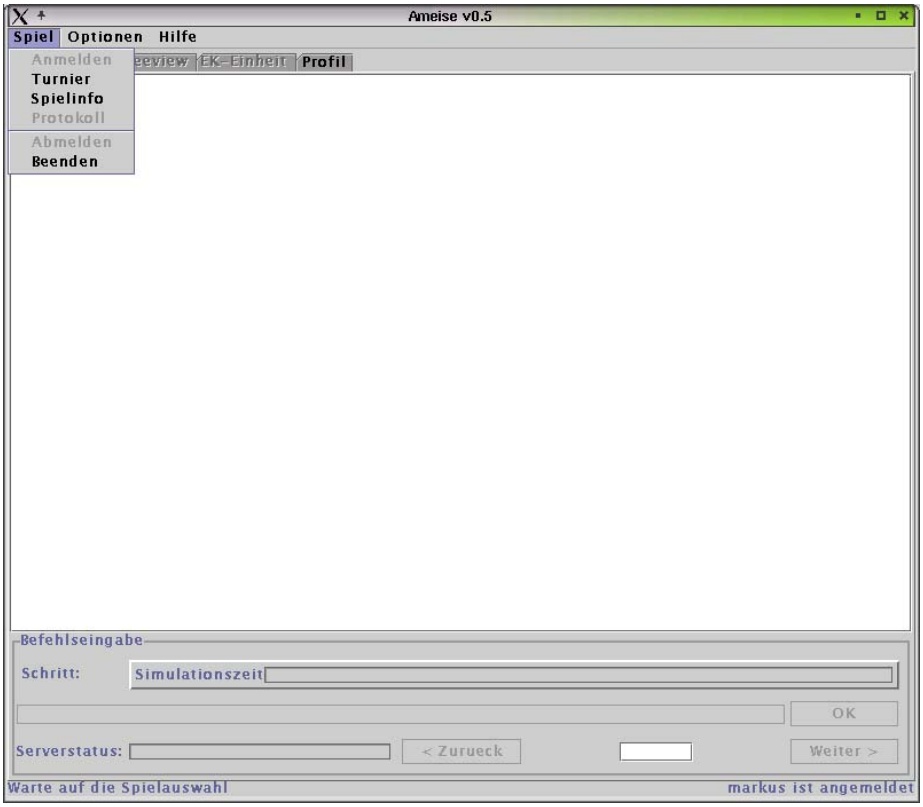

Fig. A.4: Auswahl des Turniers

Nachdem die Anforderungen des Kunden vom Spieler aufmerksam durchgelesen wurden, kann der Spieler beginnen Befehle an das System abzusetzen. Da diese Version des Clients nur die textuelle Eingabe von Befehlen verarbeiten kann, muss der Spieler diese Befehle kennen, um diese korrekt einzugeben. Die Befehlsliste muss dem Spieler vor Beginn eines Spiels übergeben werden.

Die Abbildung A.7 zeigt die Kundenanforderungen und zwei, bereits abgesetzte Befehle. Die Befehlseingabe erfolgt über das Textfeld des Clients. Nachdem ein Spieler einen Befehl eingetippt hat, muss er die „Enter“ Taste auf der Tastatur betätigen. Nur dadurch wird ein abgesetzter Befehl an den Wrapper geschickt. Wurde ein Befehl eingetippt und versehentlich der „OK“ Button gedrückt, so wird dieser eingetragene Befehl nicht ausgeführt und der Simulator wechselt zum nächsten Tag.

Bei den eingebenen Befehlen gibt es zwei Arten. Diese Befehlsarten unterscheiden sich dadurch, wann sie ein Feedback auf einen Benutzerbefehl liefern. Die in Abbildung A.7 eingetragenen Befehle liefern unmittelbar kein

