Integration eines kognitiven Modells in ein Simulationssystem zum Training der Luftverkehrsüberwachung

Diplomarbeit

Technische Universität Berlin
Institut für Angewandte Informatik
in Zusammenarbeit mit dem
Zentrum Mensch-Maschine-Systeme

1. Gutachter:
Prof. Dr.-Ing. E. Konrad

2. Gutachter:
Prof. Dr. phil. K. Eyferth

Betreuer:
Dr.-Ing. T. Jürgensohn

vorgelegt von
cand. inform. Sandro Leuchter

Berlin, 16. September 1999

Diese Diplomarbeit wurde durch ein Stipendium der Flughafen Frankfurt Main Stiftung unterstützt.

Diese Arbeit ist auch als PDF-Datei auf dem WWW-Server des Zentrums Mensch-Maschine-Systeme unter `http://www.zmms.tu-berlin.de/~sandro/doc/diplom/` verfügbar.

Zusammenfassung

In dieser Diplomarbeit wird ein neuartiges Trainingssystem für Fluglotsen entwickelt. Die Konzeption geschieht vor dem Hintergrund Intelligenter Tutorieller Systeme mit einer Simulation des Flugverkehrs. Ein psychologisch fundiertes Modell der kognitiven Strukturen und Vorgänge bei Fluglotsen wird dazu benutzt, die Aufmerksamkeit des Trainees auf relevante Objekte und Konstellationen in der Radardarstellung des simulierten Verkehrs zu lenken.

Diese Konzeption wurde in einem Prototypen implementiert. Der gewählte Design-Ansatz sieht vor, die beteiligten Komponenten in eigenständigen Prozessen umzusetzen, die verteilt in einem Rechnernetzwerk ablaufen können. Als Programmierwerkzeuge wurden dazu die interpretierten Skriptsprachen Perl und Tcl/Tk benutzt. Das kognitive Modell ist mit ACT-R in einem CommonLISP-Interpreter implementiert. Die Prozesse kommunizieren über Internet-Sockets und können durch den Einsatz von interpretierten Sprachen Metaprogrammierungsmechanismen benutzen, um sich gegenseitig dynamisch neu zu konfigurieren.

Inhaltsverzeichnis

1 Einleitung

Das Ziel in der Flugsicherung ist, den Luftverkehr sicher und ökonomisch zu steuern. Fluglotsen in der Streckenflugkontrolle haben eine anspruchsvolle Aufgabe. Sie beobachten den Flugverkehr über Radardarstellungen, antizipieren die Flugverläufe und schätzen mögliche Konflikte ab. Die Fluglotsen geben den Piloten Anweisungen, wie sie ihren Kurs zu ändern haben, damit Konflikte vermieden werden.

Fluglotsenanwärter werden nach strengen Kriterien ausgewählt und erhalten eine aufwendige Ausbildung. Auch Lotsen, die ihre Ausbildung bereits beendet haben, müssen Training und Prüfung absolvieren, um die Berechtigung zur Kontrolle eines neuen Bereiches zu erwerben.

Um adäquate Unterstützungssysteme für die Arbeit und Ausbildung der Lotsen entwickeln zu können, ist es wichtig, die kognitiven Strukturen und Prozesse der Fluglotsen zu kennen.

Ein seit 1993 von der Deutschen Forschungsgemeinschaft gefördertes Projekt an der Technischen Universität Berlin zur Modellierung von Fluglotsenleistungen entwickelte deshalb auf der Basis der Ergebnisse zahlreicher psychologischer Experimente mit Streckenfluglotsen ein Modell der kognitiven Leistungen, die für diese Tätigkeit erforderlich sind. Das Modell basiert auf der Annahme, symbolverarbeitende kognitive Vorgänge ließen sich analog zu Rechenvorgängen in Computern weitgehend adäquat abbilden.

Es entstand eine lauffähige Implementierung dieses Modells auf der Basis der kognitiven Architektur ACT-R. Diese Architektur ist ein verbreitetes System zur Modellierung mentaler Prozesse. In ACT-R wird menschliches Problemlösen als gesteuertes Abarbeiten von Regeln und die Repräsentation der jeweiligen Aufgabe durch ein Netzwerk von Wissensknoten abgebildet.

Die Zielstellung dieser Diplomarbeit ist es, dieses kognitive Fluglotsenmodell zu benutzen, um ein computergestütztes Trainingssystem zu entwickeln, das in der Ausbildung der Lotsenanwärter eingesetzt werden kann.

Beim vergleichbaren Teil des gegenwärtigen Ausbildungssystems werden aufwendige Installationen benutzt, um den Verkehr und die Arbeitssituation bei der Flugkontrolle zu simulieren. Ein erfahrener Ausbilder beobachtet die Leistung des Lotsenschülers bei der Simulation und gibt anschließend Hinweise, welche anderen Strategien nützlich gewesen wären.

Diese bisherige Situation soll mit dem neuen Trainingssystem verbessert werden. Das Ziel ist die Implementierung eines Systems, das Lotsenschüler außerhalb der Simulations-Infrastruktur der Ausbildung ohne Unterstützung durch einen Ausbilder einsetzen können. Der Vorteil liegt in einer besseren Ressourcennutzung für die Träger der Ausbildung, weil weniger Unterstützung durch Ausbilder und technische Systeme beim Training erforderlich ist, und einer Flexibilisierung für den Lotsenschüler, weil er selbst bestimmen kann, wann und wie lange er trainiert.

Das neue computergestützte Trainingssystem soll also auch teilweise die Funktionen des Ausbilders übernehmen. Das ist natürlich nur sehr eingeschränkt möglich. Deshalb muß als Teil des zu entwickelnden Konzeptes spezifiziert werden, in welchen Ausbildungsabschnitten und in

welchem organisatorischen Rahmen dieses System eingesetzt werden kann.

Diese Diplomarbeit soll das entstandene Trainingskonzept und -system dokumentieren. Sie ist in zwei Teile untergliedert.

Zuerst wird in Kapitel 2 die theoretische Grundlage des Trainingskonzeptes erörtert. Dafür wird näher auf den Rahmen der Intelligenten Tutoriellen Systeme, die kognitive Modellierung und das Fluglotsenmodell eingegangen. Existierende computergestützte Trainingssysteme im Flugsicherungsbereich und in anderen Domänen werden analysiert. Darauf aufbauend wird das neue Trainingskonzept präsentiert.

In Kapitel 3 dieser Arbeit wird ein Bericht über Anforderungsanalyse, Design und Implementierung des lauffähigen Trainingssystems gegeben. Dabei werden jeweils nacheinander die Entwicklungsideen und -konzepte für die Haupt-Subsysteme Luftverkehrssimulation, Radararbeitsplatz und Trainingskomponente im Rahmen des Trainingskonzeptes aus Kapitel 2 vorgestellt.

Am Ende schließt sich ein Ausblick an, in dem auf denkbare Erweiterungen dieses Trainingssystems eingegangen wird. Die Möglichkeiten eines Einsatzes des entwickelten Trainingskonzeptes in anderen Domänen werden kurz diskutiert.

Da diese Arbeit in einem interdisziplinären Feld angesiedelt ist, schließt sich ein Glossar mit einigen Begriffserläuterungen und Erklärungen der verwendeten Abkürzungen aus den unterschiedlichen Bereichen ab S. 75 an.

2 Entwurf des Trainingskonzeptes

In diesem Kapitel wird der Entwurf eines computergestützten Trainingskonzeptes für die Ausbildung von Fluglotsen in der Streckenflugkontrolle vorgestellt. Als allgemeiner Rahmen wird zuerst in Abschnitt 2.1 eine Klasse von computergestützten Trainingssystemen eingeführt, die mit Methoden der Künstlichen Intelligenz arbeiten. Dabei werden Lernsituationen mit und ohne Computerunterstützung verglichen.

Das zu entwickelnde Trainingssystem benutzt als Kern ein kognitives Modell. Um dieses Modell am Ende des Abschnittes 2.2 präsentieren zu können, wird zuerst die Aufgabe und das Arbeitsumfeld von Fluglotsen vorgestellt. Was kognitive Modelle sind und mit welchen Werkzeugen sie entwickelt werden können, wird danach erläutert.

In Abschnitt 2.3 werden existierende Trainingskonzepte vorgestellt: Zuerst wird die Ausbildungssituation für deutsche Fluglotsen beschrieben. Anschließend werden Aufbau und Funktionen computergestützter Trainingssysteme für die Fluglotsenausbildung und für andere Bereiche erläutert.

Am Ende dieses Kapitels werden in Abschnitt 2.4 die bis dahin vorgestellten Methoden und Prinzipien zur Entwicklung eines eigenen computergestützten Trainingskonzeptes angewandt.

2.1 Intelligente Tutorielle Systeme

Computer werden in der Ausbildung und im Training seit langer Zeit eingesetzt. Die verbreiteteste Form dieses *computer based trainings* (CBT) benutzt vorgefertigte Bildschirmmasken, auf denen Informationen präsentiert werden. Die Masken sind untereinander wie in einem Hypertext verknüpft. Dazu wird oft eine lineare Anordnung von aufeinander aufbauenden Informationen benutzt. Mit einem „weiter“- oder „zurück“-Button wird zur jeweils nächsten bzw. vorhergehenden Seite gesprungen. Am Ende einer aus mehreren Masken bestehenden Lektion werden Fragen zum Inhalt gestellt, die der Lerner beantworten muß. Bei Fehlern in der Beantwortung wird zu den entsprechenden Masken in der Lektion zurückgesprungen.

Je besser sich die Wissensvermittlung an den Lerner anpaßt, desto höher wird der Lernerfolg sein. Es ist deshalb wichtig, die Adaptionsfähigkeit von unterschiedlichen CBT-Systemen und anderen Lernumgebungen zu vergleichen. Dazu können zwei Arten von Adaption unterschieden werden: Die *Makroadaption* und die *Mikroadaption*.

Bei der *Makroadaption* wird die Fähigkeit betrachtet, die Rahmenbedingungen — die „äußere“ Lernsituation — an den Lernenden anzupassen. Das bedeutet, daß der Schüler z.B. den Zeitpunkt oder die Art des Unterrichts bestimmt. Auch die Einrichtung von Förderunterricht für einige Schüler wäre solch eine Anpassungsmaßnahme. Vergleicht man die Makroadaption von herkömmlichem Unterricht mit CBT-Anwendungen, zeigt sich, daß CBT-Systeme Vorteile haben. Der CBT-Lerner kann selbst bestimmen, wann und wie lange er trainiert. Er ist nicht an

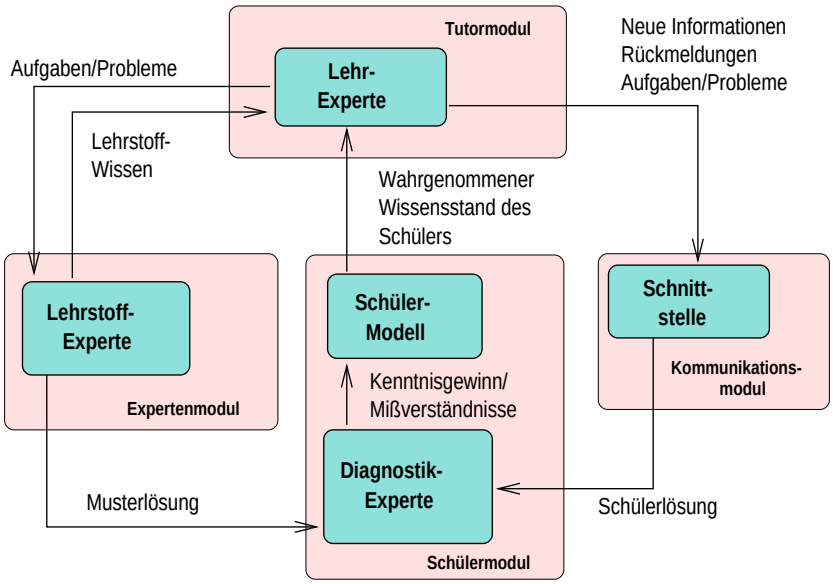

Abbildung 2.1: Architektur Intelligenter Tutorieller Systeme (entnommen aus [45])

einen bestimmten Ort der Wissensvermittlung gebunden.

Die *Mikroadaption* ist die Fähigkeit, während der Wissensvermittlung auf die Bedürfnisse des Lerners einzugehen. Beim herkömmlichen Unterricht würde das bedeuten, daß der Lehrer bemerkt, wenn ein Schüler etwas nicht verstanden hat, und die Lektion mit anderen Worten oder Lehrmitteln wiederholt. Das Maß der Mikroadaption im herkömmlichen Unterricht ist offensichtlich abhängig von der Größe der Klasse. CBT-Systeme haben im Vergleich zu einem menschlichen Lehrer nur sehr eingeschränkte Fähigkeiten, den Wissensstand des Schülers zu erkennen und die Art der Darstellung daran anzupassen.

Durch den Einsatz von Methoden aus der Künstlichen Intelligenz in CBT-Systemen versucht man diese Fähigkeiten zu verbessern. Im folgenden Abschnitt werden der Aufbau und die Methoden in den resultierenden „Intelligenten Tutoriellen Systemen“ (ITS) beschrieben.

Komponenten eines ITS

Um in CBT-Systemen die Diagnose des Kenntnisstandes des Lernenden zu verbessern und den Lehrstoff entsprechend aufzubereiten, werden mehrere Expertensysteme[1] in mehreren Modulen miteinander verknüpft (s. Abb. 2.1). Die Module eines ITS sind: Schülermodul, Tutormodul, Expertenmodul und Kommunikationsmodul. Die Module und die Methoden, die in ihnen verwendet werden, werden im folgenden beschrieben.

Schülermodul Das Schülermodul enthält zwei Komponenten: Das *Schüler-Modell* und einen *Diagnostik-Experten*. Das Schüler-Modell ist eine Repräsentation des Kenntnisstandes des Lernenden, die vom Diagnostik-Experten aufgebaut wird. Im Schüler-Modell werden Informationen darüber gespeichert, welche Leistungen ein Schüler erbracht hat. Zusätzlich werden

[1] Expertensysteme sind Software-Programme, die große Regelsammlungen enthalten, mit denen Wissen verarbeitet wird. Mit Inferenzmechanismen wird aus bekannten Fakten und Regeln neues Wissen abgeleitet und dem Benutzer präsentiert.

oft Fehler oder Mißverständnisse des Schülers repräsentiert.

Üblich ist, daß nur deklaratives Wissen repräsentiert wird. Dazu werden z.B. Objekt-Attribut-Wert-Tripel als Prolog-Fakten[2] oder Stereotype benutzt (s. [47]). Das Standard-Paradigma zur Strukturierung des repräsentierten Wissens ist das *overlay*-Modell [14]: In einer Menge von allen möglichen zu lernenden Wissenseinheiten („Topics“) wird vermerkt, welche vom Schüler bereits beherrscht werden. Eine Erweiterung des *overlay*-Modells ist, zusätzlich Fehler des Schülers, die außerhalb der möglichen Wissenseinheiten liegen, zu repräsentieren.

Ein anderer Mechanismus, um das Wissen des Schülers zu repräsentieren, sind *Bayes'sche Netzwerke* [14]. Hierfür wird ein Graph von Wissensknoten benutzt. Jeder Knoten im Netz repräsentiert eine Wissenseinheit, die der Schüler lernen soll. An jedem Knoten wird ein Sicherheitsmaß gespeichert, das ausdrückt, wie wahrscheinlich es ist, daß der Schüler das dazugehörende Konzept bereits beherrscht. Mit diesem Formalismus lasssen sich jedoch keine Fehler oder Mißkonzeptionen des Schülers repräsentieren.

Das Schüler-Modell wird vom Tutormodul benutzt (s.u.). Durch eine Repräsentation der aktuellen Leistungen des Lerners kann der Lehrstoff individuell präsentiert werden.

Der Diagnostik-Experte des Schülermoduls baut dieses Schüler-Modell auf. Dazu wird die Interaktion des Schülers mit dem ITS analysiert. Der Diagnostik-Experte benutzt dazu wissensbasierte Methoden, die z.B. aus den Fehlern des Schülers ein Modell der Abweichungen von einem ideellen fehlerlos arbeitenden Lerner aufbauen (*oppurtunistic strategy*, s. z.B. bei [47]).

Andere Systeme bauen mit dem Tutormodul einen Plan auf, wann welches Wissen zu vermitteln sein sollte (*plan based strategy*). Die geplante Abfolge der Einheiten muß jedoch vom Lerner nicht eingehalten werden. Diese Abweichungen vom geplanten Lernweg werden vom Diagnostik-Experten erkannt. Sie sind dann die Grundlage für das Schüler-Modell. Diese Strategie wird z.B. von dem WWW-basierten System ELM-ART benutzt (s. [21]).

Eine andere Methode benutzen Upmeyer und Günther zum Aufbau eines Schüler- bzw. Benutzermodells [68]. In ihrem CAD-System werden alle Interaktionen (Tastaturbedienung) eines Benutzers mit dem System registriert. Je nach Benutzungsgeschwindigkeiten unter unterschiedlichen Bedingungen werden Unsicherheiten oder Fehlkonzepte abgeleitet.

Tutormodul Das Tutormodul enthält ein Expertensystem, den Lehr-Experten, zur Steuerung des Prozesses der Wissensvermittlung. Es trifft z.B. die Entscheidung, wann welche neuen Informationen zu präsentieren sind, ob eine Wissenseinheit erneut darzustellen ist und welche Art der Wissensvermittlung am besten geeignet ist (z.B. abstrakt oder anhand eines Beispiels) [45].

Die Funktionen des Lehr-Experten sind nach Beck et al. ([14]):

1. *Auswahl eines Themas.*

   Damit ein geeignetes Thema ausgewählt werden kann, muß der Lehr-Experte auf den Lehrstoff-Experten (s.u.) und das Schüler-Modell zugreifen, um Informationen zu finden, die entweder wiederholt werden müssen oder die neu präsentiert werden sollen, um das Lernziel zu erreichen.

[2] Bei der Repräsentation mit Objekt-Atttibut-Wert-Tupeln werden 3-stellige Prädikate benutzt: oaw(*Objekt*, *Attribut*, *Wert*). *Objekt* ist eine interne Referenz auf eine Struktur, die mehrere Variablen, die *Attribute*, enthält. Jedes Attribut eines Objektes kann dann einen konkreten *Wert* annehmen. Durch eine Menge von solchen oaw-Fakten kann dann das gesamte deklarative Wissen repräsentiert werden. Der Vorteil dieses Repräsentationsformalismus' ist, daß interne Bezüge zwischen den Objekten dargestellt werden können.

Um diese Entscheidung treffen zu können, benutzt der Lehr-Experte eine „Meta-Strategie“ [14]. Eine Meta-Strategie ist ein Plan, auf welche Weise einem bestimmten Lerner Wissen am besten vermittelt werden kann. Mögliche Meta-Strategien sind z.B. der „Sokratische Dialog“ oder die Präsentation eines Beispiels. Die Meta-Strategie wird aufgrund von Informationen im Schüler-Modell ausgewählt. Im günstigsten Fall stehen mehrere zur Auswahl.

2. *Generierung eines Problems.*

   Nachdem der Lehr-Experte ein Thema ausgewählt hat und eine passende Meta-Strategie verfolgt, muß aus dem Thema ein konkretes Problem generiert werden und dem Schüler zur Lösung präsentiert werden.

   Diese Form der Wissensvermittlung wird häufig in ITS benutzt, weil die Bearbeitung des Problems beobachtet werden kann und sich daraus Schlüsse für den Aufbau des Schüler-Modells ziehen lassen.

   Zur Generierung eines Problems wird der Lehrstoff-Experte benutzt (s.u.). Das Problem wird dann an das Kommunikationsmodul zur Präsentation gegeben (s.u.).

3. *Erzeugung von Feedback.*

   Nachdem ein Problem vom Schüler bearbeitet worden ist, muß der Lehr-Experte eine geeignete Form des Feedbacks zur Lösung des Problems geben. Dazu greift der Lehr-Experte auf das Schüler-Modell zurück, in dem vermerkt ist, ob die Lösung fehlerhaft ist.

   Es kann auch passieren, daß der Schüler während der Bearbeitung des Problems eine „Hilfe-Funktion“ im ITS aktiviert.

   In beiden Fällen greift der Lehr-Experte auf den Lehrstoff-Experten zurück, um die Teilschritte abzurufen, die zur Lösung nötig sind (s.u.). Je nach gewählter „Meta-Strategie“ werden dann entsprechende Hinweise zur Lösung gegeben bzw. die korrekte Bearbeitung gelobt.

Das Ziel beim Entwurf der Wissensbasis des Lehr-Experten ist also, mit pädagogischem Wissen Teilfunktionen eines Lehrers zu ersetzen.

Expertenmodul Das Expertenmodul enthält den Lehrstoff-Experten. Dieses Software-Modul ist ein Expertensystem für den zu vermittelnden Wissensbereich. Es enthält eine Sammlung von Fakten und Regeln über die Domäne.

Die Fakten sollen vermittelt werden. Beim *overlay*-Konzept zur Repräsentation des Kenntnisstandes werden diese Fakten im Schüler-Modell repräsentiert. Regeln, die im Lehrstoff-Experten enthalten sind, verknüpfen die Fakten, so daß neues Wissen inferiert werden kann. Es ist möglich, daß auch das Regelwissen vermittelt werden soll. Es dient aber normalerweise dazu, neue Aufgaben aus vorgegebenen Klassen, die in den Regeln repräsentiert sind, zu erzeugen.

Der Lehr-Experte des Tutormoduls benutzt das Wissen des Lehrstoff-Experten, um Aufgaben für den Schüler zu erzeugen. Der Diagnostik-Experte des Schülermoduls benutzt den Lehrstoff-Experten, um die Lösungen des Schülers mit den Musterlösungen des Lehrstoff-Experten zu vergleichen und so Abweichungen festzustellen, die dann im Schüler-Modell repräsentiert werden.

Kommunikationsmodul Interaktionen mit dem Lerner werden über dieses Modul und die darin enthaltene Schnittstelle geregelt. Es ist für die Präsentation der Lektionen und Übungen zuständig. Dazu gehören z.B. die Dialogsteuerung und das Bildschirmlayout.

Der Lehrexperte des Tutor-Moduls sendet Beschreibungen anzuzeigender Aufgaben oder neuer Informationen an dieses Modul. Die Eingaben des Schülers werden an den Diagnostik-Experten des Schülermoduls zurückgemeldet.

Verknüpfung Von dieser konzeptuellen Aufteilung in Module wird in der Implementierung von ITS jedoch häufig abgewichen, damit der Wissensaustausch zwischen den beteiligten Komponenten vereinfacht wird [14]. Dadurch ist die Wiederverwendbarkeit der entstandenen Komponenten stark eingeschränkt. Deshalb gibt es Ansätze, diese Module bzw. Expertensysteme als eigenständige Agenten zu implementieren, die mit standardisierten Methoden (KQML, s. auch Abschnitt 3.2.1, S. 42) kommunizieren (s. z.B. bei [59] und [7]).

2.2 Modellierung der Fluglotsenleistungen

Für das Training in der Flugsicherungsdomäne ist ein konventioneller ITS-Ansatz nicht anwendbar. In den folgenden Abschnitten wird dies begründet und eine indirekte Methode zum Aufbau des Schüler-Modells entwickelt. Dafür wird ein Modell der kognitiven Vorgänge bei Fluglotsen benutzt.

Das kognitive Modell wird in den nächsten Abschnitten vorgestellt. Dazu wird zuerst die Aufgabe der Fluglotsen beschrieben. Danach werden unterschiedliche Methoden zur kognitiven Modellierung erörtert. Schließlich werden Struktur und Funktionen des implementierten Modells präsentiert.

2.2.1 Luftverkehrsüberwachungsaufgabe

Die Aufgabe der Flugsicherung ist, den Luftverkehr sicher und ökonomisch zu steuern. Dazu werden Flugpläne erarbeitet, die den zu erwartenden Verkehr regeln. Diese Planung ist global und langfristig. Der Flugverkehr in Europa ist im Moment noch größtenteils an Flugrouten, die sogenannten Luftstraßen gebunden. Sie entstehen durch die Verbindung von Navigationspunkten. Flugzeuge fliegen eine Liste von Navigationspunkten ab, wenn sie sich nach der Planung der Flugsicherung auf Luftstraßen bewegen.

Die Planung des Luftverkehrs kann jedoch nur selten exakt eingehalten werden, weil Flugzeiten oder -routen z.B. wegen des Wetters geändert werden müssen. Dadurch kann es passieren, daß ein Konflikt zwischen zwei Luftfahrzeugen entsteht: Würden sich die Flugzeuge weiter auf ihren Routen bewegen, würden sie sich einander zu weit annähern und möglicherweise kollidieren. Deshalb fliegen die Luftfahrzeuge unter der Kontrolle der Fluglotsen, die für eine kurzfristige lokale Planung verantwortlich sind. Die Kontrolle eines Lotsen erstreckt sich jeweils auf einen *Sektor*. Der Sektor ist ein Gebiet in dreidimensionaler Ausdehnung, das an mehrere andere Sektoren anschließt. Der gesamte kontrollierte Luftraum ist in Sektoren unterteilt. Piloten melden sich beim Eintritt in einen neuen Sektor beim nächsten Lotsen über Funk an.