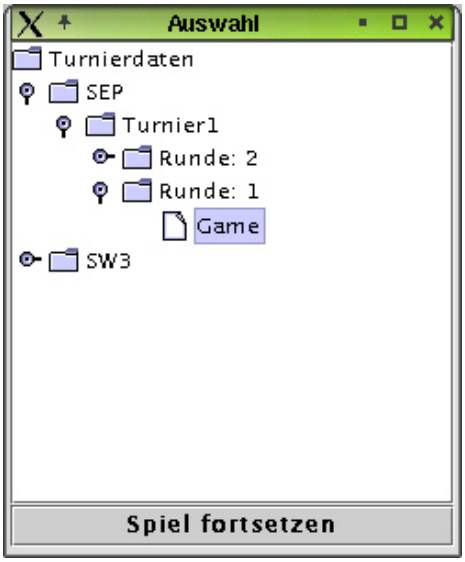

Fig. A.5: Turnierauswahlbaum

Feedback, wenn sie erfolgreich waren. Fehlerhafte Befehle werden dem Spieler durch eine Fehlermeldung der Simulation mitgeteilt. Auf manche Befehle wird unmittelbar nach deren Senden ein Feedback an den Spieler geschickt.

Bei den in den Abbildungen gezeigten Befehlen wurde auch immer gleich der Entwickler angegeben. Der Ameiseclient erlaubt es aber auch, nur den Befehl abzusetzten. In diesem Fall befragt der Simulator den Spieler, welchen Entwickler er für eine Tätigkeit auswählen will. Diese Rückfragen an den Spieler werden durch „Callbacks“ realisiert.

Die Abbildung A.8 zeigt eine Rückfrage des Systems. In diesem Fall wurde nur der Befehl „hire“ abgesetzt. Das System fragt den Spieler, welchen Mitarbeiter er einstellen will. Durch Auswahl des Mitarbeiter und betätigen des „OK“ Buttons, wird diese Auswahl übernommen. Wenn diese Rückfrage des Systems durch einen Fehler bei der Eingabe entstanden ist, hat der Spieler die Möglichkeit, diese Frage mit einem „Cancel“ zu beantworten. Wenn „Cancel“ übergeben wird, wird dieser Befehl nicht vom Simulator behandelt.

Die Abbildung A.9 zeigt einen abgesetzten Befehl auf den direkt ein Feedback vom System erteilt wird. Durch diesen Befehl wird der Mitarbeiter nach seinen Eigenschaften befragt. Weiter ist in der Abbildung A.9 zu erkennen, dass zwei Mitarbeiter zu der Spezifikation eingeteilt wurden und anschließend wurde der „OK“ Button gedrückt. Durch das Drücken des „OK“ Buttons