Es werden je nach Höhe, aber auch horizontaler Ausdehnung unterschiedliche Typen von Sektoren unterschieden. So gibt es z.B. den Bereich *approach*, der für den Verkehr in der Nähe von Flughäfen zuständig ist. Im folgenden wird der *en-route* Bereich, die Streckenflugkontrolle, genauer erläutert, denn das System, das in dieser Diplomarbeit entwickelt wird, soll Lotsenschüler für die Kontrolle dieser Sektoren trainieren.

Im *en-route*-Bereich haben die Flugzeuge bereits ihre Reisegeschwindigkeit und -höhe erreicht. Der in Abb. 2.2 gezeigte en-route-Sektor West Radar 2 (WR2) hat eine vertikale Ausdehnung von etwa 5.000 bis 8.000 Metern über dem Meeresspiegel. Die horizontale Ausdehnung soll durch die eingezeichneten Städte verdeutlicht werden.

In jedem *en-route* Sektor kontrollieren zwei Fluglotsen den Verkehr. Die Arbeit wird nach dem „Planungshorizont“ zwischen ihnen aufgeteilt. Der *Planungslotse* arbeitet mit einem Zeithorizont von bis zu 20 Minuten. Er ist dafür verantwortlich, daß mögliche spätere Konflikte bereits vor dem Eintritt in den Sektor aufgelöst werden. Er kommuniziert dazu mit den Planungslotsen der Nachbarsektoren über eine Telefonverbindung. Als Informationsquelle benutzt er neben dem Radarbild vor allem Flugstreifen (s. Abb. 3.14, S. 66), auf denen ihm angekündigt wird, welche Luftfahrzeuge in seinem Sektor zu erwarten sind. Auf den Flugstreifen werden Informationen zur gewünschten Route, Flughöhe und -geschwindigkeit, sowie der Flugzeugtyp eingetragen. Sie werden nach den geplanten Zeitpunkten des Eintreffens an besonders relevanten Luftstraßenknoten im Sektor sortiert (s. Abb. 2.3).

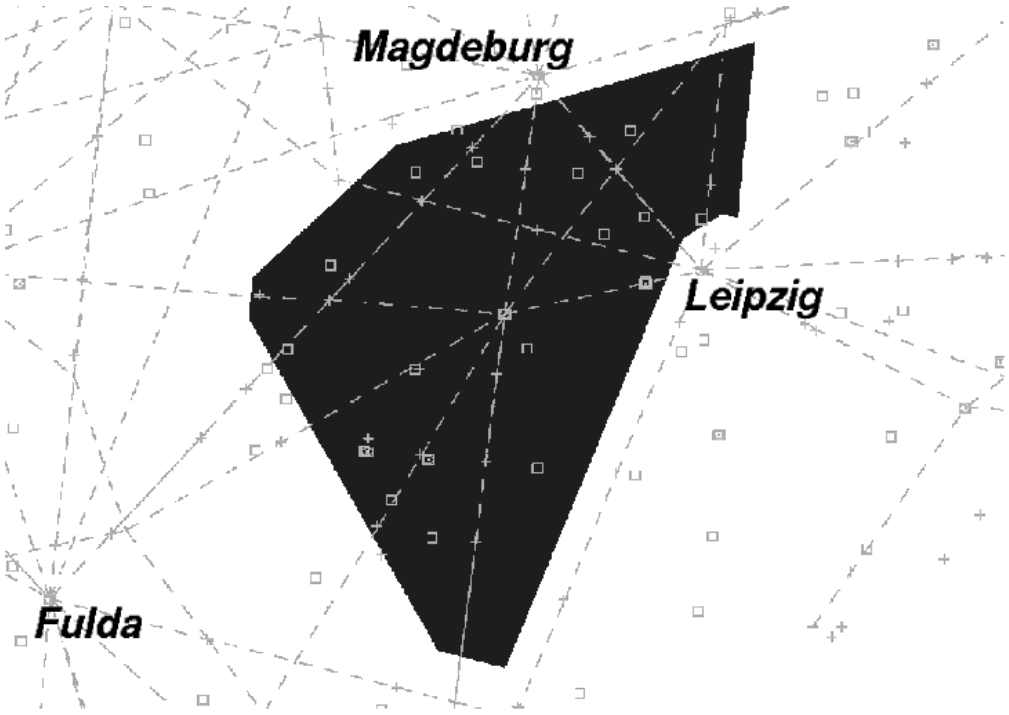

Abbildung 2.2: Radardarstellung des *en-route*-Sektor WR 2 (dunkler Bereich); die gestrichelten Linien sind Luftstraßen.

Abbildung 2.3: Fluglotsenarbeitsplätze bei der DFS. Im Vordergund sind drei Führungsschienen zu sehen, in denen Flugstreifen sortiert werden.

Der andere Fluglotse ist der *Radar-* oder *Exekutivlotse*. Seine Aufgabe ist es, kurzfristige Anweisungen an Flugzeuge in seinem Sektor zu geben und sie bei möglichen Konflikten umzuleiten. Er ist für die Sicherheit im Sektor verantwortlich. Als Informationsquelle benutzt er hauptsächlich das Radarbild, hat aber auch Zugriff auf die Flugstreifen. Er kommuniziert auf einer zu dem Sektor gehörenden Funkfrequenz mit den Piloten. Zur Lösung von Konflikten werden häufig Höhen- und Richtungsänderungen angewiesen.

2.2.2 Kognitive Modellierung

Die mentale Belastung von Fluglotsen in der Streckenflugkontrolle ist sehr hoch. Zur Verbesserung ihrer Arbeitssituation werden neue Unterstützungssysteme entwickelt. Um die dadurch entstehenden Änderungen der Arbeitssituation vor ihrer Implementierung abschätzen zu können, wurde ein Modell der kognitiven Leistungen von Fluglotsen entwickelt. Dieser Abschnitt erläutert, was kognitive Modellierung ist und welche unterschiedlichen Herangehensweisen es für die kognitive Modellierung gibt.

Kognitive Modellierung wird als Methode kognitionswissenschaftlicher Forschung, insbesondere in der Kognitiven Psychologie, benutzt, um die Theoriebildung zu unterstützen. Auf der Basis von Beobachtungen und der formalen Analyse der Aufgabe wird ein Modell erstellt. Es spiegelt die kognitiven Vorgänge wider, die bei der Bearbeitung ablaufen. Oft wird solch ein Modell dann mit einem Computerprogramm operationalisiert.

Bei der Formulierung und Implementierung kognitiver Modelle werden neben konnektionistischen Modellen oft Methoden der wissensbasierten Programmierung eingesetzt. Die wissensbasierte Programmierung ist ein der menschlichen Informationsverarbeitung analoges Modell. Bei dieser Art der Programmierung wird Regelwissen angenommen, das deklaratives Wissen modifiziert.

Die verwendeten Methoden dienen nicht nur als Hilfe zur Implementierung, sondern leiten auch den Prozeß der Formulierung des konzeptionellen Modells. Dabei lassen sich unterschiedliche Ausprägungen bei den verwendeten Methoden bzw. Implementierungsumgebungen unterscheiden.

Architekturen: Es existieren einige Theorien von Kognition, die den Anspruch haben, einen großen Bereich beobachteter Phänomene abdecken zu können. Das Ziel, „unified theories of cognition“ (UTC) zu formulieren, ist immer auch mit der Entwicklung einer Implementierungsumgebung auf der Basis von Produktionensystemen verbunden. Das Ziel von UTC ist, schnell zu validen Ergebnissen bei der Modellierung zu kommen, weil viele Phänomene, als Teil der Architektur vorgegeben, nicht implementiert werden müssen. Man erhofft sich eine bessere Verallgemeinerungsfähigkeit von Architekturen, da Grundannahmen aus dem Modell heraus in die Theorie verlagert werden [52].

Mikrotheorien: Modelle, die nicht innerhalb einer generellen Theorie der Kognition entwickelt werden, versuchen zumeist lokale Phänomene zu beschreiben. Dadurch lassen sich allgemeine Probleme existierender UTC [24] umgehen, ohne daß die resultierenden Modelle an Aussagekraft für ihr begrenztes Ziel verlieren. Sie werden oft mit allgemeinen Programmiersprachen wie Prolog implementiert (s. z.B. bei [57]).

Die Unterscheidung zwischen Mikrotheorie und Modell innerhalb einer vereinheitlichten Theorie läßt sich jedoch nicht ausschließlich am Gebrauch einer generellen Programmiersprache wie

Prolog, im Gegensatz zu einem spezialisierten Implementierungswerkzeug wie ACT-R, festmachen. Es gibt inzwischen zumindest eine Entwicklungsumgebung, COGENT (*cognitive objects within a graphical environment*; [25], [23]), deren Ziel es ist, die einfache Implementierung kognitiver Modelle außerhalb einer Architektur zu ermöglichen.

Die Vorteile einer lauffähigen Computersimulation eines kognitiven Modells sind nach [67]:

- Es entsteht ein detaillierteres Modell, da alle Annahmen explizit formalisiert werden müssen. Dabei werden häufig erst Unzulänglichkeiten erkannt.
- Der durch das Modell bestimmte Ablauf kognitiver Vorgänge kann von einem Computer simuliert werden. Dadurch können mehr und komplexere Szenarien erprobt werden, als mit einer „Handsimulation“.
- Ein lauffähiges Modell kann präzise empirisch überprüft werden.
- Die Simulation kann bei der psychologischen Hypothesenbildung helfen.

Zusätzlich ist es unter bestimmten Voraussetzungen möglich, ein implementiertes kognitives Modell direkt, d.h. außerhalb der psychologischen Theorienbildung, anzuwenden. Eine solche Anwendung stellt z.B. das in dieser Arbeit beschriebene Trainingssystem dar. Hier soll ein kognitives Modell zur Schülermodellierung eingesetzt werden.

Anforderungen an eine Umgebung zur Modellierung von Fluglotsenleistungen

Bass et al. [10] und Baxter & Ritter [13] zeigen, welche Eigenschaften kognitive Modelle haben müssen, um direkt in Simulationen einer Umwelt handeln zu können: Es ist notwendig, konzeptuell eine kontinuierliche Eingabe-/Reaktionsschleife in Aufgabensimulation und kognitivem Modell vorzusehen. Dafür besteht bei der Modellierung für die reine Theorienbildung oft kein Bedarf.

Neben dieser allgemeinen Anforderung lassen sich spezifische Bedingungen für die kognitive Modellierung von Fluglotsenleistungen aufstellen:

Für die Modellierung der kognitiven Leistungen von Fluglotsen wird angenommen [56], daß die Lotsen eine mentale Repräsentation der sich ständig verändernden Verkehrssituation aufbauen.

Dieses „picture“, wie sie die Lotsen selber nennen, muß als Basisrepräsentation die unterschiedlichen Verarbeitungserfordernisse steuern und ständig aktualisiert werden. Die Möglichkeit, eine solche Repräsentation aufzubauen, zu aktualisieren und mit ihr die unterschiedlichen Aufgabenerfordernisse zu sequenzieren, muß in einer Implementierungsumgebung vorhanden sein.

Das Modell muß mit der Aufgabenumgebung, also einer Luftraumsimulation interagieren können, um z.B. sich verändernde Daten in das simulierte „picture“ zu übernehmen oder einem Luftfahrzeug Anweisungen zu übermitteln.

Im folgenden Abschnitt werden unterschiedliche Architekturen zur kognitiven Modellierung vorgestellt. Es wird erörtert, ob diese Anforderungen erfüllt werden.

2.2.3 Architekturen zur kognitiven Modellierung[3]

Erprobte, umfangreiche Modellierungsumgebungen, die den Anspruch haben, Struktur- und Funktionseigenschaften des menschlichen kognitiven Systems auf der Basis von Produktionensytemen abzubilden, sind ATC-R (*adaptive control of thought — rational*; [4]), SOAR (*state, operator and result*; [40]; [52]), EPIC (*executive-process interactive control*; [49];[50]; [51]) und MIDAS (*man machine interactive design and analysis system*; [27]). Im folgenden werden diese vier kognitiven Architekturen mit den für uns relevanten Struktur- und Funktionseigenschaften kurz skizziert und unter folgenden Aspekten diskutiert:

- mentale Repräsentation der dynamischen Aufgabenumgebung,
- kontextabhängige Sequenzierung simultaner Aufgabenerfordernisse,
- Interaktion zwischen Modell und Aufgabensimulation.

Den genannten Umgebungen ist gemeinsam, daß sie auf Produktionensystemen basieren, in denen zwischen deklarativem Wissen (Fakten) und prozeduralem Wissen (Regeln) unterschieden wird. Jede Regel hat einen Bedingungsteil, der durch den Zustand des deklarativen Wissens erfüllt sein kann. Eine dieser passenden Regeln wird dann nach unterschiedlichen Schemata (Konfliktresolution) ausgewählt und angewandt, wobei entweder Aktionen in der Umgebung initiiert werden oder der Zustand des deklarativen Wissens geändert wird. Der Zustand der wahrgenommenen Aufgabenumgebung wird im deklarativen Wissen repräsentiert, die Verarbeitungsschemata werden in Form von Regeln modelliert.

ACT-R

In ACT-R [4] wurden bislang häufig Lösungen statischer Probleme modelliert.

Die grundlegenden Wissenseinheiten im deklarativen Gedächtnis heißen Chunks, die als Knoten in einem semantischen Netz andere Chunks referenzieren. Jeder Knoten des Netzes besitzt zu einem bestimmten Zeitpunkt einen bestimmten Aktivationsgrad, der seine aktuelle Relevanz für den Verarbeitungsprozeß widerspiegelt. Sie entsteht durch den Enkodierungsprozeß, die Ausführung von Produktionen und die Generierung von Zielknoten. Je aktivierter ein Chunk ist, desto schneller kann er abgerufen werden. Wenn mehrere Chunks mit dem Bedingungsteil einer Produktion übereinstimmen, wird der aktivierteste Chunk abgerufen. Zielgerichtetes Verhalten wird in ACT-R durch eine hierarchische Zielstruktur im deklarativen Wissen modelliert, die die externe Aufgabenstruktur reflektieren soll.

In ACT-R wird die Verarbeitungssequenz jeweils durch das oberste Ziel auf dem Zielstapel bestimmt. In einer dynamischen Aufgabenumgebung existiert aber keine statische Zielhierarchie. Indessen werden Sequenzierungsstrategien, die die Relevanz und die Dringlichkeit der mentalen Operationen berücksichtigen, benötigt.

In ACT-R ist es nicht vorgesehen, daß das System mit einem Benutzer oder mit einer simulierten aktiven Aufgabenumgebung kommuniziert. Die Entwicklung ACT-R/PM (ACT-R/Perceptual Motor; [5]) erweitert ACT-R aber in diese Richtung, indem perzeptuelle und motorische Komponenten mit einem ACT-R-Kern und einer Aufgabensimulation verknüpft werden.

[3] Der Abschnitt 2.2.3 ist aus [55] entnommen.

SOAR

In SOAR ([40]) wurden zahlreiche Problemlösungen modelliert. In SOAR wird Problemlösen als die schrittweise Transformation eines Ausgangszustandes (die Problemerfassung) in einen Zielzustand (die Problemlösung) aufgefaßt. Auch in SOAR wird zwischen deklarativem und prozeduralem Wissen, repräsentiert als Operatoren, unterschieden.

In dynamischen Situationen ist aber nicht unbedingt ein Zielzustand identifizierbar, so daß u.U. mehrere parallele Ziele verwaltet werden müssen [1]. In SOAR ist Metawissen als Konzept vorgesehen, mit dem sich Kontrollprozeduren als „comparison"-Produktionen formulieren lassen. Im Gegensatz zu ACT-R sieht SOAR Schnittstellen für eine Interaktion des Modells mit einer simulierten Aufgabenumgebung vor.

EPIC

Als Alternative zu den bisher beschriebenen kognitiven Architekturen, die vorwiegend an Hand statischer Problemlösungen entwickelt wurden, stellt EPIC ein Rahmenmodell dar, das die Modellierung multipler simultaner Aufgabenbewältigung und deren flexible Koordination vorsieht. Bisher wurde EPIC für die Modellierung der Bearbeitung von Doppelaufgaben [48], komplexer realistischer Probleme [42] und die Simulation von Mensch-Computer Interaktionen mit dem Ziel der Systemgestaltung angewandt (z.B. Systemgestaltung einer Telefonvermittlung; [43]). Diese Modelle konnten Zeiten und zum Teil auch Genauigkeiten der Aufgabenausführung vorhersagen.

EPIC verbindet einen kognitiven Prozessor mit motorischen und perzeptuellen Prozessoren, die parallel arbeiten und miteinander kommunizieren. Die perzeptuellen Prozessoren, die die visuelle, auditive und taktile Wahrnehmung emulieren, identifizieren Stimuli in der simulierten Aufgabenumgebung und tragen deren symbolische Repräsentation in das Arbeitsgedächtnis des kognitiven Prozessors ein.

Die Verarbeitung innerhalb des kognitiven Prozessors ist parallel. Mentale Kapazitätsbegrenzungen werden in EPIC in die Peripherie verlegt, d.h. die Informationsverarbeitung, die innerhalb des kognitiven Prozessors abläuft, ist durch die motorischen Prozessoren, die manuelle, artikulatorische und visuelle Aktionen ausführen, und durch das Update der perzeptuellen Prozessoren begrenzt.

Das prozedurale Gedächtnis umfaßt aufgabenspezifische Produktionen, die Kommandos an motorische Prozessoren geben oder Inhalte des Arbeitsgedächtnises ändern und exekutive Prozesse, die die Inhalte des Arbeitsgedächtnisses verwalten und Performanz in Abhängigkeit von den Aufgabeninstruktionen und den perzeptuellen und motorischen Begrenzungen koordinieren.

EPIC wurde eher für reaktive multiple Aufgaben konzipiert. Ein Aktivierungskonzept ist wegen der parallelen Verarbeitung im kognitiven Prozessor nicht vorgesehen. EPIC sieht für die Modellierung von Interaktion mit dynamischen Systemen geeignete Konstrukte, wie die exekutive Kontrolle über die Aktionen im Arbeitsgedächtnis, vor. Die Konzentration dieses Systems auf perzeptuelle und motorische Prozesse läßt es jedoch für vorwiegend kognitive Aufgaben kaum prädestiniert erscheinen.

MIDAS

Das Modellierungssystem MIDAS [26] wurde von der NASA zur Bewertung von Cockpit-Designs entwickelt. Jedes MIDAS-Modell besteht aus einer realistischen Simulation der Aufgabenumgebung und einem oder mehreren Operateur-Modellen. Das symbolische Operateur-

Modell besteht aus einem deklarativen und einem prozeduralen Teil und interagiert mit der Simulation der Aufgabenumgebung. Zur Interaktion hat das Operateur-Modell ein Perzeptionsmodul, das auf der Basis der Simulation von Blickbewegungen und auditiver Wahrnehmung Elemente der deklarativen Wissensbasis, der variablen Weltrepräsentation, ändert und ein Motormodul, das an Hand anatomischer Modelle Eingriffe in der simulierten Aufgabenumgebung vornimmt. Deklaratives Wissen ist in Form von Objekten mit Attributen repräsentiert, wobei auch eine Vergessensfunktion vorgesehen ist. Zwischen Objekten sind Relationen möglich, deren Gewichtung die semantische Nähe zwischen den Objekten repräsentiert. Prozedurales Wissen repräsentiert die Relationen von Ereignissen in der Umgebung zu Aktivitäten der Operateure.

Realisierbare Aktivitäten werden in Warteschlangen verwaltet, die von einem Steuerungsmodul (*scheduler*) gemäß einem „task load-model“ aktiviert werden. Der *scheduler* berücksichtigt Merkmale der Aktivitäten, wie deren Unterbrechbarkeit, Priorität oder Dauer. Die Verarbeitung in MIDAS ist jedoch, wie in EPIC, perzeptionsgesteuert. Durch das relativ feste Sequenzierungsschema läßt sich auch MIDAS besonders für reaktive Aufgaben einsetzen.

Alle dargestellten Architekturen weisen Vor- und Nachteile für die Implementierung von kognitiven Modellen in dynamischen Mensch-Maschine Systemen auf. Architekturen, die für die Modellierung des Umgangs mit reaktiven dynamischen Aufgaben ausgelegt sind, sind EPIC und MIDAS. Sie orientieren sich an reaktiven Aufgabenstellungen und bieten deshalb im Gegensatz zu ACT-R und SOAR wenig Möglichkeiten, Problemlösen und Lernen zu modellieren.

Eine mentale, veränderbare Repräsentation der simulierten Aufgabenumgebung als moderierende Instanz für eine kontextabhängige Steuerung kognitiver Verarbeitung ist in keinem dieser Modellierungssysteme ein zentrales Konstrukt.

Als Modellierungsarchitektur wurde ACT-R gewählt. Im folgenden Abschnitt wird das entwickelte kognitive Modell der Fluglotsenleistungen beschrieben.

2.2.4 Modell der kognitiven Vorgänge bei Fluglotsen in der Streckenflugkontrolle[4]

Das Modell MoFL (Modell der Fluglotsenleistungen) beinhaltet eine partielle und temporäre Repräsentation von Objekten und Ereignissen in dem kontrollierten Sektor. Die Repräsentation bildet die Grundlage für die Inferenz von Relationen zwischen Objekten vor dem Hintergrund der Konflikterkennung und Lösung sowie für die flexible Sequenzierung der kognitiven Verarbeitungschritte. Die wesentlichen Komponenten von MoFL sind fünf Module (Datenselektion, Antizipation, Konfliktresolution, Update und exekutive Kontrolle) und drei Informationsverarbeitungszyklen (Monitoring, Antizipation, Konfliktresolution), die diese Module miteinander verbinden. Die Module bauen das „picture“ auf und greifen bei der Verarbeitung auf dieses zurück (vgl. Abbildung 2.4). Ein weiteres Modul, das Sektorwissen, ist vorgesehen, aber bisher noch nicht weiter spezifiziert worden. Datenselektion und Antizipation beinhalten diagnostische Prozeduren. Prozeduren in dem Konfliktlösungsmodul bereiten die Eingriffe in die simulierte Aufgabenumgebung vor. Die kontextabhängige Sequenzierung der Verarbeitungsschritte unter dem Aspekt optimalen Zeitmanagements wird durch exekutive Kontrollprozeduren vorgenommen. Im folgenden werden die funktionalen Aspekte der Module durch die drei Informationsverarbeitungszyklen beschrieben.

[4] Der Abschnitt 2.2.4 ist aus [55] entnommen.

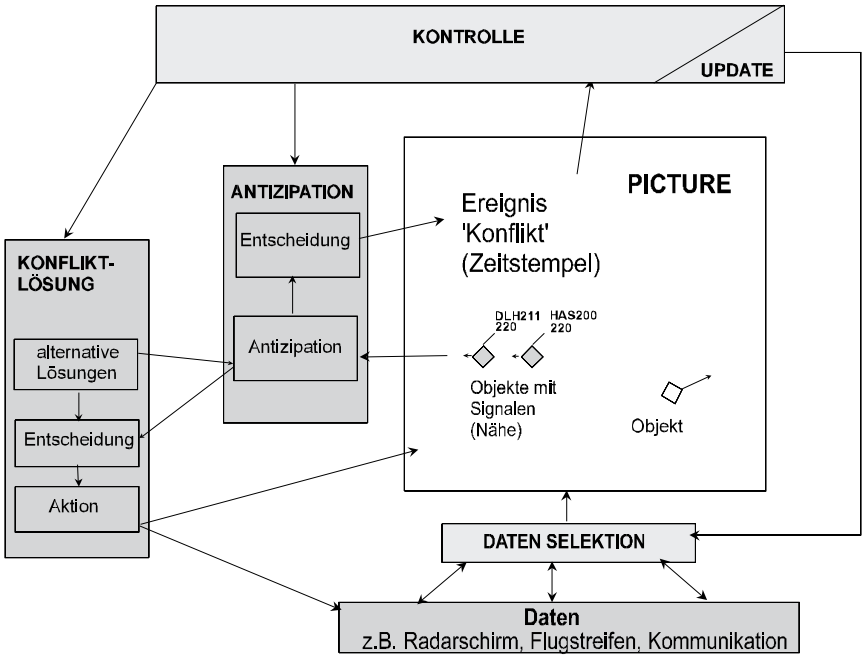

Abbildung 2.4: Die wesentlichen Komponenten des Modells der Fluglotsenleistungen (MoFL) (entnommen aus [55])

Die aktuelle Relevanz eines Objektes in der mentalen Repräsentation ist durch einen Aktivierungsparameter bestimmt. Je aktivierter ein Objekt ist, desto schneller können seine Merkmale abgerufen werden. Um die Sequenzierung zu modellieren, nehmen wir an, daß die Grade der Aktivierung nur relativ zum Status aller Objekte und Relationen, d.h. gemäß der Gesamtheit des „pictures“, gesetzt werden können. Dies macht deutlich, daß eine aktiv konstruierte mentale Repräsentation der Aufgabenumgebung Voraussetzung für das Setzen von Prioritäten ist.

Der Monitoringzyklus: Datenselektion und Update

Der Monitoringzyklus beinhaltet Datenselektionsprozesse und das kontinuierliche Update von Luftfahrzeugen. Ergebnisse von zwei Experimenten zur Datenselektion zeigten, daß Konstruktion und Aufrechterhaltung des „pictures“ auf einer sehr reduzierten Datenbasis stattfinden. Diese besteht hauptsächlich aus dem Callsign des Luftfahrzeuges, der Flughöhe sowie der jeweiligen Flugrichtung [17], [18]. Darüber hinaus ergaben Interviews, ein Experiment und vergleichbare Studien aus der Literatur, daß Lotsen während des Abscannens des Radarbildschirms nach spezifischen Merkmalen von Luftfahrzeugen und Sektorgegebenheiten suchen, die auf eine weitere Überwachung oder auf ein Konfliktpotential des Luftfahrzeuges hinweisen. Dieses sind Merkmale wie Vertikalbewegungen, die laterale Nähe zu einem anderen Luftfahrzeug oder sektorspezifische Merkmale, wie z. B. Kreuzungspunkte von Luftstraßen, an denen häufiger Konflikte auftreten (z.B. [17], [53], [6]). Aufgrund dieser Merkmale werden Flugzeuge mit Signalcharakter hoch aktiviert, d. h., daß sie aufmerksamkeits-fordernde Objekte sind. Auch neu zu integrierenden Luftfahrzeugen (einfliegende Luftfahrzeuge) wird zunächst ein hohes Aktivierungsniveau zugewiesen. Flugzeuge, die solche Merkmale nicht aufweisen, sind wenig aktiviert, d.h. es wird ihnen eine geringere Bedeutung in der aktuellen Situation beigemessen, und sie sind nur

mit den Informationen, die auf ihre Existenz verweisen (laterale Position, rudimentärer Richtungshinweis) repräsentiert. Die Dynamik der Situation erfordert ein kontinuierliches Update der Objekte. Dabei ist die Updaterate für Objekte mit einem hohen Aktivierungsniveau höher als für Objekte mit einem niedrigen Aktivierungsniveau.

Die Interaktion zwischen dem Computermodell und der simulierten Aufgabenumgebung wird als Netzwerkkommunikation zwischen zwei Prozessen abgebildet. Dabei werden zwei Wege der Kommunikation unterstützt:

- Asynchrone Kommunikation: Die Umgebung meldet MoFL das Auftreten definierter Ereignisse in der Aufgabenumgebung, wie z.B. eine von Piloten initiierte Sprechfunkkommunikation.
- Synchrone Kommunikation: MoFL fragt aktiv nach Informationen in der simulierten Aufgabenumgebung, wenn ein interner Bedarf an neuen Informationen über ein spezielles Luftfahrzeug in der Aufgabenumgebung identifiziert wird, oder bestimmte Luftfahrzeugdaten aktualisiert werden müssen.

Antizipation

Die Antizipation von Luftfahrzeugen ist der nächste Schritt in der Diagnose konfligierender Konstellationen. Antizipationsprozesse greifen auf hoch aktivierte Objekte zurück. Für jedes dieser Objekte wird der künftige Zustand antizipiert. Abhängig vom Resultat der Antizipation werden Flugzeuge mit Signalcharakter dann als Ereignis repräsentiert. Ein Ereignis ist durch den antizipierten Typ der Relation zwischen Luftfahrzeugen oder zwischen Luftfahrzeugen und Sektorstruktur definiert. Das Ergebnis einer Antizipation erlaubt die Entscheidung, ob die künftigen Trajektorien von Flugzeugen konfligieren, sicher separiert sind oder weiter überwacht werden müssen. Im Falle einer Konflikterkennung wird das Ereignis „Konflikt“ generiert, dem zusätzlich eine Abschätzung über die verbleibende Zeit zur Konfliktlösung (*timestamp*) beigefügt ist. Relationen, die als sicher eingeschätzt wurden, werden nun extrafokal. Ist das Resultat der Antizipation zu unscharf, da beispielsweise zwei Luftfahrzeuge noch weit voneinander entfernt sind, wird ein Überwachungsereignis generiert, daß darauf verweist, daß die Relation solange weiter überwacht und antizipiert werden muß, bis entschieden werden kann, ob ein Konflikt vorliegt oder die Flugzeuge sicher voneinander separiert sind. Die Klassifikation von Verkehrskonstellationen in „konflikthaft“, „sicher separiert“ und „weiter zu überwachen“ wurde durch Befunde aus einer Kategorisierung von Konstellationen durch Lotsen unterstüzt [54].

Der Antizipationszyklus ist in sequentiellen Produktionsregeln implementiert, die die folgenden vier Fragen testen (vgl. Abbildung 2.5):

Luftstraßenvergleich: Sind die Flugzeuge auf derselben Luftstraße oder auf kreuzenden Luftstraßen?

Höhenvergleich: Haben beide Flugzeuge die gleiche Flughöhe oder ist ein Flugzeug im Steig- oder Sinkflug?

Prognose: Führt die Fortschreibung der Trajektorien der Luftfahrzeuge zu einer Unterschreitung der Separationskriterien[5]? Diese Prädiktion wird mit Hilfe des Richtungsvektors (*velocity leader*) vorgenommen. Der Richtungsvektor ist eine Linie auf dem Radarschirm, welche die erwartete Bewegung eines Flugzeuges in einer kurzen Frist beschreibt. Es wird geprüft, ob es eine Unterschreitung der Separationskriterien geben wird.

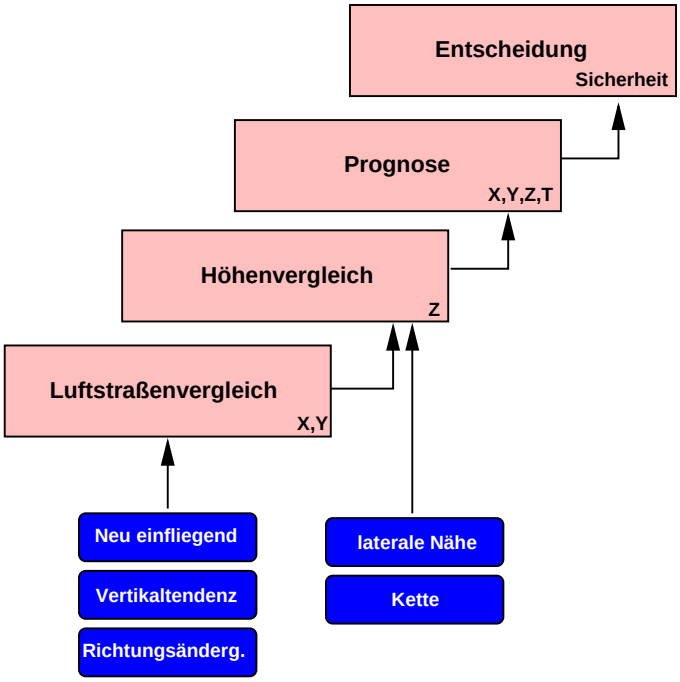

Abbildung 2.5: Die vier Antizipations-Stufen in MoFL (entnommen aus [55])

Entscheidung: Wie sicher war diese Fortschreibung? Sicherheit wird über die Zeit bis zur Unterschreitung der Separationskriterien gemessen. Zusätzlich wird der letztmögliche Zeitpunkt für eine Konfliktlösung berechnet.

Gemäß dieser Abfolge wird die Fokalität von Luftfahrzeug-Chunks modifiziert oder werden Ereignisse generiert.

Das „picture“

Das „picture“ wird als mentale Repräsentation von Objekten, Typen von Ereignissen und Objekten mit Relationen zu anderen Objekten und/oder der Sektorstruktur charakterisiert. Die Implementierung emuliert das „picture“ als die Gesamtheit der kognitiv verfügbaren Objekte zu einer gegebenen Zeit, ihrer Attribute und ihrer wahrgenommenen und inferierten Relationen in Zeit und Raum durch Chunks in einem semantischen Netz. Die Architektur ACT-R sieht nicht vor, eine bildhafte Repräsentation im Arbeitsgedächtnis zu implementieren. Einige der Objekte haben räumliche Positionen, die es ermöglichen, diese über die Position abzurufen. Bildhafte Operationen auf höherer Ebene, wie der Zugriff über die Entfernung zu anderen Luftraumobjekten, müssen emuliert werden. In Abbildung 2.6 ist die Klassenhierarchie der Chunks dargestellt. Jedes Luftraumobjekt hat eine Position auf dem Radarschirm. Abgeleitete Klassen sind Luftstraßen, Sektorgrenzen und Luftfahrzeuge, die zusätzliche Attribute haben, wie Rufzeichen, Geschwindigkeit und Höhe. Die Luftfahrzeug-Klasse wird dann weiter spezifiziert als einflie-

[5] Ein Konflikt zwischen zwei Luftfahrzeugen ist definiert als eine Unterschreitung der Spearationskriterien: Abhängig von der möglichen Ungenauigkeit des Radars, d.h. von der Anzahl von Radarstationen, die einen bestimmten Bereich erfassen, dürfen sich zwei Luftfahrzeuge nicht näher als 5.000 – 8.000 nautische Meilen horizontal und 1.000 ft vertikal annähern.

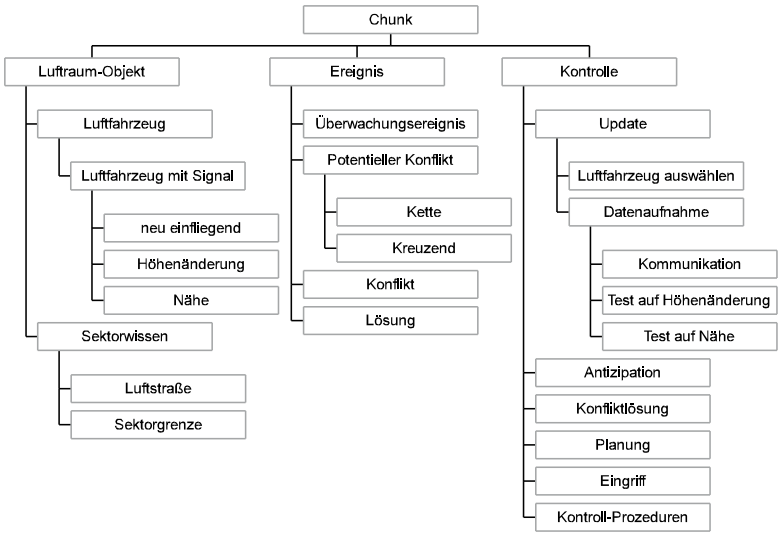

Abbildung 2.6: Vereinfachte Vererbungshierarchie der Chunk-Typen in MoFL (entnommen aus [55])

gende, die Höhe wechselnde und einem anderen Objekt nahe Luftfahrzeuge. Für jede dieser Klassen werden Instanzen als Chunks im Arbeitsgedächtnis durch Datenselektionsprozeduren während des Monitoringzyklus generiert oder modifiziert. Ereignisse repräsentieren inferiertes Wissen über Flugzeuge. Alle Ereignisse beziehen sich auf Luftraumobjekte durch spezifizierte Kanten im semantischen Netz. Instanzen werden durch Produktionsregeln im Antizipations- und Konfliktlösungsmodul generiert. Sie gehören zu den Ereignisunterklassen: Monitoring, Konflikt und Entscheidung. Konflikte können die Ausprägungen „Trajektorien kreuzend“ oder „hintereinander fliegend (Kette)“ haben. Konfliktereignisse haben ein zusätzliches Attribut, das die Referenz zum Konfliktpartner speichert.

Konfliktlösung

Wenn Konflikte entdeckt werden, initiiert der Problemlösungszyklus Prozesse, um einen drohenden Konflikt zu vermeiden. Der dringlichste Konflikt wird ausgewählt, um Lösungen aus dem Gedächtnis abzurufen (Alternativlösungen). Diese Lösungsvorschläge werden durch den Zugriff auf das Antizipationsmodul simuliert, um zu prüfen, ob Folgekonflikte auftreten würden. Die ausgewählte Lösung wird ausgeführt (Handlung).

Die Implementierung nutzt vordefinierte Standardlösungen, die auf bestimmte Konflikttypen passen. Um diese Standardlösungen nutzen zu können, wird die Klasse des Konfliktes durch Produktionsregeln bestimmt. Gemäß dieser Klassifikation werden einige Lösungen aus der Menge der Standardlösungen generiert. Die Produktionsregeln in dieser Simulation des Antizipationszyklus werden durch Ziele ausgelöst, die auf zu berücksichtigende Lösungen hinweisen. Wenn eine Lösung keine Folgekonflikte produziert, wird ein Lösungs-Chunk generiert. Eine Lösung besteht aus einer Sequenz von Handlungen, die durch das Modell ausgeführt werden. Die Zeit, die für die erste Intervention der Sequenz verbleibt, ist im Lösungs-Chunk vermerkt.

Kontrollprozeduren

Die Vielfalt der repräsentierten Objekte, Relationen und Eigenschaften innerhalb des „pictures“ erfordert, daß alle zeitkritischen Verarbeitungsschritte angemessen sequenziert werden. Die Koordination der oben beschriebenen Module (Datenselektion und Update, Antizipation, Konfliktlösung, Handlung) wird durch Kontrollprozeduren gesteuert. Für das Modell ist festgelegt worden, daß die Verarbeitungskomponenten in ihrem Funktionsablauf nicht unterbrochen werden können. Das Modell schaltet zwischen diesen hin und her, z.B. zwischen der Lösung eines Konfliktes und weiterem Monitoring (Update inklusive Datenselektion). Auf der Basis des Zustandes des „pictures“ wählen Kontrollprozeduren den wichtigsten und dringlichsten Verarbeitungsschritt.

In ACT-R postuliert Anderson eine hierarchische Zielstruktur, die die Aufgabenabhängigkeiten in der Umgebung repräsentiert. Um diese Hierarchie der Ziele zu modellieren, können mehrere Chunks auf den Zielstapel gelegt werden. Als Zielstapel wird eine spezielle Struktur innerhalb des Arbeitsspeichers bezeichnet. Die Verarbeitung wird vom augenblicklichen Ziel kontrolliert, also vom obersten Element auf dem Zielstapel. Das gegenwärtige Ziel verbreitet Aktivierung unter seinen Nachbarn im semantischen Netz. Das System fokussiert zu einem Zeitpunkt nur das oberste Ziel [4].

In der dynamischen Aufgabenumgebung der Flugsicherung existiert aber keine hierarchische Zielstruktur. Zu jedem Zeitpunkt sind mehere Aufgaben aktiv. Jede Aufgabe wird durch eines der Module Datenselektion, Antizipation oder Konfliktlösung bearbeitet. Das übergeordnete Modul „Kontrollprozeduren“ muß abhängig von der gegenwärtigen Struktur des „pictures“ ad hoc einen Prozeßfluß aufbauen. Der Prozeßfluß wird durch Metaproduktionen, die ein Modul mit einem Objekt oder Ereignis als Parameter auslösen, in einem Kontrollprozedurmodell bearbeitet. Um ein Modul auszulösen und es ununterbrechbar zu machen, wurde eine neue Klasse von Chunks eingeführt (s. Abb. 2.6). Diese Kontroll-Chunks sind die einzigen, die auf den Zielstapel gelangen.

Das Modell erfüllt mittels der Anwendung dieses Kontroll-Schemas die Anforderungen einer dynamischen Umgebung.

2.3 Trainingssysteme für die Kontrolle dynamischer Mensch-Maschine Systeme

Das beschriebene Modell der kognitiven Leistungen von Fluglotsen in der Streckenflugkontrolle wird in dem zu entwickelnden Trainingskonzept die Rolle eines allgemeinen Schüler-Modells nach dem ITS-Ansatz (s. Abschnitt 2.1) übernehmen.

Um dieses Konzept entwickeln zu können, werden zuerst allgemeine Anforderungen an das Training von Fluglotsen erläutert und die aktuelle Ausbildungssituation bei der Deutschen Flugsicherungs GmbH (DFS), die für die Flugsicherung in Deutschland zuständig ist, vorgestellt.

Danach werden andere computergestützte Trainingssysteme in ähnlichen Bereichen präsentiert.

2.3.1 Training von Fluglotsen

Wie in Abschnitt 2.2.1 deutlich wurde, ist die Arbeit der Fluglotsen in der Streckenflugkontrolle sehr anspruchsvoll. Es werden hohe Anforderungen an die Bewerber gestellt, die in aufwendigen Testverfahren ausgewählt werden [20]. Angenommene Bewerber durchlaufen eine etwa dreijährige Ausbildung.

Ausbildung bei der DFS

Das Ausbildungskonzept bei der DFS wurde vor kurzem umgestellt [46]. Das neue Trainingskonzept DATS (DFS *air traffic management training system*) sieht nun vier Ausbildungswege (FIS [6], FMP[7], FDB[8] und FVK[9]) vor. Das Ziel der Umstellung ist die Verbesserung der Flexibilität, damit Mitarbeiter leichter Stellen innerhalb der DFS wechseln können. Dabei soll eine stärkere Modularisierung der Ausbildung helfen. Daneben wird eine bessere Verzahnung von Theorie und Praxis, bzw. Lernen und Arbeiten, angestrebt.

Die Ausbildung gliedert sich in einen für alle Ausbildungswege identischen 16wöchigen Grundkurs an der Flugsicherungsakademie in Langen. Neben Grundwissen im Flugsicherungsbereich wird das Sprechfunkzeugnis erworben.

Danach teilt sich die Ausbildung in „ATM Specialist“ (FIS, FDB und FMP) und „ATM Controller“ (FVK). Die hier angebotenen Spezialisierungskurse verzahnen sowohl Theorie- als auch Praxisanteile und finden weiterhin an der Akademie statt (*institutional training*).

Nach erfolgreichem Abschluß dieser Module beginnt das *operational training* an einer der DFS-Niederlassungen, wo neben der Praxis-Einführung „on the Job“ weiterhin Theorie z.T. im Selbststudium vermittelt wird. Dazu werden als Methoden der Wissensvermittlung auch Simulation und CBT-Systeme eingesetzt.

Verantwortlich ist in diesem Ausbildungsabschnitt der Seniorwachleiter und ein Trainingsteam, zu dem der Wachleiter und zwei Coaches gehören. Die DFS hat detaillierte Richtlinien für die Arbeit der Trainer entwickelt. Zur Struktur dieses angeleiteten Lernens gehört eine tägliche Abschlußbesprechung (*debriefing*) zwischen Schüler und Trainer.

Auch das *operational training* ist in Trainingsabschnitte unterteilt, an deren Ende Leistungsbewertungen stattfinden. Der erste dieser Abschnitte dient der generellen Einweisung in den

[6] *flight information service*
[7] *flow management position*
[8] Flugdatenbearbeitung
[9] Flugverkehrskontrolle

Abbildung 2.7: Fluglotsenarbeitsplätze für die Simulation in Training und Entwicklung bei der DFS.

Tätigkeitsbereich, die folgenden Abschnitte haben den Erwerb von Berechtigungen zum Ziel. Fluglotsen brauchen sowohl eine allgemeine Qualifikation, als auch für jeden zu kontrollierenden Sektor eine eigene Berechtigung, die belegt, daß die Besonderheiten der räumlichen und inhaltlichen Struktur des Sektors und dessen Verkehr beherrscht werden. Mit dem erfolgreichen Abschluß des *operational trainings* ist die Ausbildung beendet.

Die DFS erhofft sich von dem modularen Aufbau des Trainingskonzeptes DATS nicht nur ein verbessertes Qualitätsmanagement in der Ausbildung von Anfängern, sondern auch Vorteile für die Fortbildung und Umschulung.

In Abschnitt 2.4 wird das eigene Trainingskonzept in diesen Ausbildungszyklus eingeordnet.

CBT-Anwendungen im Flugsicherungsbereich

In der Flugsicherungsausbildung gibt es drei Bereiche, in denen Computersysteme zur Unterstützung eingesetzt werden.

1. Während des *institutional trainings* bei dem DFS-Ausbildungskonzept DATS werden Simulationsszenarien zum Training benutzt. Mit Hilfe eines Computers werden Flugverläufe in einem vordefinierten Szenario simuliert. Dieser Verkehr wird den Lotsenarbeitsplätzen, in erster Linie Radaranzeigen, eingespielt, an denen Trainees den Verkehr beobachten (s. Abb. 2.7). Anweisungen werden über ein Interkom-System, das den Sprechfunk nachbildet, an einen oder mehrere Geisterpiloten gegeben, die über eigene Schnittstellen in den simulierten Verkehr eingreifen und so das Verhalten der angesprochenen Piloten der simulierten Flugzeuge nachbilden. Der Simulationsverlauf wird beobachtet, protokolliert und anschließend ausgewertet.

2. Die DFS und andere Flugsicherungsinstitutionen wie EUROCONTROL bieten inzwischen auch selbst entwickelte Lernprogramme für Personal-Computer an [28], [30]. Diese CBT-Software wird sowohl während des *institutional*, als auch *operational trainings* zum Selbststudium benutzt. Die von der DFS angebotenen CBT-Produkte haben den einheitlichen Aufbau Wissensvermittlung, Wissensanwendung/Training und Selbstkontrolle/Test. Es gibt Programme zu Themen wie Flugzeugtypen und -eigenschaften, Flugrouten und Prozeduren, Ortsbezeichner in Europa, Wetter, Eigenschaften und Umgang mit der Radardarstellung P1, Radar-Übungen und Identifikationsmethoden und Verhalten in ungewöhnlichen Notsituationen. EUROCONTROL bietet ähnliche Software an. Alle verwendeten CBT-Produkte basieren auf vorgefertigten Lektionen mit vorher geplanten Interaktionsmöglichkeiten und nicht auf einer kontinuierlichen Simulation des Luftverkehrs.

3. Daneben werden inzwischen von dritter Seite kostengünstige realitätsnahe Flugsicherungssimulationen für den PC angeboten. Dabei wird sowohl der Tower[10]-, als auch der Center-Bereich[11] abgedeckt. Sie sind jedoch natürlich nicht Teil der offiziellen Ausbildungslehrgänge und haben keine explizite Funktion zur Wissensvermittlung.

2.3.2 Computergestützte Trainingssysteme in anderen Bereichen

Entscheidender Faktor für die Effizienz und Effektivität von Lernsoftware ist deren Adaptivität an die Bedürfnisse des Lernenden [45]. Die Adaptivität der drei oben beschriebenen Formen der Computerunterstützung in der Fluglotsenausbildung ist unterschiedlich. Die Simulatorunterstützung während des *institutional* und *operational trainings* erfolgt unter der Anleitung eines Trainers. Beim *Debriefing* werden Probleme, die der Schüler bei der Übung hatte, erörtert. Im Rahmenwerk des ITS-Ansatzes erfolgt in diesem Fall die Umsetzung von Schüler-Diagnose, Tutoring und Lehrstoff innerhalb des menschlichen Experten, also beim Trainer. Da er ein sehr differentiertes Modell des Schülers hat, selber Experte in der Domäne ist und eine Fortbildung zum Training gemacht hat, ist die Effektivität dieser Lernumgebung sehr hoch. Der Nachteil ist, daß sowohl großer technischer Aufwand für die Simulation, als auch erhöhter Personalaufwand für Trainer und Geisterpiloten getrieben werden muß. Die Ausbildung erfolgt in einem realistischen Rahmen, ist dafür aber an die Akademie, bzw. die Niederlassungen gebunden.

Die CBT-Produkte von DFS und EUROCONTROL sind dagegen auf jedem PC ohne weitere Unterstützung zu einem beliebigen Zeitpunkt benutzbar. Es muß jedoch auf eine Simulation mit freier Interaktion zugunsten eines Lernpfades verzichtet werden.

Die dritte Gruppe von CBT-artigen Anwendungen sind die DFS-externen Simulationsprogramme. Sie sind auf dem PC benutzbar, haben aber aufgrund der vielfältigen Eingriffsmöglichkeiten in den Simulationsablauf keine expliziten Trainingsfunktionen.

In anderen Bereichen wird versucht, Systeme zum Training in komplexen Mensch-Maschine Systemen auf der Basis von ITS zu entwickeln. So gibt es z.B. schon ein kommerzielles System, das die Führung eines Unterseebootes in einem Simulationssystem mit eingebettetem ITS trainiert [39]. Der ITS-Ansatz verspricht durch die Schülermodellierung und der daraus resultierenden adaptiven Lernerführung eine erhöhte Effizienz und Effektivität. In den folgenden Abschnitten werden relevante Entwicklungen erörtert.

[10] z.B. TOWER von der Bruce Artwick Organization (inzwischen Microsoft) für Windows.

[11] z.B. RadarSim über `http://ourworld.compuserve.com/homepages/louisbreda/enwin95.htm` für Windows 95/98.