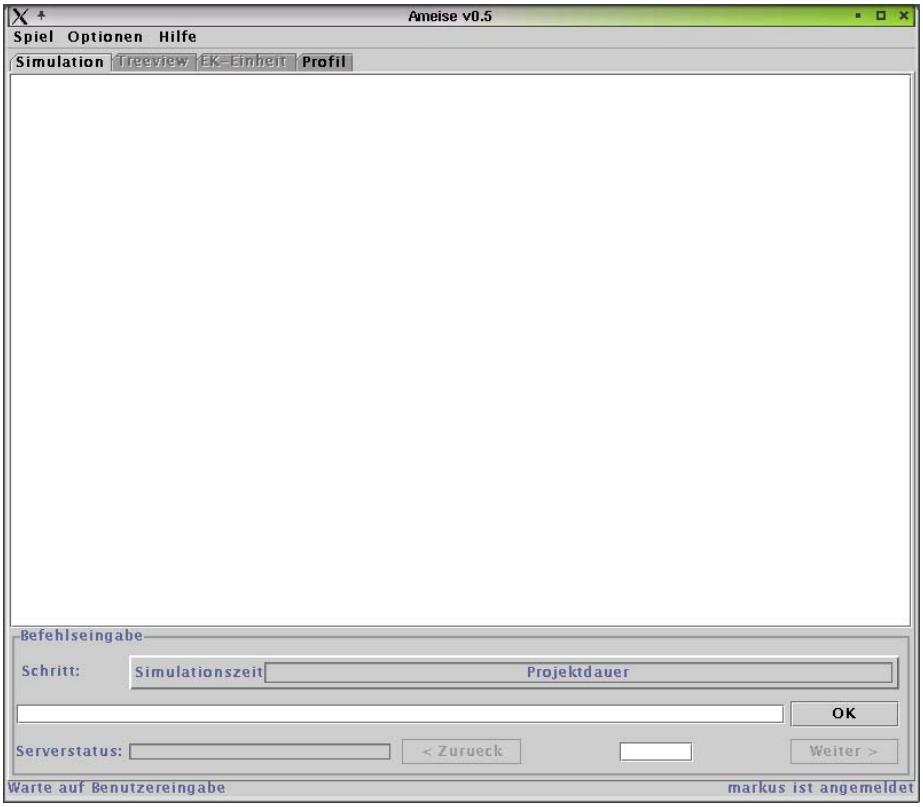

Fig. A.6: Simulationsbildschirm, Bereit für die Eingabe

wird das Spiel einen Tag weitergeschalten und alle Befehle die kein direktes Feedback gegeben haben, antworten jetzt.

Der Fortschritt im Spiel wird auch farblich durch den Simulationsbalken repräsentiert. Im Ameisesimulator besteht auch die Möglichkeit mehr als ein „Proceed“ auf einmal abzusetzen. Bei der Verwendung dieser Möglichkeit ist Vorsicht geboten. Durch das Fortschalten mehrerer Tage kann es vorkommen, dass ein Spieler mehrere Tage keiner Beschäftigung nachgeht. Die Anzahl der fortzuschreitenden Tage ist mit 25 begrenzt.

Wird ein Spieler mit einer Tätigkeit fertig, so teilt er dies dem Spieler mit. Der Spieler besitzt weiters die Möglichkeit, Informationen über die bereits verstrichen Projekttage vom System zu erfragen.

Wie die Syntax der Befehle aussieht wird, hier nicht beschrieben.

Hat der Spieler alle Phasen zu seiner Zufriedenheit beendet, kann er dem Kunden das entwickelte System übergeben. Diese Übergabe erfolgt durch den Befehl „deliver system“. Wird dieser Befehl abgesetzt, so wird dem Kunden die Möglichkeit entzogen, weiterzuspielen. Erhält der Wrapper diesen Befehl,

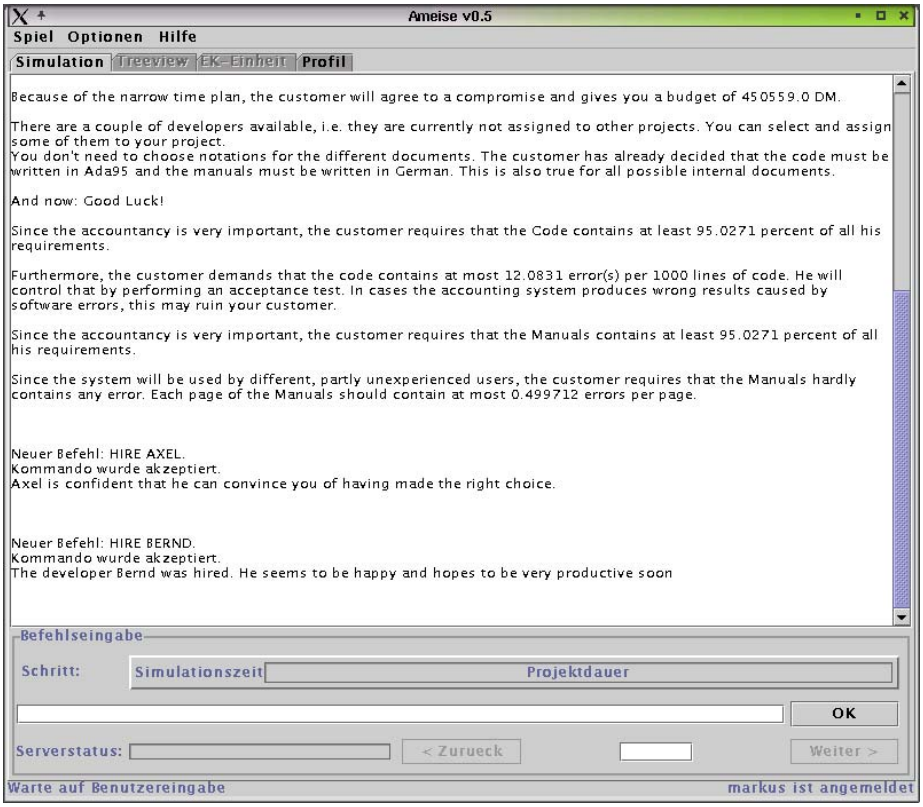

Fig. A.7: Spielbeginn

wird das Spiel des Spielers ausgewertet und der Spieler erhält vom Kunden ein Feedback über das gelieferte Produkt.