Kenny & Saito's Trainingssystem zur Kontrolle von Hubble

Kenney und Saito beschreiben eine Umgebung zu Training von Operateuren des Weltraumteleskops Hubble [41]. In ihrem System wird eine Simulation des Verhaltens des Satelliten eingesetzt, um mit unterschiedlichen Szenarien bestimmte Situationen zu trainieren. Dabei soll der Einsatz von Darstellungs- und Eingabemethoden aus der Virtual Reality-Entwicklung helfen, eine verbesserte Kontrolle über das Verhalten des Satelliten zu erlangen. Ihr System ist als eine nicht adaptive Mikrowelt zu klassifizieren. Kenney und Saito erörtern, daß die Integration von Methoden des ITS-Ansatzes für ihr Trainingssystem eine ideale Ergänzung darstellen würde.

Ansatz von Bass zum *situation awareness* Training

Endsley definiert *situation awareness* als Wahrnehmen der Elemente der Umgebung, das Verständnis für deren Bedeutung und die Fortschreibung deren Zustandes in die nahe Zukunft [29, S. 116]. Sie unterscheidet entsprechend drei aufeinander aufbauende Stufen von *situation awareness* als Wahrnehmung, Verständnis und Antizipation der relevanten Objekte der Aufgabenumgebung.

Das Aufrechthalten der *situation awareness* ist für die Kontrolleure eines dynamischen Mensch-Maschine Systems wie z.B. die Luftverkehrsüberwachung eine wichtige Fähigkeit, um die richtigen Eingriffe zu günstigen Zeitpunkten durchführen zu können. Um die *situation awareness* von Operateuren in unterschiedlichen Bereichen zu verbessern, wurden einige computergestützte Trainingssysteme entwickelt. Die Methoden, die in diesen Trainingssystemen benutzt werden, werden in den folgenden Abschnitten erläutert.

Context Based Tools Auf der Basis von OPM (*operational procedure model*), einer regelbasierten Beschreibung von Missionen, Prozeduren, Szenarien und Verhaltensmustern, entwickeln Sherry et al. ([66]) ein „Modell intentionalen Verhaltens" von der vertikalen Steuerung eines Hochgeschwindigkeits-Personen-Flugzeuges. Das Modell nimmt dabei den Zustand der Aufgabenumgebung (*context*) wahr (in diesem Fall 55 Werte von Zustandsvariablen des Szenarios) und transformiert sie in Steuerungsvariablen (in diesem Fall 4), die Eingriffen des Operateurs entsprechen. Sie benutzen dieses Modell, um auf dessen Basis die Darstellung der Ein- und Ausgabekanäle in einem Glass-Cockpit zu verbessern und damit die *situation awareness* der Crew zu verbessern. Das Modell wurde als Teil eines Tutor-Systems, das Piloten für das Lesen und Verstehen von Flug-Zustands-Anzeigegeräten trainiert, angewandt. Obwohl es sich um ein dynamisches Mensch-Maschine System handelt, wird im Tutor-System keine Simulation im Sinne einer Mikrowelt benutzt, sondern es werden vorgefertigte Lektionen präsentiert.

Chappell et al. ([22]) stellen ein anderes Trainingssystem zur *situation* bzw. *mode awareness*, im Glass Cockpit bei der vertikalen Steuerung vor. Ihr System ist im Gegensatz zu dem von Sherry et al. ausschließlich als Trainingssystem ausgelegt. Sie benutzen eine Simulationsumgebung, mit der die Trainees interagieren. Dabei werden aufgrund des Zustandes der Simulation (*context*) besondere Fehlersituationen, wie das Versäumen notwendiger Eingriffe, beim Trainee registriert. Bei solchen Fehlern wird die Simulation angehalten, und es wird eine passende Lehreinheit abgespielt. Das System benutzt also ein Modell des Zustandes des technischen Systems, um Eingaben (bzw. Eingabeunterlassungen) des Schülers zu interpretieren.

Bass et al. ([12]) diskutieren eine ähnliche Technik unter dem Namen „Context-Based State Monitoring" (CBSM) zum *situation awareness*-Training und zur Unterstützung, die *situation awareness* aufrechtzuerhalten. Sie schlagen dafür eine Repräsentation von Wissen über mögli-

che Fehlersituationen beim Erwerb und dem Aufrechterhalten von *situation awareness* vor. Diese Repräsentation, *„expectation network“* definiert einen Graphen, der drei Typen von Knoten enthalten kann. *Initiators* repräsentieren Bedingungen im Zustand des Systems, die den Beginn einer Aktivität repräsentieren. *Situation nodes* repräsentieren Schritte, bei dem Abarbeiten der Aktivität angewandt werden, bzw. die zum Erreichen des aktuellen Ziels notwendig sind. Jeder dieser Knoten hat eine Beschreibung von Erwartungen an den registrierten Zustand des Systems und Feedback-Informationen, die dem Lerner gegeben werden, wenn diese Erwartungen nicht erfüllt werden. *Terminator*-Knoten veranlassen, daß das CBSM für die aktuelle Aktivität beendet wird. Die (gerichteten) Kanten im Erwartungsnetz, die die Knoten verbinden, sind die möglichen Zustandsübergänge von einem *initiator* über ein Subnetz von *situation* zu *termination nodes*.

Practice Auf der Basis dieses einfachen reaktiven CBSM-Ansatzes wurde später von Bass et al. ([8], [9]) ein Trainingssystem für Piloten-Schüler entwickelt. Es basiert auf einem Flugsimulator und enthält ein Entscheidungsunterstützungssystem, das ein Erwartungsnetzwerk benutzt. Sie erhoffen sich eine Verbesserung der *situation awareness* in modernen Glass-Cockpits.

Die CBSM-Methode stellt also ein integriertes Schüler-/Tutor-Modul bereit, das sofort bei der Registrierung einer Abweichung von den Erwartungen an den Flugschüler eine Feedback-Reaktion auslöst und so die *situation awareness* trainieren soll.

Dabei werden vier Unterstützungsarten angeboten: *coaching*, *scaffolding*, *guiding* und *fading*. Beim *Coaching* werden entsprechende Text-Meldungen in einem Display erzeugt, die je nach Gefährlichkeit unterschiedlich eingefärbt sind. *Scaffolding* bedeutet, daß der Schüler durch eine Präsentation der Umsetzung der Aufgabe unterstützt wird, während er selbst versucht, sie durchzuführen. *Guiding* bedeutet für Bass, daß die relevanten Elemente der Aufgabenumgebung ihrer aktuellen Bedeutung entsprechend hervorgehoben werden. Diese Unterstützungsfunktionen werden immer seltener ausgeführt, je besser der Trainee wird (*fading*).

Das Entscheidungsnetzwerk ist so aufgebaut, daß bereits vor dem Erreichen gefährlicher Situationen eine Reaktion erfolgt.

Debriefing An die Praxis-Phase des Trainings schließt sich eine computergestützte Nachbesprechung (*debriefing*) an [11]. Während des CBSM wird zusätzlich zum reinen Ausführen von Feedback-Reaktionen ein Protokoll von Erwartungsabweichungen erstellt. Dieses Protokoll wird dann für das Debriefing in eine Zeitstrahl-Grafik transformiert, die dem Schüler von einem Tutor-Modul erläutert wird.

Das *situation awareness*-Trainingssystem von Bass ist für komplexe dynamische Situationen geplant. Deshalb wird eine Aufgabensimulation benutzt, deren Zustand ständig mit dem Erwartungsnetz abgeglichen wird. Führt der Schüler das System in einen Zustand, der als von Erwartungen abweichend erkannt wird, kann einfach ein nötiges Feedback zugeordnet werden und die Abweichung für das anschließende *debriefing* protokolliert werden. Die *guide*-Funktion basiert auch auf der Zustands-Beobachtung und hebt für die *situation awareness* relevante Elemente hervor.

Zur Verbesserung der Adaptivität wird ein Schülermodell benutzt, das aufgrund von Abweichungen arbeitet. Das Wissen des Schülers selber wird nicht explizit repräsentiert. Es wird in erster Linie der Zustand des Systems überwacht. Der Kontext, also der Zustand des Systems wird als Indikator für mangelnde *situation awareness* genutzt.

Ansatz von Gopher zum Training der Aufmerksamkeitsverteilung

Im Trainingssystem von Gopher ([35]) wird eine gute Verteilung und Kontrolle der Aufmerksamkeit zwischen unterschiedlichen Objekten trainiert. Das Trainingssystem ist in Form eines Computerspiels aufgebaut, bei dem es darum geht, mit einem Raumschiff Raketen auf ein sich bewegendes Ziel zu feuern ohne selbst von unterschiedlichen Angreifern getroffen zu werden. Dabei gibt es viele sich schnell bewegende Objekte, so daß die Anforderungen als sehr hoch eingestuft wurden.

Mehrere unterschiedliche Spielstrategien konnten festgestellt werden. Davon sind viele suboptimal. Beobachtungen zeigen, daß Versuchspersonen bei ihrer suboptimalen Strategie verharren, selbst wenn ihnen später eine bessere gezeigt wird. Beim Verfolgen ihrer Strategie werden sie jedoch besser.

Das Training bestand dann darin, die Veruchspersonen durch eine Sequenz von Situationen zu führen, in denen unterschiedliche Teile des Spiels betont wurden. Durch für die jeweilige Situation geeignete zusätzliche Spielelemente, wie spezielle Punktezähler wurden die Versuchspersonen ermuntert, sich auf einen bestimmten Aspekt zu konzentrieren. Die Situationen änderten jedoch nichts an den Spielregeln oder der Lastverteilung zwischen den Objekten. Versuchspersonen, deren Aufmerksamkeitsfokus so auf einen Teilaspekt des Spiels im Training gelenkt wurde, erzielten durch die Wahl einer besseren Spielstrategie ein besseres Ergebnis als eine Kontrollgruppe (bei gleicher Trainingsdauer), die das Spiel unstrukturiert erlernt hatte. Wurden jedoch zwei unterschiedliche Fokus-Situationen nacheinander trainiert, schnitten solche Versuchspersonen noch besser ab.

Ein Fokus-Situations-Training mit einer modifizierten Version dieses Spiels von Flugschülern, die gerade die theoretische Ausbildung abgeschlossen hatten und begannen, mit leichten Flugzeugen zu fliegen, zeigte eine große und signifikante Verbesserung der (realen) Übungsflugergebnisse verglichen mit einer Kontrollgruppe ohne Spieltraining. Der Vorteil der spieltrainierten Flugschüler lag vor allem in Manövern mit einer großen Belastung.

Das Ziel dieses Trainingssystems war nicht direkt die Verbesserung des Umgangs mit dynamischen Mensch-Maschine Systemen, sondern lag in der Untersuchung von Aufmerksamkeitsverteilung. Es hat sich in diesen Experimenten trotzdem gezeigt, daß eine Steuerung der Aufmerksamkeitsverteilung im Training, ähnlich dem *guiding* bei Bass, eine Verbesserung der Leistung in realen Situationen mit hoher Last bewirken kann.

2.4 Eigenes Konzept für ein Trainingssystem auf der Basis des kognitiven Modells

Aufbauend auf dieser Darstellung von Trainingssystemen für die Kontrolle dynamischer Mensch-Maschine Systeme, dem kognitiven Modell MoFL, der Fluglotsenaufgabe und dem Rahmenwerk für Intelligente Tutorielle Systeme soll im folgenden Abschnitt das eigene Konzept für ein Trainingssystem für Fluglotsen in der Streckenflugkontrolle vorgestellt und begründet werden.

Anforderungen

Das Trainingssystem soll das Erlernen von Strategien zur Kontrolle des Verkehrs für den Radarlotsen in der Streckenflugkontrolle unterstützen. Die Strategien sollen helfen, folgende Anforderungen an Fluglotsen zu erfüllen:

- den Verkehr zu kontrollieren, d.h. ein mentales Bild (*picture*) des Verkehrs aufzubauen,
- aufgrund dieses Bildes mögliche Flugverläufe zu antizipieren,
- dabei zukünftige Konflikte zwischen den Luftfahrzeugen zu erkennen und
- entsprechende Anweisungen an Piloten zu geben, damit diese Konflikte vermieden werden.

Das Erreichen dieser „Lernziele“ soll durch das zu entwickelnde Trainingskonzept unterstützt werden.

Fisher und Kulick präsentieren ein neues Trainingssystem für die Flugsicherung, das bei der FAA (Federal Aviation Administration) eingeführt wurde ([31, S. 279]). Sie stellen in diesem Zusammenhang drei strukturelle Forderungen an gute Trainingskonzepte in diesem Bereich:

- Im Training sollen die Strategien der Experten vermittelt werden.
- Training muß sich an individuellen Bedürfnissen des Schülers orientieren.
- Ein Trainingskonzept sollte von den Schülern verlangen, daß sie demonstrieren, ob sie das nötige Wissen beherrschen, bevor sie mit Anwendungsübungen beginnen.

Diese Anforderungen werden in dem Trainingskonzept, das in dieser Diplomarbeit entwickelt wird, umgesetzt. Es wird im folgenden beschrieben.

Simulation

Wie im Vergleich der unterschiedlichen computergestützten Ausbildungsangebote bei der DFS gezeigt wurde, ist es sinnvoll, ein Trainingssystem für die Kontrolle dynamischer Mensch-Maschine Systeme auf der Basis einer Simulation des technischen Systems zu begründen. Durch den Einsatz von Maßnahmen, die die Adaptivität des Systems steigern, entsteht ein Tutorsystem.

Eine Erhöhung der Mikroadaptivität mit konventionellen Methoden der Intelligenten Tutoriellen Systeme ist aus zwei Gründen nicht möglich:

- Aus der Beschreibung der Hilfsmittel des Exekutivlotsen wird klar, daß der Arbeitsablauf in erster Linie aus der Überwachung des Radarbildes besteht. Eingriffe in den Verkehr passieren vergleichsweise selten. Die Überwachungstätigkeit ließe sich nur schwer z.B. durch Blickbewegungs-Messungen einem Intelligenten Tutoriellen System zugänglich machen, was aber für ein Trainingssystem, dessen Vorteile ja in der Mobilität und dem vergleichsweise geringen Einsatz von Ressourcen liegen, unpassend wäre. Alternativ könnte die Maskierungstechnik (s. Abschnitt 3.1.2, S. 38) [65], [16] eingesetzt werden, um dem Diagnostik-Experten eines Intelligenten Tutoriellen Systems die Informationsaufnahme des Lotsenschülers zugänglich zu machen.

- Die Eingriffe des Exekutivlotsen erfolgen anhand der Ziele Sicherheit und Ökonomie, worunter z.B. auch der Komfort von Passagieren zu zählen ist. Die möglichen Präferenzen für Eingriffszeitpunkte und Lösungsstrategien sind aber so vielfältig und individuell bei Lotsen unterschiedlich, daß sich oft nicht mit einer einfachen Metrik entscheiden läßt, welche Güte ein konkret gemessener Eingriff hat. Da die gängigen Methoden bei der Realisierung von Diagnostik-Experten für Intelligente Tutorielle Systeme aber auf dem Erkennen von Fehlern oder Abweichungen einer vorgeplanten Strategie beruhen, lassen sich konventionelle Methoden Intelligenter Tutorieller Systeme nicht auf die Fluglotsenaufgabe übertragen.

Schülermodell

Basis für die Erhöhung der Adaptivität, bzw. zur *Orientierung an den individuellen Bedürfnissen des Schülers*, ist ein Schülermodell, auf dessen Basis die Mikroadaption geschieht. Wie gezeigt wurde, ist es unmöglich, ein solches Modell auf Grund von Abweichungen aufzubauen, weil nur selten meßbare Interaktion bei der Kontrolltätigkeit von Fluglotsen in der Streckenflugkontrolle passiert und weil die Güte dieser Eingriffe nicht klar beurteilt werden kann.

Der Ausweg ist ein CBSM-System, das auf den Zustand der Simulation zugreifen kann. Der Nachteil dieser Methode ist, daß dadurch kein Situationsabbild aufgebaut wird, das an den Prozessen und Strukturen des Menschen orientiert ist, sondern lediglich der Zustand des technischen Systems analysiert wird. Da die Güte der Eingriffe schwer zu beurteilen ist, sind auch die daraus folgenden Zustände des Luftverkehrsgeschehens nicht einfach zu bewerten.

Stattdessen wird für das CBSM das kognitive Modell MoFL eingesetzt. Es baut in seinem *picture* eine Repräsentation der Verkehrssituation auf, von der angenommen wird, sie sei analog zu der eines erfahrenen Lotsen.

MoFL arbeitet in diesem Szenario parallel zu dem Lotsenschüler. Es nimmt dieselbe Verkehrssituation wie der Lotsentrainee war. Während der Lotsenschüler das Szenario bearbeitet, baut MoFL sein *picture* auf und hält es aktuell. MoFL antizipiert die Flugverläufe der kontrollierten Flugzeuge und erkennt Konflikte oder andere Ereignisse. Dabei benutzt das Modell psychologisch fundierte Mechanismen und Strukturen, die denen eines erfahrenen Lotsen entsprechen.

Trainingsfunktionen

Aufgrund dieses Schülermodells ist keine Analyse von Fehlern des Lotsenschülers möglich, denn es wird das *picture* eines erfahrenen Lotsen konstruiert. Eine Adaption an den Kenntnis- bzw. Fertigkeitsstand des Trainees ist also nur in begrenztem Umfang möglich.

Stattdessen werden die konkrete Verteilung der Aktivierung im Arbeitsgedächtnis von MoFL und antizipierte Ereignisse wie Konflikte oder Überwachungsereignisse in der Art des *guiding* von Bass in der Simulation dargestellt, damit der Lotsenschüler lernt, in der unübersichtlichen Verkehrssituation am Radarschirm seine Aufmerksamkeit auf aktuell relevante Objekte zu verteilen. Die Ergebnisse von Gopher lassen vermuten, daß sich dadurch günstige Strategien erlernen lassen.

Die Trainingsfunktion beruht darauf, daß die Verkehrssituation im *picture* von MoFL psychologisch fundiert wie bei einem Menschen repräsentiert wird. Dabei wird von dem *picture* und den kognitiven Prozessen eines erfahrenen Lotsen ausgegangen, so daß *die Strategien der Experten im Training vermittelt* werden.

Einbettung

Diese Form des CBT kann von Schülern der DFS genutzt werden, die für die Streckenflugkontrolle ausgebildet werden. Aber auch zum Üben für den Erwerb einer weiteren Berechtigung für einen neuen Kontrollsektor kann das System von bereits zugelassenen Lotsen eingesetzt werden. Es kann ebenso bei einer Umstrukturierung des Luftraums eine Trainingsfunktion bieten. Es würde die aufwendige und teure Nutzung der DFS-Simulationen teilweise vermeiden.

Im Rahmen des DATS-Konzeptes der DFS (s. Abschnitt 2.3.1) kann das System vor allem in der Phase zwischen *institutional* und *operational training* eingesetzt werden. Es wird in erster Linie der Transfer theoretischen Wissens über die Luftraum- oder Verkehrsstruktur, bzw. über allgemeine Kontrollstrategien in praktisches Erfahrungswissen trainiert. Dazu sollte vorher natürlich angeleitetes Simulatortraining erfolgen. Später kann im Selbststudium die (simulierte) Praxis wiederholt oder vertieft werden. Der Einsatz würde also für einen DFS-Schüler in einer Phase der DATS-Ausbildung erfolgen, in der *der Schüler demonstriert hat, daß das nötige Wissen beherrscht wird.*

Fazit

Dieses neue computergestützte Trainingskonzept verbindet die Vorteile der existierenden CBT-Anwendungen in diesem Bereich (gute Makroadaption durch Flexibilität im Selbststudium) mit den Vorteilen, die die Simulatorausbildung in der Flugsicherungsausbildung hat. Bei der Simulatorausbildung kann theoretisches Wissen, das in der Ausbildung erworben wurde, in der Praxis in Fertigkeiten und Strategien zur Kontrolle der Verkehrssituation umgesetzt werden. Dazu ist eine Unterstützung durch einen Trainer erforderlich. Dadurch war das Simulatortraining nicht für das Selbststudium geeignet. Mit dem beschriebenen Trainingskonzept ist ein menschlicher Trainer für viele Situationen des Simulatortrainings nicht mehr notwendig. Die gute Anpassungsfähigkeit an den Schüler, die ein menschlicher Trainer hat, geht aber trotzdem durch den Einsatz des kognitiven Modells MoFL nicht verloren (Mikroadaption).

Dieses Trainingskonzept erfüllt die Anforderungen von Fisher und Kulick, wie in den Abschnitten Schülermodell, Trainingsfunktionen und Einbettung hervorgehoben wurde. Die „Lernziele“, also Strategien zum Aufbau des *pictures*, zur Antizipation, zum Erkennen von Konflikten und zum Eingriff zu erlernen, können besonders gut durch die Verwendung einer Simulation des Flugverkehrs vermittelt werden. Die im Abschnitt Trainingsfunktionen beschriebene *guiding*-Funktion soll dabei dem Trainee helfen, besonders günstige Strategien einzuüben.

Im folgenden 3. Kapitel wird beschrieben, wie dieses vorgestellte Trainingskonzept in einem neuen Softwaresystem für diese Diplomarbeit implementiert wurde.

In Abschnitt 3.2.4 (S. 47) und 3.3.4 (S. 66) wird die technische Umsetzung dieses Konzeptes mit dem kognitiven Modell MoFL beschrieben. In Abschnitt 3.2.4 werden unterschiedliche Ausprägungen des Einsatzes und seine organisatorische Einbettung diskutiert.

3 Entwicklung des Trainingssystems

Im folgenden Kapitel wird der Entwicklungsprozeß des Trainingssystems beschrieben. Dazu werden die Ergebnisse der Anforderungsanalyse vorgestellt. Auf dieser Basis wird der umgesetzte Entwurf entwickelt. Dessen Implementierung wird zum Schluß dieses Kapitels erläutert.

3.1 Anforderungsanalyse

Das Trainingssystem muß aus den in Kapitel 2 gezeigten Gründen die Bewegung der Objekte einer Verkehrssituation hinreichend realistisch in Echtzeit simulieren. Die Verkehrssituation soll dem Lernenden in einem Anzeigesystem präsentiert werden, das einem gängigen Fluglotsen-Arbeitsplatz ähnlich ist. Eingriffe des Schülers in die Simulation der Verkehrssituation müssen möglich sein.

Als Hilfsmittel sollen dem Lotsenschüler Flugstreifen (für die mittelfristige Planung) und Radarbild (für die operative Kontrolle) zur Verfügung gestellt werden. Diese beiden Informationsquellen sind für diese Aufgabe nahezu unerläßlich und werden deshalb am häufigsten benutzt. Andere Hilfsmittel des realen Arbeitsplatzes werden in diesem Trainingssystem vernachlässigt. Dazu gehören z.B. Informationssysteme für aktuelle Wetterdaten. Auch modernere (Software-) Planungshilfsmittel, wie sie an aktuellen Lotsenarbeitsplätzen vorhanden sind, sollen nicht umgesetzt werden. Sie bieten über die reine Darstellung von Informationen hinaus Funktionen an, die der Lerner selbst trainieren soll.

Das in Abschnitt 2.4 beschriebene Trainingskonzept auf der Basis des kognitiven Modells MoFL muß in das Trainingssystem integriert sein.

Neben der Anforderung, diese Grundfunktionen zu beherrschen, erwachsen aus der Art der Aufgabe und der Systemeinbettung einige zusätzliche spezifische Anforderungen an die Realisierung des gesamten Trainingssystems. Sie werden in den folgenden Abschnitten erläutert.

3.1.1 Verteiltheit

Das Trainingssystem soll aus getrennten Komponenten bestehen, die in eigenen Prozessen ablaufen. Für diese Trennung gibt es sowohl unterschiedliche Effektivitäts- und Effizienz-, als auch funktionale Gründe. In diesem Abschnitt werden Fragen zum Design vorwegnehmend erörtert, da sie zu einer zentralen Anforderung an den späteren Entwurf führen.

Die Anwendungsfunktion erfordert, daß Komponenten des Gesamtsystems auf unterschiedlichen Arbeitsstationen angezeigt werden, denn in einem Anwendungsfall soll es von zwei Ak-

teuren gleichzeitig benutzt werden: Der Lotsentrainee überwacht an einem Arbeitsplatz den Verkehr und greift durch eine Sprechfunksimulation in den Verkehr ein. Die Piloten der kontrollierten Flugzeuge werden von einem „Geisterpiloten“, als zweitem Akteur, simuliert. Er nimmt die Sprechfunkanweisungen entgegen und ändert mit einer eigenen Schnittstelle entsprechend die Daten in der Verkehrssimulation. Die Arbeitsplätze dieser beiden Akteure müssen räumlich getrennt sein.

Alternativ könnte ein Trainer zusammen mit einem fernlernenden Lotsenschüler ein Szenario bearbeiten. Es soll möglich sein, daß sich beide an unterschiedlichen Orten aufhalten. Sie können hier untereinander über den simulierten Sprechfunk bzw. Telefon und das Trainingssystem kommunizieren. Daher muß das Trainingssystem solche Kommunikationsfunktionen ähnlich einem Groupware-Programm enthalten. Beide müssen dabei eine identische Darstellung der Verkehrssituation haben.

Diese beiden Trainingssituationen bedingen, daß zumindest einige Komponenten verteilt in einem Rechnerverbund laufen müssen. Neben diesem funktionalen Argument gibt es Implementierungserfordernisse, die für ein Design mit getrennten Prozessen sprechen.

Das in das Trainingssystem zu integrierende kognitive Modell ist mit dem in diesem Gebiet verbreiteten Entwicklungswerkzeug ACT-R [4] implementiert worden. ACT-R wiederum setzt als Erweiterung auf CommonLISP auf. Es ist offensichtlich ungünstig, den Rest des Trainingssystems ebenfalls in LISP zu realisieren. Eine Integration von Modulen, die mit anderen besser geeigneten Programmiersprachen erstellt wurden, ist in einen LISP-Rumpf über *foreign function calls* möglich. Die Verwendung von *foreign function calls* sollte jedoch vermieden werden, weil die dazu nötigen Schnittstellen noch nicht standardisiert sind und das resultierende System deswegen kaum portabel sein würde. Die Einbindung eines CommonLISP-Interpreters in den Rest des Systems scheidet aufgrund mangelnder existierender Schnittstellen und der Größe und Komplexität eines modernen optimierenden Interpreter-Kerns mit automatischer *garbage collection* ebenfalls aus. Allenfalls einbettbare *scheme*-Interpreter sind im Moment verfügbar [33]. Der Aufwand, ACT-R von CommonLISP nach *scheme* zu portieren, wäre jedoch sehr hoch.

Es bietet sich also eine Implementierung mit unterschiedlichen, jeweils an die Aufgabe bzw. Komponente angepaßten Programmierumgebungen an. Die Möglichkeit, die entstehenden Module mit einer Script-Sprache unter Verwendung des „*glue code*“-Design-Patterns [61] zu integrieren, scheidet wegen der Unzugänglichkeit und Komplexität erhältlicher CommonLISP-Interpreter aus. Deshalb sollte das Trainingssystem als kooperierende Kollektion eigenständiger Prozesse entworfen werden.

Die realistische Simulation umfangreichen Flugverkehrs und deren Darstellung stellen genauso wie die Simulation der kognitiven Vorgänge mit ACT-R hohe Anforderungen an die Rechengeschwindigkeit, da alle Vorgänge möglichst zeitgleich in Echtzeit zu simulieren sind. Eine Verteilung der Komponenten auf unterschiedliche Rechner verspricht deshalb eine bessere Ressourcenauslastung.

Es existieren kommerzielle, aber auch unter der GPL[1] veröffentlichte Komponenten, wie z.B. Flugverkehrssimulationssysteme[2], die in das Trainingssystem integriert werden könnten. Existierende Komponenten sind leichter über eine externe Schnittstelle zu koppeln, als fest in einen

[1] GPL = *GNU Public License*. GPL-Software kann ohne Kosten benutzt und weiterentwickelt werden [58, S. 21]

[2] z.B. gats (*generic airtraffic simulation system*) [63]

Prozeß zu integrieren. Daher bringt die Realisierung des Trainingssystems in verteilten Komponenten den Vorteil, leichter zugängliche Schnittstellen zwischen den Teilen zu realisieren, so daß auch fremde Komponenten in das Gesamtsystem integriert werden können.

Desweiteren kann das Trainingssystem so durch Hinzufügen neuer Komponenten leichter um neue Trainingsfunktionen erweitert werden.

Diese Erörterung unterschiedlicher Gründe für ein Design mit verteilten Komponenten führt zu der Forderung nach einer umzusetzenden Architektur, in der Komponenten als autonome Agenten interagieren können. Es sollen unterschiedliche Programmiersprachen zur Implementierung der Komponenten einsetzbar sein. Sie sollen in einem Rechnernetzwerk verteilt werden können. Die internen Schnittstellen müssen sich einfach mit möglichen externen Systemen integrieren lassen.

3.1.2 Experimentalerweiterungen

Aus allgemeinen Erfahrungen mit der Benutzung von Luftverkehrssimulationen bei Organisationen, die für die Luftraumüberwachung zuständig sind und eigene Simulatoren zum Training, aber auch zum Test neuer Strukturen und Techniken des Flugverkehrs nutzen, lassen sich einige grundlegene Anforderungen für den allgemeinen Einsatz von simulierten Lotsenarbeitsplätzen ableiten [38].

Verkehr muß danach realistisch und in Echtzeit simuliert werden können. Das bedeutet, daß die Trajektorien der simulierten Luftfahrzeuge möglichst gut die Dynamik, die z.B. durch unterschiedliche Steig- und Sinkraten entsteht, widerspiegeln müssen. Die Berechnung des Ortswechsels zwischen zwei Radarupdates muß genau dem zeitlichen Abstand zwischen den Updates entsprechen. Dabei ist weniger wichtig, ob dieser Zeitraum immer exakt gleich lang ist. Das Ergebnis der Berechnung darf erst ab dem Zeitpunkt des Updates als neuer Zustand gültig werden.

In den professionellen Flugverkehrssimulatoren besteht die Möglichkeit, die Zeit zu straffen, also den Verkehr „schneller“ als in der Realität fliegen zu lassen. Diese Option bietet Vorteile bei der Untersuchung von Parametern auf Verkehrsströme. Auch diese Eigenschaft soll das Trainingssystem haben. Damit kann z.B. in einem Szenario ein Gefühl für die generellen Bewegungen im zu kontrollierenden Sektor vermittelt werden.

In dem DFG-Projekt „Modellierung von Fluglotsenleistungen in der Streckenflugkontrolle“ [18] und der ebenfalls von der DFG geförderten Forschergruppe „Mensch-Maschine-Interaktion in kooperativen Systemen der Flugsicherung und Flugführung“ [64] wurden Luftverkehrssimulationen mit realistischen Lotsenarbeitsplätzen entwickelt, um kognitive Prozesse bei Fluglotsen in der Streckenflugkontrolle mit psychologischen Experimenten zu untersuchen. In diesen Projekten entstand das kognitive Modell MoFL (s. Abschnitt 2.2.4). Die Arbeit mit den beiden Simulationssystemen EnCoRe-PLuS [16] und simco [34] hat gezeigt, daß ein solches System mit vielfältigen Erweiterungen ausgestattet sein muß, um für diese Aufgabe nützlich zu sein.

Aus den Erfahrungen mit diesen beiden Systemen folgen mehrere Anforderungen an das neu zu entwickelnde System. Zur Durchführung psychologischer Experimente sind folgende Eigenschaften nützlich:

- Die Simulationszeit soll in Bezug auf die Realzeit in festzulegenden Grenzen beliebig skalierbar sein. Das bedeutet insbesondere, daß zusätzlich zu der schon formulierten An-

forderung der „Zeitbeschleunigung“ auch eine „Zeitverlangsamung“ möglich sein soll; also eine Skalierung der Zeit mit einem Faktor sowohl größer als Null, als auch kleiner als Eins.

- Das System muß auch angehalten werden können (*freeze*). Dabei wird i.A. nur der Zustand der Luftraumsimulation eingefroren. Zusätzlich sollen sich die Trainingskomponenten getrennt anhalten lassen.

- Der vom Versuchsplan vorgegebene Ablauf der Simulation, also die Trajektorien der Luftfahrzeuge, kann vom Lerner durch seine Eingriffe geändert werden. Der durch die Eingriffe erzeugte Verkehr muß aufgenommen und später wieder eingespielt werden können (*replay*).

- Der Beginn der Simulation muß in Bezug auf den vorgefertigten Verkehr im Versuchsplan bzw. im *replay* frei festgelegt werden können. Das bedeutet, daß der Beginn des Simulationsablaufes an einer beliebigen Stelle im simulierten Verkehr liegen kann.

- Am simulierten Lotsenarbeitsplatz, der Schnittstelle des Trainingssystems zum Lotsenschüler, sollen modale Dialogfenster mit unterschiedlichen Ausprägungen eingeblendet werden können, in denen z.B. Ja/Nein-Fragen dargestellt werden.

Neben der Nützlichkeit für psychologische Experimente sind alle diese Eigenschaften auch für ein Trainingssystem wichtig.

Die Schnittstelle, über die diese Kommandos abgesetzt werden, muß sich von unterschiedlichen Stellen ansprechen lassen:

- Im Versuchsplan, der den Ablauf des Experimentes steuert, kann zu bestimmten Zeitmarken ein entsprechendes Kommando aktiviert werden.

- Der Versuchsleiter kann zu einem von ihm gewählten Zeitpunkt über eine grafische Schnittstelle ein entsprechendes Kommando senden.

- Ein Modell der Relevanz der Objekte und Konstellationen in der Verkehrssituation, entweder als Teil der Luftraumsimulation oder getrennt von ihr in der Trainingskomponente, kann aufgrund seines Zustandes ein solches Kommando aktivieren.

Diese unterschiedlichen Möglichkeiten, den Ablauf der Simulation zu beeinflussen, müssen im Entwurf der Schnittstelle zur Luftraumsimulation vorgesehen werden.

Protokollierung

Alle Eingriffe des Lotsenschülers, sowie besondere Ereignisse, wie Separationsunterschreitungen, die unerlaubte Zustände markieren, müssen protokolliert werden. Dabei wird eine entsprechende Meldung in ein Logfile geschrieben, das nach dem Versuchsdurchlauf ausgewertet werden kann. Für die Wahl des Protokollformats sind grundsätzlich drei denkbare Ausprägungen möglich:

Klartextprotokollierung: Bei der Klartextprotokollierung werden die Ereignisse und deren etwaige Parameter mit ihren natürlichsprachlichen Namen vermerkt. Dabei wird i.A. jedes Ereignis in einer eigenen Zeile gespeichert, wobei jedoch kein spezielles Spaltenformat

eingehalten wird. Ein sehr formales Klartextprotokoll wird z.B. vom Simulationssystem EnCoRe-PLuS benutzt [16].

Protokollierung in der Syntax des Auswertungswerkzeuges: Die Ereignisse und ihre Parameter können auch einem Code zugeordnet werden. Durch die Verwendung eines geeigneten Codes entsteht eine feste Spaltenzuordnung, die ein Einlesen in Werkzeuge wie MS Excel oder SPSS erleichtert. Corker und Smith beschreiben ein Simulationssystem dessen Logfiles in Excel-Syntax vorliegen [26].

Matrixprotokollierung: Es existieren Datenformate mit dazugehörenden Software-Bibliotheken, mit denen insbesondere hochvolumige, multidimensionale Daten gespeichert werden können. Die bekanntesten Vertreter sind CDF (*common data format*; [36], [37]) und HDF (*hierarchical data format*; [32]). Systeme, die gemessene Daten in solchen Protokolldateien speichern, und Auswertungsprogramme, die die so gespeicherten Daten aufbereiten oder auswerten, müssen sich der dazugehörenden Software-Bibliothek bedienen, um auf das benutzte Format zugreifen zu können. Standardsoftware in dem hier beschriebenen Bereich, wie MS Excel oder SPSS, haben keine solche Schnittstelle zu CDF oder HDF.

Da selbst bei einem langen Simulationsszenario keine wirklich großen zu protokollierenden Datenmengen anfallen, kann auf ein Format wie CDF verzichtet werden, weil die Auswertungswerkzeuge dann unnötig komplex würden. Die Erfahrung hat gezeigt, daß selbst auswertungsnahe Formate mit Reportgeneratoren aufbereitet werden müssen. Deshalb scheint eine Klartextprotokollierung, wo ein eindeutiges Format zur späteren Transformation mit einem Werkzeug wie Perl oder *awk* benutzt wird, am hilfreichsten zu sein, weil Ergebnisse oder mögliche Systemprobleme vom Versuchsleiter oder vom Trainer schnell aus den Klartext-Logfiles gelesen werden können.

Zur späteren Auswertung sind Zeitverläufe von entscheidender Bedeutung, deshalb müssen alle protokollierten Ereignisse mit dem Zeitpunkt ihres Eintretens gespeichert werden. Um auch Reaktionszeitmessungen zu ermöglichen, muß die Auflösung der protokollierten Zeit im Bereich von wenigstens $\frac{1}{100}$ Sekunde liegen.

Maskierungstechnik

Die Maskierungstechnik wurde entwickelt, um ohne aufwendige Blickbewegungsmessungen Zugriff auf den Vorgang der Informationsaufnahme beim Umgang mit Computerprogrammen zu gewinnen (*moving window technique* in [70]). Sie wurde bereits im Bereich der Luftverkehrsüberwachung eingesetzt [16].

Alle alphanumerischen Informationen auf dem Simulationsdisplay werden mit einem grafischen Element, der Maske, verdeckt, so daß nur ihre Existenz, nicht aber die dargestellten Werte sichtbar sind. Durch Überfahren mit der Maus wird die Maske aufgedeckt, und die dargestellte Information sichtbar gemacht. Der Eintritt des Mauszeigers in die Maske wird registriert und bewirkt das Transparentmachen der Abdeckung. Die Informationen unter der Maske bleiben solange sichtbar, wie sich der Mauszeiger über dem abgedeckten Feld befindet (s. Abb. 3.1). Wird der Mauszeiger aus dem Feld bewegt, wird die Maske wieder aktiv, und die dargestellte Information wird wieder unlesbar. Zeitpunkt und Dauer der Aufdeckungen werden registriert und protokolliert.

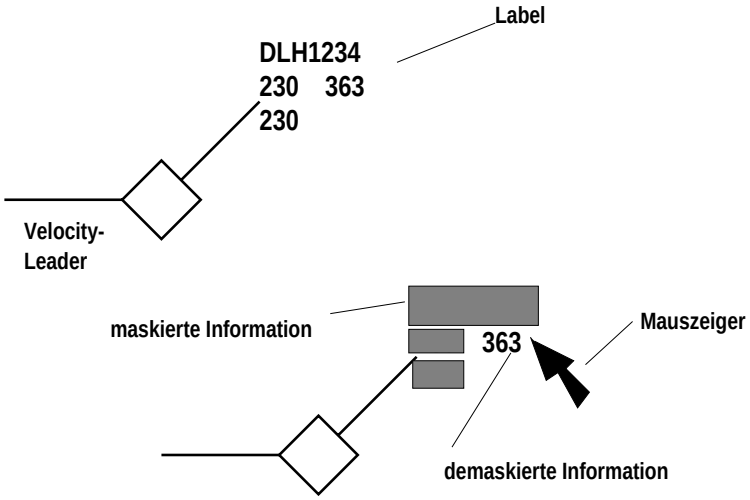

Abbildung 3.1: Darstellung der Maskierungstechnik. Es werden zwei Luftfahrzeuge dargestellt. Die alphanumerischen Informationen im Label des oberen sind nicht maskiert. Beim unteren Luftfahrzeug sind alle alphanumerischen Informationen des Labels maskiert, bis auf eine, die durch den Mauszeiger demaskiert wurde.

Obwohl sich dadurch die Aufgabe ändert, konnte gezeigt werden, daß die so erhobenen Daten mit Blickbewegungsmessungen vergleichbar sind [65].

Die Maskierungstechnik eröffnet einen Zugang zu den Überwachungstätigkeiten der Fluglotsen. Über die Analyse der Informationsaufnahme am Radarbildschirm, der Hauptinformationsquelle des Exekutivlotsen, lassen sich möglicherweise die aktuell verfolgten Strategien und deren Fehler diagnostizieren, die dann von einer Wissensvermittlungskomponente benutzt werden könnten, entsprechende Hinweise oder visuelle Hilfen in einem Trainingssystem zu geben.

Auch zur Untersuchung des Umgangs der Lotsenschüler mit dem Trainingssystem könnte sich die Maskierungstechnik als hilfreich erweisen. Deshalb soll sie als Option im Trainingssystem vorgesehen werden.

3.1.3 Planen der Veränderung

Besonders wichtig ist die Flexibilität des zu planenden Trainingssystems. Es muß ohne großen Aufwand möglich sein, neue Funktionalitäten hinzuzufügen oder bestehende Funktionen zu ändern. Der Zeitpunkt dieser Änderungen kann unterschiedlich sein.

Es ist zu erwarten, daß sich das Flugsicherungssystem in der Zukunft ändern wird. Es wird neue Unterstützungssysteme an den Fluglotsenarbeitsplätzen geben; die Schnittstelle der verwendeten Systeme wird weiterentwickelt; Verkehrs- und Luftraumstrukturen werden sich ändern. Deshalb werden von Flugsicherungsinstitionen wie NATS in Großbritannien Anstrengungen unternommen, Entwicklungsverfahren einzuführen und in die bestehende Organisation einzubinden, die eine kontinuierliche Änderung und Bewertung auch der Simulationssysteme ermöglichen [44]. Um eine realistische Trainingsumgebung bereitzustellen, müssen diese Änderungen am (realen) Flugsicherungssystem auch im Trainingssystem nachvollzogen werden.

Maßnahmen zur Änderung

Während eine Änderung der Verkehrsstruktur durch einfache Änderungen an der *Konfiguration* von Flug- bzw. Simulationsplänen und neu darzustellende Luftraumkarten in den Radarsimulationsplätzen am Trainingssystem kompensiert werden kann, sind die anderen Veränderungen tiefgreifender.

Für eine Revision der Arbeitsplatzschnittstelle bzw. Softwareoberfläche muß in den Programmcode des Darstellungssystems eingegriffen werden. Deshalb muß dessen interne Schnittstelle dokumentiert sein und Erweiterungen erlauben. Dieser Bereich deckt sich mit dem der *Prototyping*-Methoden in der Systemerstellung, deren Bedeutung besonders in der menschzentrierten Systemgestaltung im Flugsicherungsbereich betont wird [69, S. 215], [15].

Sollen dem Lotsenarbeitsplatz jedoch zusätzliche Funktionalitäten, wie neue Unterstützungssysteme hinzugefügt werden, müssen neue Komponenten, bzw. Systemmodule erstellt und dem Gesamtsystem hinzugefügt werden. Die Herausforderung an solchen Änderungen liegt in der Integration in das Gesamtsystem. Deshalb müssen übergeordnete Schnittstellen, die die Teilsysteme verbinden, offen angelegt sein. Dieser Bereich der Änderungen läßt sich mit *Componentware* (s. z.B. [60]) abdecken.

Zeitpunkt der Änderung

Die Zeitpunkte der nötigen Erweiterungen sind entsprechend unterschiedlich. Zu planen sind z.B. Veränderungen am Trainingskonzept, die aus den Erfahrungen mit der Qualität der didaktischen Komponente des Trainingssystems entstehen. Sie erfolgen zu einem späten Zeitpunkt des Lebenszyklus' des Trainingssystems. Auch Änderungen, die aufgrund der Weiterentwicklungen im Flugsicherungsbereich eingefügt werden müssen, werden erst sehr spät passieren.

Hingegen werden Luftraum- und Verkehrsstrukturänderungen häufiger und früher nötig werden. Die Basis an Trainingseinheiten oder -Lektionen, auf der das Trainingskonzept aufbaut, muß ständig erweitert werden.

Mittelfristige Änderungen an der Schnittstelle im Sinne des *rapid prototyping* beim Auftreten von Benutzungsproblemen nehmen eine Zwischenstellung in Bezug auf den zu erwartenden Änderungszeitpunkt ein. Daneben werden diese Änderungen aber auch langfristig nötig, wenn sich der Fluglotsenarbeitsplatz bei den Institutionen, die für die Flugsicherung zuständig sind, ändert.

An das zu entwickelnde Trainingssystem muß als Anforderung gestellt werden, daß Änderungen, die auf unterschiedlichen Ebenen, nämlich Konfiguration, Oberflächen-Prototyping und Komponentenmanagement zu ganz verschiedenen Zeitpunkten im Lebenszyklus des Systems möglich sind. Der Aufwand, eine Änderung des Systems einzuführen, muß gering sein, damit eine schnelle Adaption an neue Anforderungen möglich bleibt.

3.2 Entwurf

Die beschriebene Anforderungsdefinition deckt bereits einen Teil des Entwurfes ab. Die Alternativen für konkrete Ausprägungen der Designentscheidungen werden im folgenden Abschnitt erörtert.

In den folgenden Abschnitten 3.2 und 3.3 wird für Diagrammdarstellungen der Formalismus UML (*unified modeling language*; s. z.B. [3]) benutzt. UML vereint die Eigenschaften und Notationen der unterschiedlichen objektorientierten Analyse- und Designwerkzeuge und setzt sich in diesem Bereich als allgemein akzeptiertes Modellierungswerkzeug durch.

3.2.1 Rahmenarchitektur

Für den Entwurf des Trainingssystems wird eine Rahmenstruktur definiert. Sie beschreibt die Ideen, welche Funktionen in welchen Komponenten zusammengefaßt werden und wie die Komponenten verknüpft werden.

Um die Anforderungen nach Verteiltheit (s. Abschnitt 3.1.1), leichter Erweiterbarkeit (s. 3.1.2) und Veränderbarkeit (s. 3.1.3) zu erfüllen, wird ein Ansatz gewählt, bei dem einzelne Programmodule als eigenständige Prozesse ablaufen. Dieses Konzept unterstützt die Anforderung nach leichter Verteilbarkeit in einem Rechnerverbund per se. Durch die im Vergleich zu einem großen monolithischen System kleineren Programmeinheiten wird die Wartbarkeit unterstützt, weil weniger Abhängigkeiten bestehen. Durch Hinzufügen oder Auswechseln weiterer Modul-Prozesse kann die Funktionalität erweitert oder geändert werden, ohne daß andere Komponenten betroffen sind. Eine Änderung der Funktionalität wird im günstigsten Fall durch eine Rekonfiguration der beteiligten Prozesse erreicht. Dadurch läßt sich ein solches System gut in Experimentalumgebungen einsetzen.

Kopplung

Die entstehenden Komponenten müssen über eine wohldefinierte Schnittstelle gekoppelt werden. Diese Schnittstelle muß so beschaffen sein, daß folgende wichtige Randbedingungen erfüllt werden:

- Die Schnittstelle muß es ermöglichen, Prozesse von Programmen, die mit unterschiedlichen Programmierumgebungen erstellt wurden, zu koppeln.
- Die Kopplung muß auch über mehrere Rechner hinweg möglich sein.
- Metaprogrammierung als Werkzeug zur Umkonfiguration oder Änderung der Funktionalität einer beteiligten Komponente muß möglich sein (s. z.B. [62]).

In der Informatik gibt es unterschiedliche ausgearbeitete Konzepte zur Kopplung verteilter Komponenten. Ihre Eigenschaften werden im folgenden Abschnitt beschrieben. Auf dieser Grundlage wird auf S. 43 der eigene Ansatz erörtert.

Im Forschungsgebiet der verteilten Künstlichen Intelligenz werden Softwareeinheiten (Agenten) miteinander verbunden und mit Koordinations- und Kooperationswissen oder -verhalten ausgestattet, damit sie gemeinsam ein Problem lösen. Die Kopplung kann auf unterschiedlichen Ebenen untersucht werden (Kooperation, Koordination und Kommunikation).

Der Begriff des Agenten läßt sich auf die Komponenten in verteilten, kooperativen Simulationsumgebungen übertragen. Durch die Funktionsteilung und die Heterogenität der in diesem

Simulationssystem entstehenden Komponenten werden Kooperation und Koordination durch die entstehenden Agenten zum großen Teil selbst behandelt. Es ist deshalb nicht nötig, ein explizites Konzept dafür zu entwerfen.

Zur Kommunikation können jedoch unterschiedliche Ansätze und Methoden benutzt werden. Sie sollen kurz vorgestellt und für die Aufgabe in diesem Simulationssystem verglichen werden[3].

- Beim *Blackboard-Ansatz* wird eine zentrale Datenstruktur eingeführt, die von allen beteiligten Agenten zugreifbar ist. Dort werden Zwischenergebnisse und Anfragen gespeichert und von anderen Agenten gelesen. Die Aufgabe eines Blackboard-Systems ist also in erster Linie die formatunabhängige Repräsentation von Nachrichten und eine Synchronisation und Absicherung von Lese- und Schreibzugriffen. Das Ziel beim Einsatz eines Blackboards ist, entweder unterschiedliche Wissensrepräsentationsmechanismen unterschiedlicher Teilsysteme zu integrieren (s. z.B. bei [2]) oder unterschiedliche Ebenen der Problemlösung zu koppeln (s. z.B. bei [19, Kap. 11]).

- Beim *message passing* kommunizieren Agenten direkt miteinander. Sie tauschen Nachrichten in einem vorher festgelegten Protokoll aus. *Message passing* beschreibt lediglich einen Kommunikations- bzw. Transportmechanismus. Die Semantik liegt in dem verwendeten Protokoll.

  Zum Vergleich werden hier zwei Protokolle aufgeführt, die mit unterschiedlichen Zielstellungen entwickelt wurden.

  – Im Bereich der Agententechnologie ist als *message passing*-Protokoll besonders KQML (*knowledge query and manipulation language*) weit verbreitet. Es basiert auf der Sicht, Kommunikation sei der Austausch von Sprechakten[4]. Sprechakte übernehmen dann zu einem großen Teil die Funktion eines Koordinationsmechanismus'.