Fig. A.8: Callback

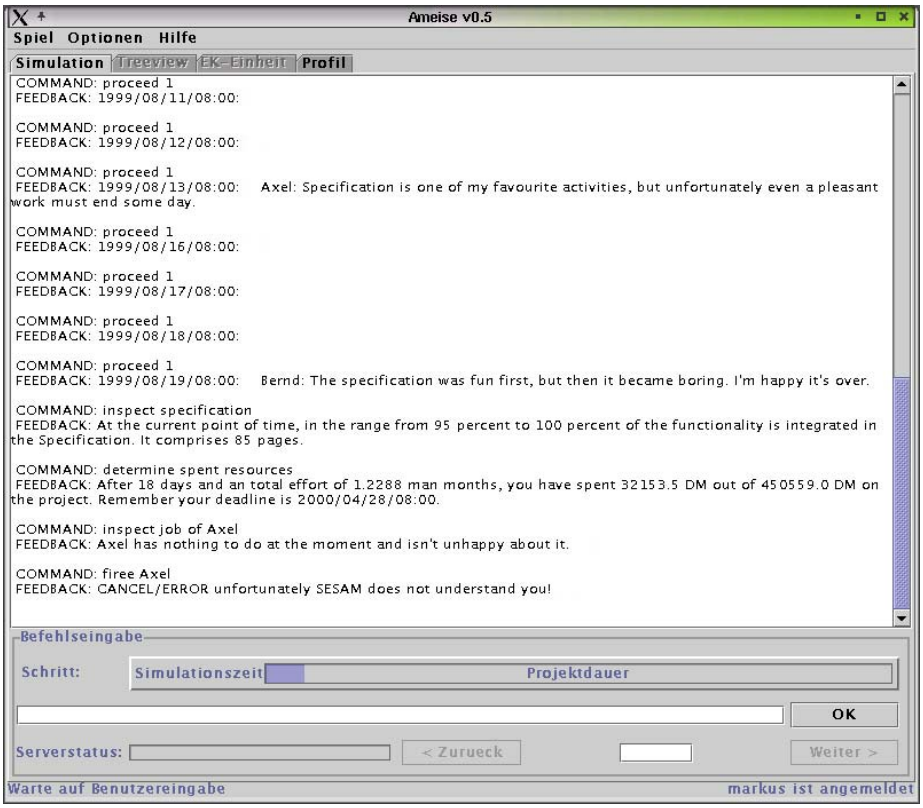

Fig. A.9: Fortschalten im Spiel

Anhang B

TABELLENVERZEICHNIS

Anhang C

ABBILDUNGSVERZEICHNIS

Anhang D

GLOSSAR

Anhang E

INDEX

LITERATURVERZEICHNIS

[Althammer und Pree] Althammer, E. und W. Pree. *Design and Implementation of a MVC-Based Architecture for E-Commerce Applications*.

[Anke Drappa 1999] Anke Drappa, Jochen Ludewig (1999). *Simulation in Software Engineering Training*.

[Cori 1985] Cori, Kent A. (1985). *Fundamentals of Master Scheduling for the Project Manager*.

[Drappa 1999] Drappa, Anke (1999). *Quanitative Modellierung von Softwareprojekten*.

[Drappa et al. 1995] Drappa, D., A. Deininger und M. Ludewig (1995). *Modeling and Simulation of Software Projects*.

[Hampp 2001] Hampp, Tilmann (2001). *Eine feingranulare SESAM-Variante*.

[Jaeger 2003] Jaeger, Susanne (2003). *Analyse von Projektverläufen*.

[Krasner und Pope 1988] Krasner, G.E. und S. Pope (1988). *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system*.

[Prosise 1999] Prosise, J. (1999). *Windows-Programmierung mit MFC*. Microsoft Press.

[R.Eckstein 1998] R.Eckstein, M.Loy, D.Wood (1998). *Java Swing*.

[Yin] Yin, Tong Sin. *The Smalltalk MVC paradigm with pluggable views*.