    Beim Entwurf einer Agentengesellschaft werden Rollen definiert. Die Modellierung nach Rollen dient u.a. der Koordination. Bspw. kann in einer Agentengesellschaft die Rolle eines Maklers vergeben werden. Ein Makler hat Kontakt zu allen anderen Agenten und kann Aufgaben nach Dringlichkeit oder den Fähigkeiten der anderen Agenten zuteilen. Durch die Organisation der Agenten nach Rollen wird die Koordination zwischen ihnen implizit definiert. KQML bietet auf einer anderen Koordinationsebene die Möglichkeit, Dialoge zwischen zwei Agenten durchzuführen.

  – Ein anderes Protokoll, das aus dem Bereich der verteilten Simulation und nicht der verteilten Künstlichen Intelligenz stammt, ist DIS (*distributed interactive simulation*). Verteilte Komponenten werden über *message passing* miteinander verbunden. Im Protokoll sind sowohl Koordinations- als auch Informationseinheiten vorgesehen. Dieses Protokoll ist äußerst komplex und damit mächtig. Es wird in existierenden Simulationsumgebungen des Militärs und der Luftfahrt eingesetzt.

    DIS benutzt nicht explizit die Terminologie von Agenten. Im Mittelpunkt der Koordination der Komponenten steht auch nicht die Rollenkoordination, sondern Konstrukte zum Sicherstellen der Realzeitfähigkeit und der Synchronisation zwischen den verteilten Komponenten.

[3] Auch die *konnektionistische Verknüpfung* von Einheiten (simulierte Neuronen) in einem künstlichen neuronalen Netzwerk wird als Agentenkopplung aufgefaßt. Die dabei benutzten Ansätze lassen sich aber offensichtlich nicht auf das hier entwickelte Simulationssystem übertragen.

- Im Bereich der *middleware* werden Architekturen zur Kopplung von Systemen entwickelt, z.B. zur Verbindung eines Anwendungsprogramms mit unterschiedlichen Datenbanken. Durch die weite Verbreitung der Objektorientierung in der Anwendungsentwicklung geschieht diese Kopplung inzwischen durch die Kommunikation von Objekten untereinander. Dafür wurden unterschiedliche softwaretechnische Umgebungen entworfen und umgesetzt. Sie basieren auf dem Ansatz, Kommunikation sei der Aufruf von Methoden bei entfernten Objekten, d.h. von einem Rechner zu einem anderen. Zu diesen Mechanismen zählen RPC (*remote procedure calls*), RMI (*remote method invocation* zwischen Java-Objekten) und CORBA (*common object request broker architecture*). Mit CORBA und bedingt auch mit RPC können die kommunizierenden Objekte in unterschiedlichen Programmiersprachen implementiert sein. Das CORBA-Konzept sieht Mechanismen zur Echtzeitkontrolle vor, die jedoch von den meisten ORB-Implementierungen (*object request brokern*) nicht umgesetzt werden.

  Im Gegensatz zu *blackboards* und *message passing* werden hier keine Agenten, sondern Objekte betrachtet.

Im folgenden Abschnitt wird erörtert, welche der beschriebenen Ansätze zur Kopplung von Komponenten sich für die Implementierung des Trainingssystems eignen und eine eigene Architektur daraus entwickelt.

Wegen der Komplexität der nötigen Implementierungsarbeiten, bzw. der teilweise fehlenden Unterstützung der zu verwendenen Implementierungsumgebungen für das Trainingssystem, sind alle Kommunikationsmethoden bis auf einfaches *message passing* zwischen den Komponenten nicht anwendbar. *Message passing* schränkt weder die Verwendung heterogener Implementierungsumgebungen noch die Verteilung über vernetzte Rechner ein.

Zur Anwendung von *message passing* zwischen den Komponenten wird eine eigene Architektur anhand der Rollenverteilung zwischen den Komponenten eingeführt und ein eigenes Protokoll entworfen.

Für die Koordination wird eine weitere Komponente eingeführt, die die Rolle eines zentralen Kommunikationsservers übernimmt. Alle Nachrichten werden über diesen „Kopplungsserver“ an die zuständigen Komponenten verteilt. Er übernimmt weitere zentrale Aufgaben, wie z.B. die Protokollierung. Jede Komponente muß sich also nur mit diesem zentralen Agenten verbinden, so daß eine „Stern-Architektur“ entsteht, in der der Kopplungsserver im Zentrum steht.

Das Format des *message passing* Protokolls ist auf einfache (menschliche) Lesbarkeit und einfache Parsierbarkeit mit regulären Ausdrücken ausgelegt: Jede Nachricht beginnt mit einer Kennung des Senders, die beim Weiterreichen durch den Kopplungsserver an den eigentlichen Empfänger nicht geändert wird. Darauf folgt ein Symbol, das den Typ des Sprechaktes kennzeichnet. Schließlich folgen die Parameter der Instanz des Sprechaktes. Jeder Sprechakt wird in einer einzigen Zeile versandt.

Anwendung

Diese Systemarchitektur unterstützt die geplante Benutzung des Gesamtsystems (s. Abb. 3.2):

[4]Ein Sprechakt ist ein Element eines Kommunikationsvorganges. Die Bezeichnung als Sprechakt impliziert die Sicht, daß der Zweck der Kommunikation, also z.B. das Auslösen einer Reaktion oder das Informieren des Empfängers, im Vordergrund steht. Sprechakte werden nach der Art der auszulösenden Reaktion klassifiziert und bilden so einen geeigneten Rahmen, um zielgerichtete und situierte Dialogführung und Koordination von Software-Agenten zu formalisieren.

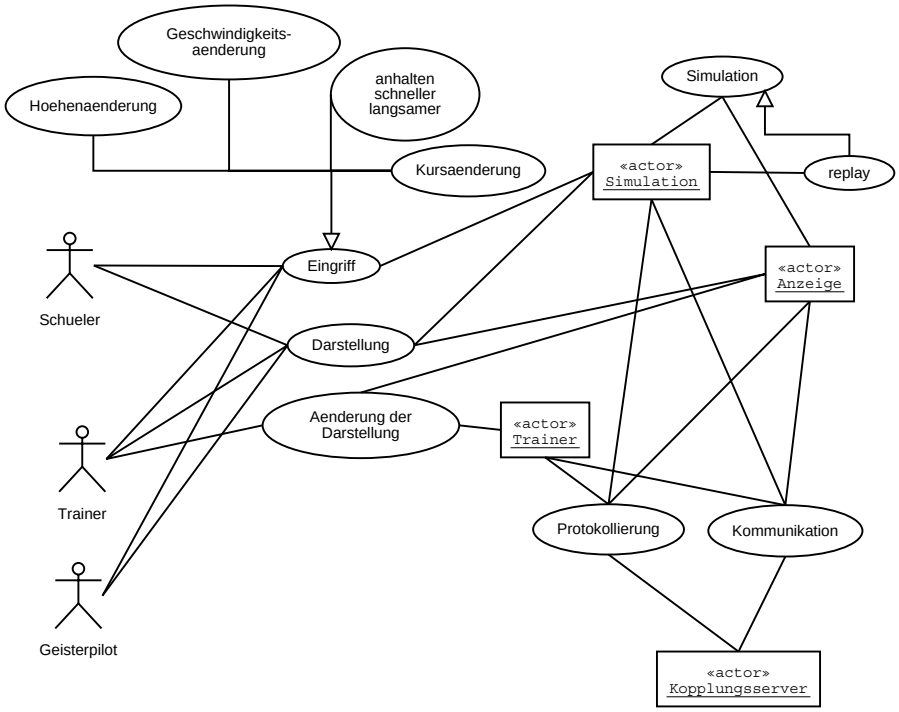

Abbildung 3.2: Use-Case-Diagramm des gesamten Simulations- und Trainingssystems in UML.

Alle Benutzungsmöglichkeiten (*use-cases*) geschehen über *message passing* zwischen den beteiligten Komponenten über den Kopplungsserver (*Kommunikation*). Der Kopplungsserver als zentrale Instanz ist auch für die *Protokollierung* zuständig, die entweder automatisch passieren kann oder durch einen speziellen Sprechakt veranlaßt wird.

Alle beteiligten (menschlichen) Akteure erhalten eine *Dartsellung* über die Anzeige-Komponente, die dafür auf Informationen der Simulation zurückgreift. Die *Darstellung* der Anzeige-Komponente kann von Trainer-Akteuren (sowohl menschliche, als auch technische) *geändert* werden, wozu der Anzeige-Komponente Sprechakte geschickt werden. Alle menschlichen Akteure können in den Verkehr eingreifen. Dazu wird eine entsprechende Nachricht an die Simulationskomponente geschickt. Die Simulationskomponente selbst erbringt den zentralen Dienst der Fortschreibung der simulierten Flugzeugbewegungen, deren Positionen an die Darstellungskomponenten übermittelt werden. Das Replay ist ein spezieller Dienst der Simulation bei dem eine bereits durchgeführte Simulation erneut eingespielt wird.

3.2.2 Simulation

Der Entwurf der Simulationskomponente ist generisch: Die Einbettung ist unabhängig vom Anwendungsbereich Luftfahrt. Deshalb wird eine allgemeine Schnittstelle definiert (s. Abb. 3.3). Die allgemeine Schnittstelle (`Simulation`) hat ein Attribut, in dem die aktuelle Simulationszeit vermerkt wird (in Sekunden des Simulationstages seit 0:00 Uhr). Die Methode `init_objects` lädt eine Reihe von Flug-Objekten (`Flight`). Die Methode `replay_dump` erzeugt eine textuelle Repräsentation der aktuellen `Flight`-Objekte, die später für ein *replay* benutzt werden kann. `exception` prüft, ob eine Ausnahme-Situation vorliegt (z.B. eine Separationsunterschreitung zwischen zwei Luftfahrzeugen) und gibt eine textuelle Repräsentati-

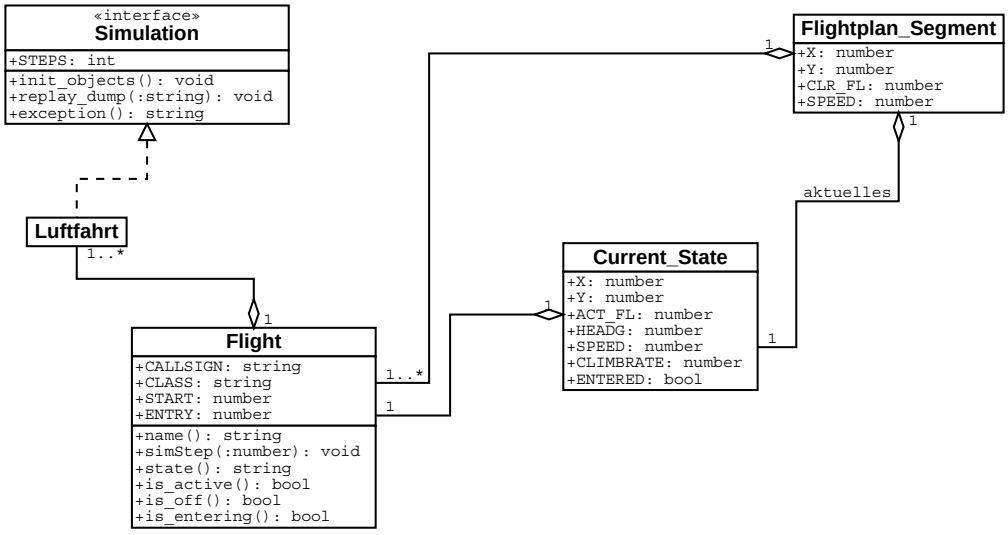

Abbildung 3.3: Klassendiagramm der Simulationskomponente in UML.

on dieser Ausnahmebedingung zurück. Diese Schnittstelle wird im Simulationssystem von der Klasse `Luftfahrt` implementiert.

`Luftfahrt` hat Referenzen auf eine Menge von `Flight`-Objekten. Jedes `Flight`-Objekt repräsentiert ein Flugzeug, dessen Bewegung simuliert wird. Für jedes Flugzeug wird vor der Simulation ein Flugplan aufgestellt. Der Flugplan besteht aus einer verketteten Liste von Segmenten (`Flightplan_Segment`). Jedes Segment markiert einen Punkt im Raum, der angeflogen wird und die Geschwindigkeit, die das Flugzeug zum Anfliegen des nächsten Punktes erreichen soll.

Der aktuelle Zustand jedes Flugzeuges wird in einer eigenen Struktur (`Current_State`) repräsentiert, die auch eine Referenz auf das aktuell bearbeitete Flugplansegment enthält (d.h. welcher Punkt gerade angeflogen wird).

Zur Vereinfachung der Darstellungskomponente wird als räumliche Basis (in der horizontalen Ebene) das Pixelkoordinatensystem der Darstellungskomponente benutzt. Dadurch werden einige einfache Umrechnungen z.B. für die Geschwindigkeit notwendig.

Ein Eingriff des Geisterpiloten in den Simulationsablauf wird als das Einfügen eines neuen Flugplansegmentes und eine Manipulation des aktuellen Segmentes für diesen Flug (s. Abb. 3.4) umgesetzt. Das betrachtete Flugzeug fliegt im Moment den Punkt (300,600,230) mit einer Geschwindigkeit von 300 an (aktuelles Segment in „Plan“). Es befindet sich gerade am Punkt (200,500,200) als die Anweisung kommt, die Höhe auf 100 zu ändern. Also wird der Flugplan so geändert, daß der aktuelle Ort ein Anflugziel wird (erstes Segment im Flugplan „nach Eingriff“) und der eigentliche Zielpunkt mit dem geänderten Parameter CLR_FL, also (300,600,100) angesteuert wird (zweites Segment im Flugplan „nach Eingriff“).

Dadurch spiegelt der Flugplan als die Menge der Flugplansegmente den gesamten Flugverlauf wieder. Das wird benutzt, um ein *replay* zu ermöglichen: Bei jedem Eingriff wird der modifizierte Flugplan als textuelle Repräsentation in eine Datei geschrieben. Er kann in einer späteren Simulationssitzung erneut eingelesen werden. Dann werden die Routen mit den erfolgten Eingriffen abgeflogen.

Der Aufbau der textuellen Repräsentation des Flugplans orientiert sich an den verwendeten Datenstrukturen. Die Syntax der Flugpläne entspricht der Programmiersprache Perl, weil da-

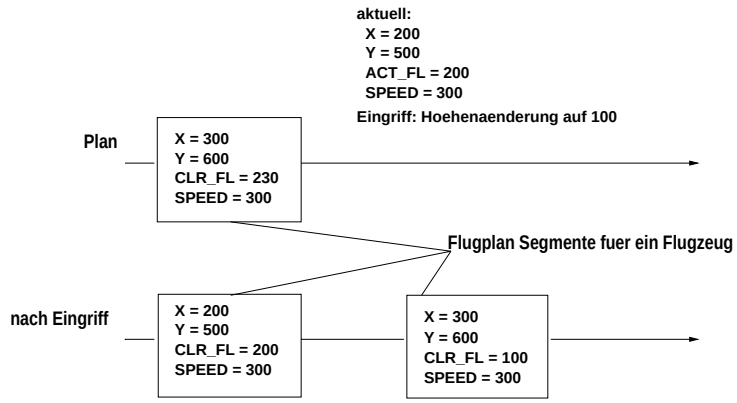

Abbildung 3.4: Höhenänderung als Eingriff in den Flugplan. Es wird ein neues Flugplansegment erzeugt.

```
{CALLS => 'DLH2439', CLASS => 'A310', START => '36600', ENTRY => '37646',
 FPL => [{X => 883, Y => 1020, CLR_FL => 0, SPEED => 0.33794333},
         {X => 741, Y => 848, CLR_FL => 180, SPEED => 0.67215243},
         ...
         {X => 45, Y => 259, CLR_FL => 220, SPEED => undef}]},
{CALLS => 'EWG441', CLASS => 'ATR', START => '36480', ENTRY => '37287',
 FPL => [{X => 982, Y => 454, CLR_FL => 180, SPEED => 0.40786625},
         {X => 830, Y => 533, CLR_FL => 180, SPEED => 0.44278161},
         ...
         {X => 78, Y => 583, CLR_FL => 220, SPEED => undef}]},
...
```

Abbildung 3.5: Ausschnitt aus einem Flugplan zur Verdeutlichung des Formats.

durch bereits ein einsetzbarer Parser für dieses Format vorliegt. Entweder wird Perl direkt zur Implementierung benutzt oder der Perl-Parser zu dem Programm hinzugelinkt.

Im Flugplan (s. Abb. 3.5) ist der Zeitpunkt des Eintritts in den zu kontrollierenden Sektor explizit vermerkt (`ENTRY`). `CLASS` gibt den Flugzeugtyp an. Er wird zur Parametrisierung einiger Dynamikeigenschaften des Flugzeuges benutzt. Alternativ kann statt der Geschwindigkeit (`SPEED`) auch der Zeitpunkt des Erreichens des spezifizierten Punktes angegeben werden.

3.2.3 Anzeigesystem

Das Anzeigesystem steht jedem beteiligten (menschlichem) Akteur in einer eigenen Instanz zur Verfügung. Es beinhaltet immer eine Radardarstellung des aktuellen Verkehrs.

Die Darstellung des Verkehrs im Radardisplay wird periodisch aufgefrischt. Bei jedem dieser *updates* wird eine Repräsentation des aktuellen Zustandes der Verkehrssituation von der Simulationskomponente übertragen. Sie besteht aus einer Beschreibung der aktuellen Punkte im Raum für jedes Luftfahrzeug, den Prädiktoren[5] und Geschwindigkeiten der Luftfahrzeuge und der aktuellen Simulationszeit.

[5]Der Prädiktor (*velocity leader*) eines Luftfahrzeuges ist ein Geschwindigkeitsvektor auf Basis der aktuellen Bewegung, die aus den letzten beiden vom Radar gemessenen Positionen ermittelt wird (s. Abb. 60, S. 60).

Die horizontalen Positionen werden im Koordinatensystem des Radardisplays übertragen und können so direkt zur Darstellung benutzt werden. Jedes Luftfahrzeug hat in der Radardarstellung ein Label, in dem alphanumerische Informationen angezeigt werden. So werden z.B. Höhe und Geschwindigkeit direkt im Label dargestellt.

Für jedes Flugzeug wird innerhalb der Darstellungskomponente eine Datenstruktur verwaltet, in der die letzten horizontalen Positionen gespeichert werden. Auf dieser Basis kann eine Flugspur angezeigt werden, die die bisherige Bewegung des Luftfahrzeuges darstellt. Die geschätzte zukünftige Bewegung wird beim Update übertragen und kann direkt angezeigt werden. Die Radardarstellung kann parametrisiert werden, um die Länge der Prädiktoren und der Flugspur einzustellen.

Als Nachrichten erhält die Darstellungskomponente in erster Linie Informationen zum Auffrischen der Flugzeugpositionen. Alle anderen Nachrichten werden als Anweisungen zur Modifikationen der Darstellung interpretiert. So kann z.B. die Farbe der Darstellung eines Luftfahrzeuges oder die angezeigte Länge seines Prädiktors geändert werden.

Die Art der Darstellung unterscheidet sich für die beteiligten Benutzer. Der Lotsentrainee erhält ausschließlich ein Radardisplay. Die anderen Beteiligten haben zusätzlich ein Fenster, mit dem in die Simulation eingegriffen werden kann. Der Geisterpilot kann damit Eingriffe des Schülers umsetzen. Der Trainer kann außerdem Anweisungen an alle anderen angeschlossenen Radardisplays senden, mit denen die Art der Darstellung manipuliert werden kann.

Je nach Konfiguration kann von der Darstellungskomponente aus die Anweisung abgesetzt werden, die Simulation anzuhalten, zu beschleunigen oder zu verlangsamen.

3.2.4 Trainingsfunktionen

Wie in Abb. 3.2 gezeigt wird, sind zwei unterschiedliche Trainer-Akteure vorgesehen: Ein menschlicher Akteur und eine technische Komponente. Im Trainingssystem ist der Einsatz eines menschlichen Trainers *oder* der angepaßten Computersimulation MoFL möglich.

Der Einsatz eines simulierten Trainers ist natürlich ein ökonomisches Zugeständnis an die Lehrqualität im Vergleich zu einer Situation mit einem erfahrenen (menschlichen) Trainer. Deshalb ist in diesem Trainingssystem auch die Teilnahme eines menschlichen Trainers möglich.

Der Vorteil gegenüber der bestehenden Situation (s. Abschnitt 2.3.1) liegt dann z.B. in der möglichen räumlichen Trennung von Trainee und Trainer. Die Funktionen des Trainer-Kontrollfensters erlauben es, gezielt Flugzeuge unterschiedlich einzufärben, aufleuchten zu lassen oder eine multiple choice Frage zu stellen, die der Schüler durch Mausklick beantworten muß. Weitere Funktionen lassen sich leicht hinzufügen.

Mit diesen Funktionen kann der Trainer die Leistung des Schülers beobachten und in kritischen Situationen, in denen der Schüler nicht reagiert, die Aufmerksamkeit durch Farbsignale lenken.

Außerhalb der Schnittstelle des Trainers ist ein Abbruch der Simulation und ein Wiederaufsetzen zu einem anderen Zeitpunkt möglich. Durch die Verwendung eines *replays* des Schülers ist so ein einfaches Zurückspulen zu einer Situation, die einen späteren Konflikt verursachen wird, möglich. Dadurch können Trainer und Schüler gemeinsam iterativ an einer Lösung für einen Konflikt arbeiten und die Folgen beobachten. Dafür kann auch die Simulationsgeschwindigkeit an die Bedürfnisse angepaßt werden.

Das System unterstützt keine speziellen Kommunikationswerkzeuge zwischen Lehrer und Trainee, falls Trainer und Trainee räumlich getrennt sind. Werkzeuge wie Telefon oder Instant

Messaging können parallel zum Trainingssystem genutzt werden. Ihre Anwendung in das Trainingssystem zu integrieren ist nicht sinnvoll.

Bei einem Einsatz ohne menschlichen Trainer, sondern mit einer zusätzlichen Trainingskomponente wird der Entwurf aus Abschnitt 2.4 umgesetzt. Er legt weitgehend den Aufbau und die Funktionen dieser Komponente fest.

Als Basis für diese Komponente wird die Simulation von MoFL (s. Abschnitt 2.2.4) eingesetzt. Alle Funktionen, die einen Eingriff in die Verkehrssituation erzeugen, werden jedoch entfernt, denn der Eingriff soll vom Lotsenschüler vorgenommen werden. Daher muß eine Form der Nicht-Monotonie in MoFL integriert werden: Die Eingriffe des Schülers bedingen eine Änderung der Verkehrssituation. Bei einem Eingriff des Schülers müssen alle Elemente des Arbeitsgedächtnisses in MoFL, auf die sich der Eingriff (auch indirekt) auswirkt, gelöscht werden.

Bei der ursprünglichen Implementierung von MoFL (s. Abschnitt 2.2.4) liegt die Initiative zum Eingriff beim Modell selbst. Das bedeutet, daß alle Eingriffe von MoFL veranlaßt werden. Dadurch ist es nicht möglich, daß sich *picture* und Verkehrssituation auseinanderentwickeln. Im Trainingssystem liegt die Initiative zum Eingriff jedoch beim Lotsenschüler, bzw. Geisterpiloten. Hier müssen die Änderungen an MoFL weitergeleitet werden, weil sonst das *picture* von MoFL und die Verkehrssituation sich nicht mehr decken würden.

Während MoFL parallel zum Lerner arbeitet, werden Informationen über die Verkehrssituation aufgenommen, und das *picture* aufgebaut. Bei der Antizipation werden besondere Ereignisse oder Konstellationen erkannt und gespeichert. Ein neues zusätzliches Modul in MoFL sendet beim Erkennen besonderer Ereignisse entsprechende Anweisungen zur Modifikation der Darstellung an die angeschlossenen Radardisplays.

Die Vorteile beim Einsatz eines so modifizierten MoFLs sind sowohl ökonomisch, weil für diese Trainingseinheiten kein speziell ausgebildeter Trainer benötigt wird, als auch individuell, denn der Schüler kann selbst bestimmen, wann, wo und wie lange er trainieren möchte.

Ein Szenario, an dem sowohl ein menschlicher Trainer, als auch MoFL teilnehmen, kann sinnvoll sein. So kann der menschliche Trainer mehrere Sessions mit Lotsentrainees, die an unterschiedlichen Orten üben, gleichzeitig betreuen. Er würde das Training jeweils eines Schülers beobachten und gegebenfalls über Telefon oder Instant-Message korrigierend eingreifen, während der Schüler die meiste Zeit autonom mit MoFL trainiert. Da jedoch auch der Trainer ein *picture* von der gegebenen Verkehrssituation aufbauen und behalten muß, ist die Zahl der parallelen Sessions sehr begrenzt.

Ob ein Geisterpilot an einem Trainingsszenario teilnimmt, hängt von den äußeren Bedingungen ab. Eine wichtige Fertigkeit für Lotsen ist die Beherrschung des Sprechfunkverkehrs. Es werden spezielle Phrasen und Idiome benutzt. Die Verständigung fällt oft wegen der Verbindungsqualität schwer. Es kann also primäres Trainingsziel sein, die Kommunikation mit den Piloten zu üben. Dann ist der Einsatz eines Trainingsmoduls wie MoFL unwichtig, während die Teilnahme eines oder mehrer Geisterpiloten notwendig ist.

In anderen Situationen ist das Training der Kommunikation weniger wichtig. Es sollen dann besonders die Kontrollfertigkeiten geübt werden. In diesem Fall ist die Einbindung von MoFL vorteilhaft, während der Einsatz eines menschlichen Geisterpiloten entfallen kann. Der Schüler kann selbst mit dem Kontrollfenster des Geisterpiloten die Verkehrssimulation beeinflussen. Es werden ihm keine zusätzlichen Informationen angezeigt, die den Lernerfolg schmälern könnten.

In einer Trainingssituation mit einem menschlichen Trainer kann dieser natürlich die Rolle

des Geisterpiloten mitübernehmen. Deshalb ist das Kontrollfenster des Trainers eine erweiterte Version des Fensters des Geisterpiloten. Das setzt natürlich voraus, daß der Trainer nicht mit dem Verkehr und der Kontrolle mehrer Schüler überlastet wird.

3.3 Implementierung

Die Entwurfsideen werden im folgenden Abschnitt konkretisiert. Hier wird die Umsetzung des Entwurfes beschrieben und erörtert.

Zur Implementierung des Entwurfes wurden interpretierende Skriptsprachen verwendet. Für die Umsetzung der Anforderungen und des Entwurfes ergeben sich dadurch folgende Vorteile:

Portabilität: Die Portabilität des Programmcodes ist sehr hoch. Perl, Tcl/Tk, scheme und Python-Programme können unter günstigen Bedingungen ohne Änderungen oder neues Compilieren auf Windows-, UNIX- und MacOS-Systemen ablaufen.

Kurzer Programmierzyklus: Der Zyklus der Programmentwicklung Editieren — Compilieren — Linken — Testen verkürzt sich wesentlich durch das Wegfallen von Compilieren und Linken. Durch die dynamische Interpretierung des Codes ist eine interaktive Benutzung neuer Funktionen möglich. Es ergeben sich Vorteile für die Fehlersuche. Dadurch sind interpretierte Sprachen vorteilhaft für die Entwicklung von Prototypen.

Komponentenkopplung: Skriptsprachen bieten Konzepte zum Verknüpfen vorhandener Komponenten, so daß eine Integration vorhandener oder neuer Module innerhalb eines Prozesses ermöglicht wird.

Metaprogrammierung: Skriptsprachen erlauben die Benutzung von Metaprogrammierungskonzepten durch die dynamische Auswertung von Programmcode. Code wird entweder innerhalb des Prozesses oder von außen kommend im Kontext des laufenden Prozesses interpretiert und ausgeführt. Dadurch können Komponenten sich oder andere Teile dynamisch umkonfigurieren oder neuprogrammieren. Diese Technik wird in diesem System bei der Kopplung der Komponenten benutzt.

Parser vorhanden: Skriptinterpreter dienten ursprünglich der flexiblen Konfiguration großer Systeme. Sie können Dateien lesen und interpretieren, die im syntaktischen Format ihrer Sprache vorliegen. Dadurch wird kein neu zu entwickelnder Parser für ein proprietäres Format von Konfigurationsdateien benötigt. In diesem System werden deshalb Flugpläne oder Simulationsskripte in Perl-Notation beschrieben, so daß sie direkt von einem Perl-Programm eingelesen und verarbeitet werden können (s. Abschnitt 3.2.2, S. 46).

Durch die Interpretierung ergeben sich jedoch im Vergleich zu Compilaten Laufzeitverluste. Dies hat sich als kritisch für die Implementierung des Simulationsservers erwiesen. Skriptsprachen haben jedoch grundsätzlich eine Schnittstelle, über die compilierte Objekt-Dateien eingebunden werden können. So ließe sich der Nachteil durch Implementierung laufzeitkritischer Komponenten in einer maschinennahen Programmiersprache umgehen.

Die nötigen Funktionen werden auf unterschiedliche Komponenten verteilt, die jedoch zum Teil in demselben Prozeß ablaufen. Die Aufteilung in Pakete, Module und Komponenten ist in Abb. 3.6 dargestellt. Es gibt vier Pakete, die teilweise mit jeweils einer anderen Programmiersprache erstellt wurden.

1. Der *Simulationsserver* ist in Perl implementiert. In jedem Trainingsszenario gibt es genau einen Prozeß, in dem dieses Paket abläuft. Er beinhaltet drei Komponenten:

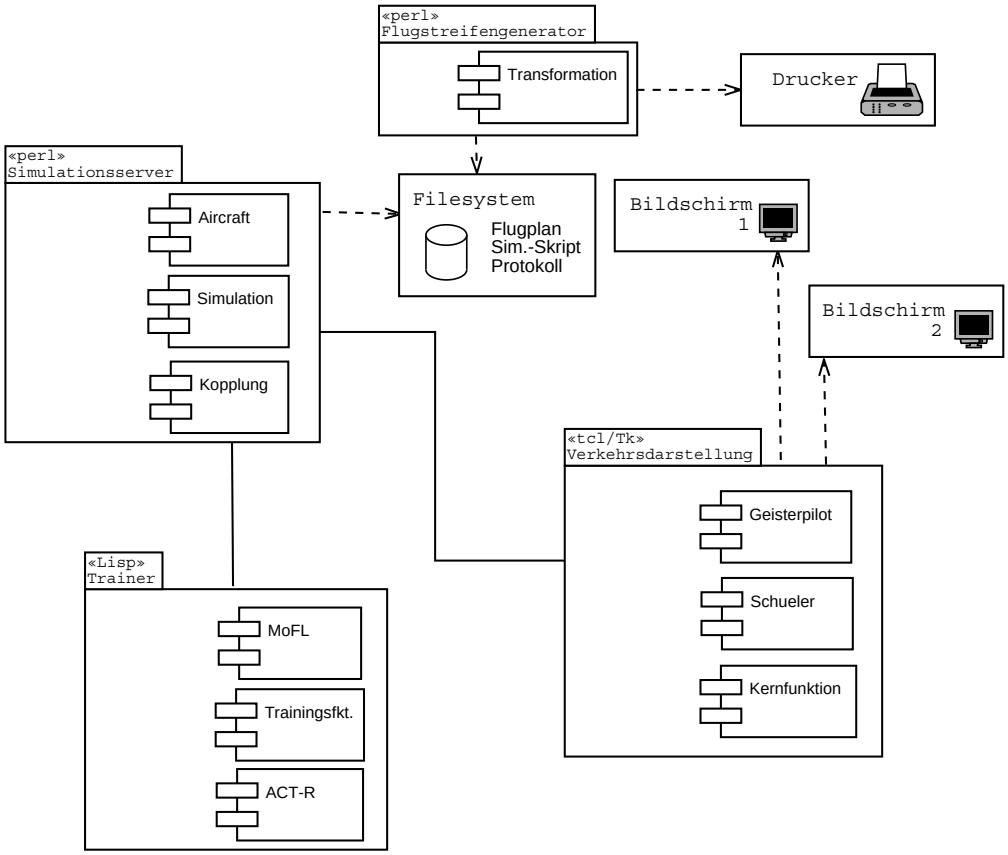

Abbildung 3.6: UML-System-Deployment-Diagramm für das Trainingssystem.

`Kopplung` Eine Komponente dient der Kopplung aller anderen Komponenten, die am Szenario beteiligt sind (s. Abschnitt Kopplung in 3.2.1).

`Simulation` Ein allgemeiner Simulationskern, der die in Abschnitt 3.2.2 beschriebenen Funktionen innerhalb des Rahmenkonzeptes benutzt.

`Aircraft` Eine Komponente, die diese Simulationsfunktionen für den konkreten Anwendungsfall Luftverkehr bereitstellt.

Das Paket benutzt das Filesystem, um gespeicherte Flugpläne und Simulationsskripte zu lesen und Eingriffe und damit die Grundlage für ein späteres Replay zu protokollieren.

2. Die *Verkehrsdarstellung* ist ein in Tcl/Tk programmiertes Paket, das in mehreren Prozessen auf unterschiedlichen Rechnern am Simulationsszenario teilnehmen kann. Es besteht aus mehreren Komponenten:

   `Kernfunktion` Eine Komponente stellt die Kernfunktionen zur Anzeige und zum Update eines Radarbildes in der Flugsicherung bereit.

   `Schueler`/`Geisterpilot` Komponenten, die in einem erweiterbaren Modulkonzept, das in Abschnitt 3.3.2 erörtert wird, umgesetzt werden, ergänzen das Paket um spezielle Sichten und Funktionen für unterschiedliche Benutzer. So gibt es z.B. eine Komponente, die die Anzeige für den Lotsenschüler (`Schueler`) konfiguriert, eine andere fügt Funktionen und grafische Elemente für einen Geisterpilotenarbeitsplatz (`Geisterpilot`) hinzu.

   Zur Darstellung wird für jeden Prozeß, der dieses Paket instantiiert, ein Bildschirm, bzw. eine grafische Oberfläche, wie ein X-Server oder die Windows-Oberfläche, benötigt, auf dem die Fenster erzeugt werden.

3. Der *Flugstreifengenerator*, der als Perl-Prozeß instantiiert wird, nimmt nicht „on-line“ an einem Szenario teil. Mit ihm werden die Fluginformationen vor dem Trainingsdurchlauf eines Szenarios zu PostScript-Dateien transformiert, die dann für das Szenario als Papierflugstreifen ausgedruckt werden können.

4. Das *Trainer*-Paket kommuniziert über die Kopplungskomponente des Simulationsservers mit der Verkehrsdarstellung. Es läuft in einem eigenen Prozeß des Trainingssystems auf einem beliebigen angeschlossenen Rechner. Es besteht aus drei Komponenten:

   `MoFL` Das kognitive Modell der Fluglotsenleistungen MoFL, das durch die Simulation kognitiver Vorgänge eine Repräsentation (*picture*) der Relevanz der Objekte und Konstellationen in der aktuellen Verkehrssituation aufbaut.

   `Trainingsfkt` Spezifische Trainingsfunktionen, die aufgrund des Zustandes des *pictures* die Darstellung in den Anzeigeeinheiten ändern.

   `ACT-R` Diese beiden Komponenten sind in dem auf CommonLISP aufbauenden ACT-R LISP-Modul programmiert, so daß ACT-R als weitere Komponente in diesem Paket dargestellt wird.

In den folgenden Abschnitten werden die vorgestellten Pakete genauer beschrieben.

3.3.1 Simulationsserver

Der Simulationsserver übernimmt zwei Funktionen im Trainingssystem: Er schreibt die Trajektorien der simulierten Luftfahrzeuge fort und koppelt die anderen Komponenten.

Kopplung

Die Kopplung der beschriebenen Pakete erfolgt über einen Interprozeß-Kommunikationsmechanismus. Dafür werden Sockets benutzt, über die Prozesse auf unterschiedlichen oder gleichen Rechnern miteinander verbunden werden können. Als Kommunikationsprotokoll wird das verbindungsorientierte Internetprotokoll TCP benutzt. TCP erzeugt in den übertragenen Datenpaketen automatisch Sicherungsinformationen, die vom Empfänger ausgewertet werden. Im Fehlerfall fordert der Empfänger automatisch ein erneutes Senden des betroffenen Paketes an. Dadurch ist sichergestellt, daß der Empfang der Datenpakete in der richtigen Reihenfolge und fehlerlos passiert.

Die generierten Sicherungsinformationen verringern den Durchsatz an Nutzdaten. Doch für die Kommunikation der Prozesse im Trainingssystem ist eine fehlergesicherte Übertragung zwingend erforderlich, so daß bei der Verwendung eines effizienteren aber nicht fehlergesicherten Protokolls wie UDP eigene Fehlerkorrektur-Mechanismen implementiert werden müßten, die den Nutzdatendurchsatz genauso wie bei TCP verringern würden. Lediglich der Betrieb innerhalb eines begrenzten lokalen Netzwerkes würde eine Fehlersicherung überflüssig machen. Da das Trainingssystem jedoch auch zum Fernlernen eingesetzt werden soll, kann auf TCP nicht verzichtet werden.

Innerhalb der Client/Server-Rahmenarchitektur wird vom Kopplungsserver ein Netzwerkdienst implementiert, mit dem sich beliebig viele Clients gleichzeitig verbinden können, indem sie eine TCP-Socket-Verbindung zu einem spezifizierten Port auf dem Rechner aufbauen, auf dem der Kopplungsserver-Prozeß läuft.

Es gibt unterschiedliche Modelle, wie eine Client/Server-Verbindung aufgebaut werden kann. Bei den meisten Internet-Dienst-Servern wird z.B. für jeden neuen Client ein eigener Kindprozeß abgespalten, der ausschließlich für die Kommunikation mit seinem Client verantwortlich ist, so daß der Elternprozeß weiter auf neue Verbindungswünsche reagieren kann. Da die Serverprozesse jedoch alle in unterschiedlichen Namensräumen laufen, ist eine Kopplung der Clients über den Serverprozeß nicht möglich.

Zur Verbindung mehrerer Clients untereinander über einen Server kann dagegen ein anderes Konzept benutzt werden. Der Server-Prozeß implementiert eine Verarbeitungsschleife, in der nacheinander alle Kommunikationskanäle auf neue Nachrichten untersucht werden, die dann gelesen und verarbeitet werden. Zur effizienten Implementierung dieses Prinzips dient der UNIX-Systemaufruf select. Mit ihm kann eine Reihe von Filedeskriptoren (bei Socketverbindungen ist die Schnittstelle analog zu der von Dateien) auf das Anliegen neuer Nachrichten oder die Möglichkeit Zeichen zu senden untersucht werden. Eine Rückkehr aus diesem Systemaufruf hat den Deskriptor, auf dem solch ein Ereignis anliegt, zum Ergebnis.

Der select-Systemaufruf kann also im Kopplungsserver dazu benutzt werden, alle Socketdeskriptoren der Clients auf das Anliegen neuer Nachrichten zu überwachen.

Mit dieser Methode werden nun alle sich verbindenden Clients über einen Internet-Dienst-Server verkoppelt. In einer Schleife werden alle verbundenen Sockets mit dem select-Systemaufruf auf neue Nachrichten hin untersucht. Liegt eine neue Nachricht an, wird sie gelesen und in einer

Fallunterscheidung analysiert und verarbeitet.

Die Analyse der gelesenen Nachrichten erfolgt über einfache reguläre Ausdrücke[6]. Es gibt folgende vorgesehene Nachrichtenarten, die von der Kopplungskomponente des Simulationsservers verarbeitet werden:

register: Alle an dem Simulationsszenario teilnehmenden Agenten identifizieren sich über einen Namen, der in jeder Nachricht als Identifikation benutzt wird. Der Name, unter dem ein Agent angemeldet wird, wird über die *register*-Nachricht versandt. Da im Moment nur Namens-Klassen vorgesehen sind, müssen die Namen nicht eindeutig sein. Deshalb gibt es kein Konzept, schon vergebene Namen nachzuverhandeln.

Wenn eindeutige Namen für die Clients nötig wären, müßte der Server bei einem Verbindungswunsch eines Clients, der mit *register* einen Namen wünscht, der schon im Szenario vergeben ist, eine Nachverhandlung initiieren: Der Client sucht dann einen neuen Namen und bietet ihn wieder mit *register* an.

Folgende Namensklassen werden im Moment benutzt:

⇒ **ATCO** Anzeigeeinheit eines Lotsenschülers

⇒ **GHOST** Anzeigeeinheit eines Geisterpiloten

⇒ **TRAINER** Anzeigeeinheit eines erweiterten Geisterpilotenarbeitsplatzes (Trainer-Arbeitsplatz)

⇒ **MOFL** das um Trainingsfunktionen erweiterte kognitive Modell der Fluglotsenleistungen

⇒ **RADAR** der Simulationsserver.

Die Verwendung von eindeutigen Instanznamen ist in diesem System nicht nötig, weil die Namen nur dazu verwendet werden, einige Clients vom Empfang auszuschließen, wenn die Nachricht für sie irrelevant ist.

simspeed: Die Skalierung der Zeit wird für die Simulation geändert (s. Abschnitt 3.1.2, S. 36).

flush: Der Client erbittet das Senden einer leeren Nachricht. Das ist nötig, um zeitweise auftretende Verklemmungen zwischen den kommunizierenden Agenten aufzulösen. Zu diesen Verklemmungen kann es kommen, wenn ein Client „blockierendes Lesen“ von der Socketverbindung benutzt, sein Prozeß also so lange anhält, bis ein Zeichen bzw. eine komplette Nachricht vom Server gesendet wurde.

log: Der Client erbittet die Protokollierung eines Textes. Protokolliert werden Nachricht und Zeitpunkt.

quit: Der Client beendet die Verbindung zur Kopplungskomponente des Simulationsservers. Das Empfangen dieser Nachricht bewirkt das Löschen des Clients aus internen Datenstrukturen. Ist dieser Vorgang abgeschlossen, wird dem Agenten eine Nachricht zurückgesendet, in der das erfolgreiche Abmelden bestätigt wird. Der Agent kann nun auf diese Antwort mit seiner Terminierung reagieren.

[6] Reguläre Ausdrücke sind (i.A. sehr kleine) Grammatiken, mit denen überprüft werden kann, ob eine Zeichenkette ein legales Wort einer regulären Sprache ist. Diese Grammatiken werden mit einem einfachen Formalismus notiert.

action: Der Client erbittet einen Eingriff in den Verkehrsablauf. Flugparameter für ein über sein Rufzeichen zu identifizierendes Flugzeug werden in der Simulation geändert. Dieser Nachrichtentyp wird nur von den Agentenklassen GHOST und TRAINER gesendet. Die Abarbeitung dieser Nachrichten simuliert ein Umsetzen einer Lotsenanweisung durch einen Piloten im angesprochenen Flugzeug.

⇒ **heading:** Die Flugrichtung eines Luftfahrzeuges soll geändert werden. Die Richtung wird als Winkel angegeben. Dabei ist die Zuordnung: Norden: 0°, Osten: 90°, Süden: 180°, Westen: 270°.

⇒ **level:** Die Flughöhe soll geändert werden. Sie wird in Flugflächen (*flight level*, FL) angegeben. 1 FL = 100 ft = 30,48 m.

⇒ **speed:** Die Fluggeschwindigkeit soll geändert werden. Sie wird in Knoten (kt) angegeben. 1 kt = 1 nautische Meilen/h = 1,852 km/h

ask-1: Diese Nachricht wird von MoFL gesendet, um aktuelle Werte bestimmter Parameter eines über das Rufzeichen identifizierten Flugzeuges zu erfragen. Es wird eine entsprechende Antwort zurückgesendet.

goodby: Diese Nachricht wird von GHOST oder TRAINER gesendet und an MoFL weitergegeben. Die Nachricht beinhaltet das Rufzeichen eines Flugzeuges, das gerade den Sektor verlassen hat. MoFL soll es aus seinem *picture* löschen.

Wird eine Nachricht empfangen, die nicht von einem der oben dargestellten Typen ist, wird sie an alle anderen angeschlossenen Clients weitergeleitet. Dadurch können auch Clients untereinander Nachrichten austauschen. Das wird im Trainingssystem z.B. dafür benutzt, von Trainings-Komponenten wie MoFL spezielle Darstellungsanweisungen an die Anzeigeeinheit des Lotsenschülers zu senden.

Alle empfangenen Nachrichten werden protokolliert.

Simulation

Die Simulationskomponente ist generisch angelegt: Das Konzept aus Abschnitt 3.2.2 wird von einem allgemeinen Kern implementiert, der die konkreten Funktionen über das auf S. 44 beschriebene Interface in einem flugverkehrsspezifischen Modul aufruft. Es könnte z.B. durch ein Simulationsmodul für ein Kraftwerk ausgetauscht werden. Lediglich die Darstellung müßte geändert werden. Das Rahmenkonzept des Simulationspaketes könnte unverändert beibehalten werden.

Das Speicherformat der Simulationsskripte, also ihre textuelle Repräsentation, ist in der Syntax von Perl gehalten. Dadurch wird zum Einlesen kein eigener Parser benötigt. Die in Abschnitt 3.2.2 spezifizierte Struktur des Simulationsskriptes bzw. der Flugpläne wird in Perl-Hash-Objekten repräsentiert. Dies ist der in Perl übliche Weg, Datenstrukturen bzw. Objekte zu implementieren. Aus dieser Umsetzung ergibt sich eine spezifische Perl-Struktur, die in dieser Form zur Notation der Flugpläne benutzt wird. Deshalb muß die eingelesene Datenstruktur auch nicht transformiert oder interpretiert werden, sondern kann direkt als Perl-Code ausgeführt werden, der dann die entsprechende Datenstruktur im Datensegment des Perl-Prozesses erzeugt. Auf diese Struktur greifen dann später die im Entwurf spezifizierten Methoden zu.

Bei jedem Eingriff über die Geisterpiloten-Schnittstelle wird der Flugplan durch Hinzufügen neuer Segmente geändert. Das geänderte Simulationsskript wird in eine Datei geschrieben. Bei

einem späteren Aufruf der Simulation kann dieser Ablaufplan statt des ursprünglichen Simulationsskriptes eingelesen werden. Die Simulation der resultierenden Flugbahnen ist das *replay* des ursprünglichen Verkehrs mit allen erfolgten Eingriffen.

Bei der Implementierung der Simulationsfunktionen wurde zuerst ein dafür geeignetes Hilfspaket benutzt. Es stellte sich aber heraus, daß das resultierende Programm nicht echtzeitfähig war. Daraufhin wurden entsprechende Mechanismen neu programmiert, so daß das resultierende Simulationssystem unter Realzeitbedingungen benutzt werden kann.

Im folgenden Abschnitt wird zuerst die ursprüngliche Implementierung vorgestellt. Dann wird ein Modell der internen Vorgänge erläutert, das begründet, warum keine Echtzeitverarbeitung erfolgen konnte. Schließlich wird der neue Mechanismus zur Sicherung der Realzeitfähigkeit präsentiert.

Die Implementierung des allgemeinen Simulationskerns geschah ursprünglich auf der Basis des EventServer-Moduls[7] für Perl. Zur Verarbeitung der Kopplungsanforderungen wurden zwei Arten von I/O-Ereignissen mit entsprechenden Callbackfunktionen zum Akzeptieren neuer Socketverbindungen und zum Lesen aus geöffneten Sockets definiert. Das Update der Simulation und das entsprechende Aussenden an angeschlossene Clients erfolgte über Timer-Events. Ein Nachteil der Timer-Ereignisse für das EventServer-Modul ist, daß sie sich nur für ganze Sekunden-Intervalle registrieren lassen. Es ist also z.B. nicht möglich, zyklisch alle 2,5 Sekunden eine Aktion auszuführen.

Die Überprüfung der Eigenschaften der so implementierten Simulation ergab Mängel im Echtzeitverhalten. Die in Abb. 3.7 dargestelleten Kurven beschreiben das Verhältnis von interner simulierter Zeit zu externer realer Zeit. Echtzeitfähigkeit bedingt ein konstantes Verhältnis zwischen diesen Größen. Im vorliegenden Fall hätte das Verhältnis konstant 2 betragen sollen, weil eine Beschleunigungs-Skalierung um den Faktor 2 gewählt war.

Die dargestellten Mängel lassen auf ein Überlaufen der Eventqueue schließen. Es werden zu bestimmten Zeitpunkten Ereignisse erzeugt und in der Warteschlange gespeichert. Die Rate der Erzeugung ist höher als die Abarbeitungsrate. Daher füllt sich die Warteschlange langsam. Die Simulation eines detaillierten Modells dieses Vorgangs ist in der Grafik 3.7 eingetragen. Die Beschreibung dieses Modells ist in Abb. 3.8 grafisch illustriert. Es gibt mehrere Erzeuger für Ereignisse, die zeitgesteuert sind. Die Erzeugung geschieht mit unterschiedlichen Raten, die durch die Definition dieser Ereigniserzeuger entstehen:

$$Erzeugungsrate = \begin{cases} SocketMgr: & \frac{1}{10} \\ Radar: & \frac{1}{5} \\ I/O: & 1 \\ simStep: & 1 \end{cases} \quad \text{Ereignisse pro Zeiteinheit}$$

Die Abarbeitungsfähigkeit ist für die unterschiedlichen Ereignisse ungleich, weil die zuge-

[7] Ein Erweiterungsmodul für Perl von Jack Shirazi. Es stellt eine Ereignisverarbeitungsschleife zur Verfügung. Ereignisse werden mit Callback-Routinen behandelt. Es können Callbacks für unterschiedliche Typen von Ereignissen installiert werden: Timer-, File- und Signal-Ereignisse sind möglich. Verfügbar über CPAN (*comprehensive perl archive network*): `http://www.cpan.org/modules/by-module/EventServer/EventServer-2.3.tar.gz`.

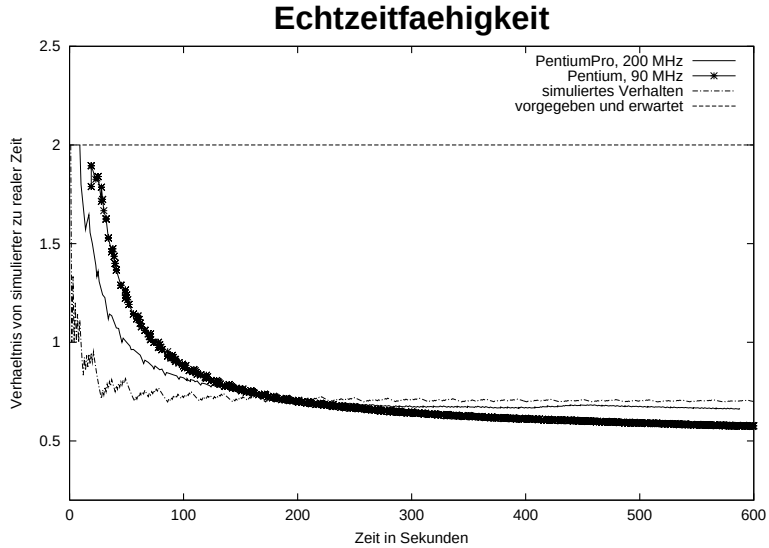

Abbildung 3.7: Darstellung des Zeitverhaltens der Simulation eines Verkehrsszenarios mit unterschiedlichen Rechnern (beide unter Linux 2.2). Vorgegeben und erwartet war ein konstantes Zeitverhältnis von 2. Zum Vergleich ist das simulierte Zeitverhalten des Verarbeitungsmodells (s. S. 56) dargestellt.

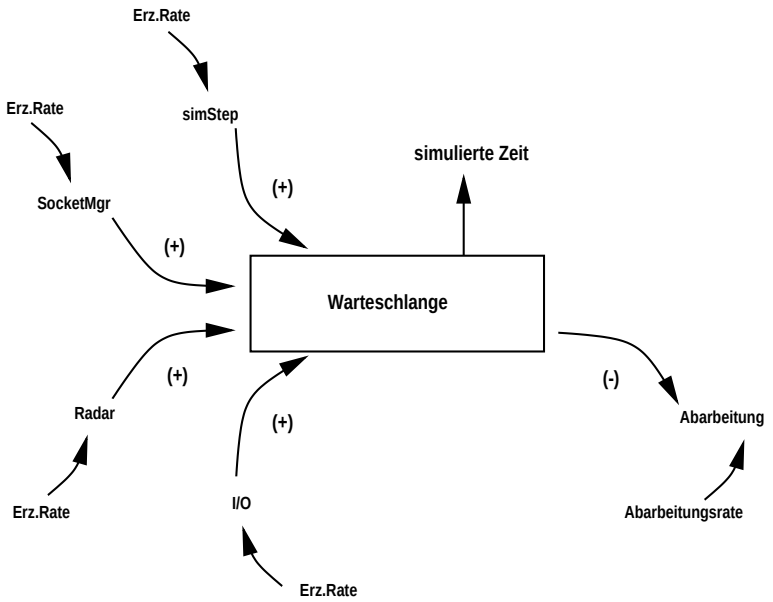

Abbildung 3.8: Modell der Verarbeitung der Ereignis-Warteschlange bei der Benutzung des EventServer-Moduls.

ordneten Callback-Funktionen einen unterschiedlichen Abarbeitungsaufwand haben.

$$Abarbeitungsaufwand = \begin{cases} SocketMgr: & 14 \\ Radar: & 6 \\ I/O: & 5 \\ simStep: & 1 \end{cases} \text{ Zeiteinheiten}$$

Die unterschiedlichen Abarbeitungsraten wurden durch Profiling des Laufzeitverhaltens des Perl-Prozesses geschätzt. Die Erzeugungsrate wird durch Multiplikation mit dem Abarbeitungsaufwand normiert. Es ergibt sich dann eine Erzeugungsrate von 8,6 Ereignissen pro Zeiteinheit. In jedem Arbeitsschritt der Ereignisverarbeitungsschleife werden in diesem Modell 3 Ereignisse abgearbeitet. Die Länge der Warteschlange nach n Zeiteinheiten ist auf $5,6 \times n$ Ereignisse, die auf Abarbeitung warten, angewachsen.

Bei jeder Abarbeitung eines simStep-Ereignisses wird die interne, simulierte Zeit um zwei Zeiteinheiten erhöht. Bei optimaler, das heißt rechtzeitiger Abarbeitung der Ereignis-Warteschlange wird die interne Zeit also in jeder Zeiteinheit um 2 erhöht. Es ergibt sich ein konstantes Verhältnis von 2 zwischen interner simulierter und realer Zeit.

Da die Warteschlange jedoch überläuft, können nicht alle Ereignisse verarbeitet werden. Durch die regelmäßige Erzeugung von Ereignissen sind in einer Warteschlange der Länge l im Mittel $\frac{10}{86} \times l$ simStep Ereignisse enthalten. Würde die gesamte Schlange abgearbeitet, was bei Realzeitfähigkeit passieren müßte, würde die interne Zeit um $\frac{10}{86} \times 2l$ Zeiteinheiten erhöht werden. Die Länge der Warteschlange ist also ein Maß für die Verspätung beim Heraufsetzen der internen Zeit. Die Verspätung zum Simulationszeitpunkt t_n, also nach n Ereignisabarbeitungsschritten ist $\frac{10}{86} \times 2 \times 5,6n = \frac{112}{86}n$. Da jedoch $2n = \frac{172}{86}n$ Zeiteinheiten vergehen sollten, kann als realer Simulationsfortschritt nur von $\frac{172-112}{86} = 0,7n$ ausgegangen werden. Es ergibt sich also unter diesen Bedingungen ein konstantes Verhältnis von 0,7 zwischen simulierter und realer Zeit.

Da die Verteilung der unterschiedlichen Ereignisse nicht gleichmäßig ist und die Warteschlange aufgrund der Normierung der Ereignisabarbeitungszeiten nicht gleichmäßig anwächst, kommt es zuerst zu einem langsameren Anstieg der Verspätung, wodurch das Verhältnis von simulierter zu realer Zeit von zuerst 2 auf 0,7 absinkt und dann stabil bleibt.

Die Verwendung dieses Modells läßt also die echtzeitfähige Simulation von Flugverkehr in diesem Skalierungsrahmen nicht zu. Dazu kommt, daß die Standard-Perl-Distribution und damit auch das EventServer-Modul nicht den Umgang mit Zeiteinheiten zuläßt, die kleiner als 1 Sekunde sind.

Deshalb mußte eine eigene Ereignisverarbeitung entworfen und umgesetzt werden, mit der ein Überlauf der Ereigniswarteschlange aufgefangen werden kann. Dazu gibt es eine Hauptschleife, in der die aktuelle Systemzeit überprüft wird.

Zum Zugriff auf Mikrosekunden, die im UNIX- und Windows-Kernel verfügbar sind, wurde das Time::HiRes-Paket[8] für Perl benutzt.

Durch einen Mechanismus zur Registrierung von Callback-Funktionen, bei dem eine Zyklusdauer-Spezifikation angegeben wird, kann die Ereignisverarbeitungsschleife entscheiden,

[8] Das Time::HiRes-Paket für Perl von Douglas E. Wegscheid implementiert Ersatzfunktionen für zeitbezogene Perl-Operationen, die statt Sekunden- Mikrosekunden-Auflösung bieten. Es ist über CPAN (*comprehensive perl archive network*) erhältlich: `http://www.cpan.org/modules/by-module/Time/Time-HiRes-01.17.tar.gz`.

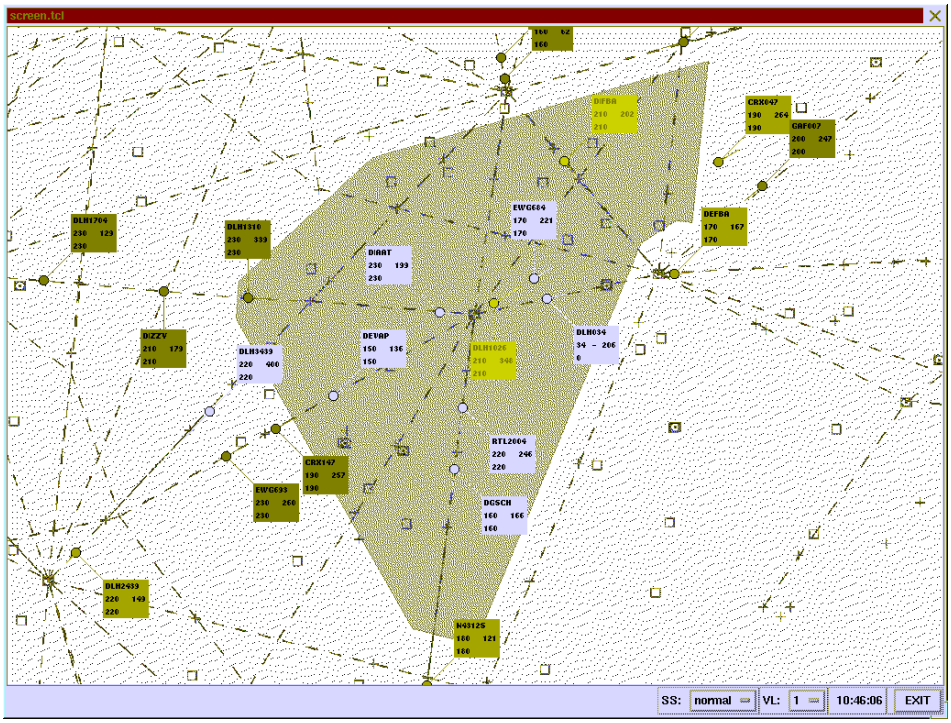

Abbildung 3.9: Simuliertes Radarbild in der Anzeigeeinheit des Trainingssystems. Die unterschiedlich eingefärbten Label signalisieren die aktuelle Bedeutung, die der Trainer annimmt.

welche Funktion zu einem Zeitpunkt ausgeführt werden soll. Dazu wird in einer Datenstruktur vermerkt, wann jede Funktion das letzte mal mit diesem Mechanismus aufgerufen wurde.

Bei einem Aufruf ist aus der Callback-Funktion die Zeitdauer seit dem letzten Aufruf aus dieser Datenstruktur zugreifbar. Die Auflösung liegt wegen der Verwendung des Time::HiRes-Paketes im Mikrosekunden-Bereich. Einige der so aufgerufenen Funktionen benutzen die Dauer seit ihrem letzten Aufruf, um ihr Verhalten zu modifizieren.

Insbesondere die simStep-Funktion benutzt diese Dauer, um die interne Simulationszeit und damit die Simulationsschrittweite an den Verlauf der realen Zeit anzupassen. Durch dieses Konzept kann Echtzeit-Verarbeitung sichergestellt werden. Dabei ist dann die Simulationsschrittweite und die Dauer zwischen zwei Updates bei den beteiligten Komponenten variabel und an die Zeitbedürfnisse innerhalb des Simulationskerns angepaßt.

Bei einem Update-Zyklus von etwa sieben bis neun Sekunden kann das beschriebene System echtzeitfähig auf einem Computer der Pentium-Klasse mit 200 MHz betrieben werden.

3.3.2 Verkehrsdarstellung

Das Paket Verkehrsdarstellung stellt ein frei konfigurierbares Anzeigesystem zur Verfügung. Es ist die Schnittstelle des Trainingssystems zu Schüler und Trainer. Es besteht aus unterschiedlichen Teilen, die getrennte Fenster auf verteilten Displays benutzen.

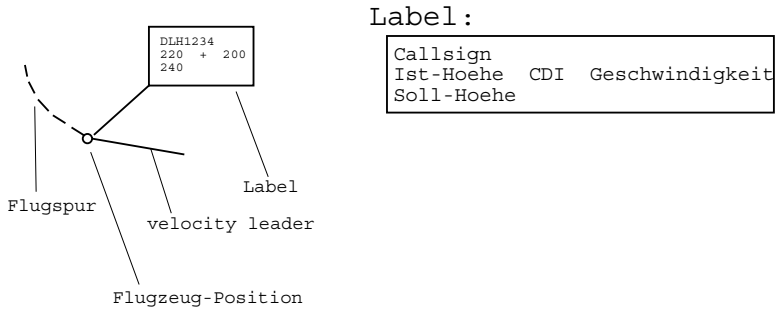

Abbildung 3.10: Darstellung der dynamischen Objekte im Radarbild: Die Flugzeug-Position wird mit einem Kreis dargestellt, im Label werden wichtige Parameter dieses Luftfahrzeuges angezeigt. *Velocity leader* (Prädiktor) und Flugspur zeigen den vorhergesagten, bzw. aufgezeichneten Flugweg an. Der *climb-descent-indicator* (CDI) gibt an, ob das Flugzeug im Steig-, Level- oder Sinkflug ist.

Radarbildschirm

Der Haupteil ist die Simulation des Radarbildes (s. Abb. 3.9). Hier werden die geografische Lage des zu kontrollierenden Sektors, die Funkfeuer und die Flugrouten statisch dargestellt. Der aktuelle Zustand der Simulation des Flugverkehrs wird dynamisch dargestellt. Für jedes Luftfahrzeug wird ein Punkt mit dazugehörendem *Label* angezeigt. Es enthält wichtige Informationen zum Zustand des Flugzeugs, z.B. Höhe und Geschwindigkeit (s. Abb. 3.10).

Die Implementierung aller Anzeigesysteme erfolgt mit der Skriptsprache Tcl/Tk. Die Verwendung der LISP-ähnlichen Sprache Tcl (*tool command language*) ermöglicht es, dynamische Aspekte der Anzeige durch Metaprogrammierung leicht umzusetzen. Das dazugehörende Tk (*toolkit*) erleichtert die Entwicklung grafischer Benutzungsoberfächen. Tcl/Tk-Programme sind ohne Änderungen unter Windows, MacOS und den meisten UNIX-Derivaten ablauffähig. Sie werden interpretiert.

Das Radarbild ist als *canvas*-Widget[9] umgesetzt. Innerhalb des Canvas' werden alle statischen Informationen als Hintergrundbild angezeigt. Das Sektorbild mit den Flugrouten und Funkfeuern ist also ein vorgefertigtes Bild. Die dynamischen Objekte, also die Punkte der Luftfahrzeuge mit den dazugehörenden Labels (s. Abb 3.10), werden mit Kommandos des Canvas' erzeugt und bewegt. Dazu kommen z.B. die tk-Komandos *window*, *oval* und *line* zum Einsatz.

Damit die Maskierung (s. Abschnitt 3.1.2) einfach implementiert werden konnte, werden die alphanumerischen Werte der Labels nicht direkt in das Canvas-Widget geschrieben, wozu die Canvas-Funktion *text* zur Verfügung stünde. Stattdessen werden neue Fenster-Widgets erzeugt, die wiederum *label*-Widgets enthalten, und im Canvas angezeigt. Dadurch läßt sich einfach auf die Attribute des Textes zugreifen, und es lassen sich auf einfache Weise Eventhandler definieren. Bei aktivierter Maskierung werden Vorder- und Hintergrundfarbe gleichgesetzt und entsprechende Eventhandler erzeugt, die bei Überfahren mit der Maus die Farben ändern und Protokolleinträge generieren.

Das Label ist in 45°-Schritten drehbar. Bei jedem Simulationsupdate werden die Punkte entsprechend der simulierten Bewegung im Canvas verschoben.

In einer Leiste unten im Radarfenster können unterschiedliche Optionen geändert werden (s. Abb. 3.11). Die Auswahl der Optionen ist bei Geisterpilot, Trainer und Trainee unterschiedlich.

[9] *widget* = *windowing gadget*: grafisches Bedienelement, z.B. Button oder Menü.

Abbildung 3.11: Alle möglichen Optionen des simulierten Radarbildes

Sie erfolgt durch Konfiguration in den unterschiedlichen eingebundenden Modulen (s. Unterabschnitt Modularisierung, S. 62).

LO (label orientation): Hier wird der Winkel eingestellt, in dem das Label eines neu dargestellten Luftfahrzeuges angezeigt wird. Es gibt acht Wahlmöglichkeiten. Eine Änderung wirkt sich nur auf die zukünftig einfliegenden Flugzeuge aus.

VL (velocity leader length): Aufgrund der bisherigen Bewegung eines Luftfahrzeuges kann ermittelt werden, in welche Richtung und wie schnell es sich bewegt. Radaranzeigen haben die Option, diese Schätzung mit einem sogenannten *velocity leader* anzuzeigen. Das ist ein Vektor, der anzeigt, an welchem Punkt sich das Flugzeug in x Minuten befinden wird. Mit dieser Option ist x wählbar. Mögliche Werte sind Null bis Neun Minuten. Eine Änderung wirkt sich auf alle dargestellten *velocity leaders* gleichzeitig aus.

TL (trace length): Analog zur Prädiktion der Bewegung mit dem *velocity leader* gibt es eine Historie der Bewegung, die mit der *trace* angezeigt wird. Auch hier gibt es die Wahl, wie weit aus der Vergangenheit die *trace* dargestellt wird. Werte und Verhalten sind analog zu VL.

SS (simulation speed): Während die bisher vorgestellten Optionen nur die Anzeige betreffen, kann mit dieser Option die Simulation selbst beeinflußt werden. Hier kann eine von fünf unterschiedlichen Skalierungen der Simulationsgeschwindigkeit gewählt werden (s. Abschnitt 3.1.2). Die Auswahl wird an den Simulationsserver gesendet und wirkt sich dort sofort aus. Die Skalierungen sind 0 (*freeze*), $\frac{1}{2}$ (halbe Geschwindigkeit), 1 (normal), 2 (doppelte Geschwindigkeit) und 4 (vierfache Geschwindigkeit).

Zusätzlich wird in einem Feld dieser Leiste die vom Simulationsserver übermittelte Simulationszeit angezeigt. Sie wird mit jedem Positionsupdate mitübertragen und bezieht sich auf Zeitpunkte im Simulationsskript, bzw. in den Flugplänen.

Mit dem „EXIT“-Button wird der Prozeß, der die Anzeigeeinheit darstellt, vom Simulationsserver abgemeldet und beendet.

Geisterpiloten-Modul

Der Lotsenschüler gibt an eines der kontrollierten Flugzeuge per simuliertem Sprechfunk eine Anweisung, die der Geisterpilot mit diesem Modul umsetzt, indem entsprechende Kommandos an den Simulationsserver gesendet werden. Die Anweisungen beziehen sich immer auf ein Luftfahrzeug, das über sein Rufzeichen (*callsign*) identifiziert wird.

Die Anzeigeeinheit für den Geisterpiloten, bzw. Trainer ist aus dem des Lotsenschülers durch eine andere Instantiierung der Module (s. Unterabschnitt Modularisierung, S. 62) abgeleitet.

Abbildung 3.12: Geisterpilotenmodul in der Anzeigeeinheit des Trainingssystems: Für das Luftfahrzeug mit dem Rufzeichen DLH1026 wird die Anweisung gegeben, die Höhe auf FL 230, also 23.000 Fuß, zu ändern.

Ebenso wie bei dem Trainee wird das Radarbild mit der Optionsleiste in einem Fenster angezeigt. Die möglichen Optionen sind „SS“ (simulation speed) und „VL“ (velocity leader length).

Zusätzlich wird in einem zweiten Fenster ein Dialog zum Eingriff in die Simulation dargestellt (s. Abb. 3.12).

Im oberen Teil dieses Fenster kann mit einem Menübutton eines der dargestellten Flugzeuge gewählt werden. Beim Update wird u.a. dafür eine Liste aller darzustellenden Luftfahrzeuge verwaltet, die in diesem Menübutton angezeigt werden.

Für das gewählte Luftfahrzeug kann dann entweder eine Änderung der Flughöhe, der Geschwindigkeit oder der Richtung (*heading*) veranlaßt werden, indem in ein Eingabefeld der gewünschte Wert eingetragen wird, und der nebenstehende mit der Anweisung beschriftete Button aktiviert wird.

Der „goodby“-Button hat die Funktion, ein Flugzeug aus einem kontrollierten Sektor zu verabschieden. Diese Verabschiedung wird für den Einsatz zusammen mit dem simulierten kognitiven Modell MoFL benötigt, da dort noch kein geographisches Wissen über den Sektor eingebunden ist. Der Geisterpilot oder Trainer muß deshalb die Flugzeuge „per Hand“ verabschieden.

Eine Variante dieses Dialogfensters für den Trainerarbeitsplatz enthält zusätzlich drei prototypische Trainingsfunktionen, mit denen der Trainer ein Luftfahrzeug im Radarbild der angeschlossenen Schüler kurz in einer anderen Farbe aufleuchten lassen oder dauerhaft eine andere Farbe zuweisen kann. Mit dem „ask“-Button wird ein modaler Dialog bei allen angeschlossenen Trainee-Radarbildern erzeugt, der eine Frage und zwei Antwortalternativen enthält. Die Antworten werden an den Simulationsserver zur Protokollierung geschickt.

Modularisierung

Die oben beschriebenen unterschiedlichen Ausprägungen des Anzeigesystems werden durch ein Modularisierungskonzept erreicht. Es gibt ein Tcl/Tk-Programm, das alle Basis-Funktionen und das Radarbild zur Verfügung stellt. Durch Kommandozeilenparameter können Module bestimmt werden, die vom Kernprogramm nachgeladen werden sollen.

Dabei wird Tcl/Tk-Code aus einer angegebenen Datei gelesen und im selben Namensraum wie der des Kerns ausgeführt. Durch die Flexibilität von Tcl/Tk lassen sich so einfach Parameter initialisieren, neue Funktionen hinzufügen und die Darstellung der Oberfläche ändern. Die Möglichkeiten reichen also von einfachen Konfigurationsaufgaben, über Erweiterungen der Funktionalität bis zur Änderung der Fensterdarstellung.

Die momentane Aufteilung der Funktionen ist in Abbildung 3.13 dargestellt. Die Darstellung erfolgt als UML-Klassendiagramm mit Spezialisierungsbeziehungen, weil der vorgestellte Mechanismus Vererbung nachbildet. Es sind Erweiterungen und Überladungen möglich. Die folgenden „Klassen“ modularisieren das Anzeigesystem:

server: In diesem Modul werden Konfigurationsvariablen gesetzt, die die Verbindung zum Simulationsserver beschreiben.

question: Dieses Modul ist ein Beispiel für die Funktions- und Darstellungserweiterung mit solchen Konfigurationsdateien: Es wird eine neue Funktion definiert, mit der im Radarfenster ein modales Dialogfenster mit einer Frage und zwei Antwortmöglichkeiten erzeugt wird. Die Antwort wird an den Simulationsserver zur Protokollierung versandt.

highlight: Dieses Modul definiert neue Funktionen, die auf Attribute der internen Repräsentation von Luftfahrzeugen zurückgreifen und die Farbdarstellung am Radarbild ändern. Das MoFL-Trainingsmodul (s. Abschnitt 3.3.4) greift auf die hier definierten Funktionen zurück.

radar: Dieses Modul definiert Farbschemata, Parameter und Optionen für das Anzeigesystem des Lotsenschülers.

maskiert: Dieses Modul ist wie *radar*. Zusätzlich wird die Maskierungsoption aktiviert.

ghost: In diesem Modul werden andere Anzeigeoptionen als für *radar* gesetzt. Zusätzlich wird das neue Fenster für den Geisterpiloten mit entsprechenden Funktionen definiert.

trainer: Ist ein weiter spezialisiertes *ghost*, in dem zusätzliche Einflußmöglichkeiten in die Anzeige des Lotsenschülers möglich sind, wenn dort *highlight* und *question* installiert sind.

getpos: Dieses Modul erweitert die Radardarstellung um die Möglichkeit, geographische Positionen in einem internen Format mit der Maus auszulesen. Diese Funktion ist im Entwicklungsprozeß neuer Szenarien hilfreich.

***sector*:** Dieses Interface beschreibt geographische Daten des Sektors, der dargestellt werden soll. Für jedes Szenario muß ein entsprechend angepaßtes Modul geladen werden.

Damit neue Funktionen in den beschriebenen Modulen möglichst wirksam programmiert werden können, wurde ein „Hook“-Konzept implementiert (zur Erweiterung von Frameworks mit „Hooks“ s. z.B. [60, S. 40]). Es ermöglicht die Änderung von internen Funktionen, ohne direkt in den Programmkern eingreifen zu müssen. Dadurch wird eine Erweiterung der Funktionalität „von außen“ über neue Konfigurationsmodule möglich.

Das „Hook“-Konzept bedeutet, daß das Kern-Programm, das die Module importiert, bestimmte Erweiterungsstellen, bzw. Hotspots vorsieht. Dieses Konzept aus der generativen Programmierung (s. für die Erweiterung von Frameworks mit Hotspots z.B. [60, S. 63ff]) versucht,

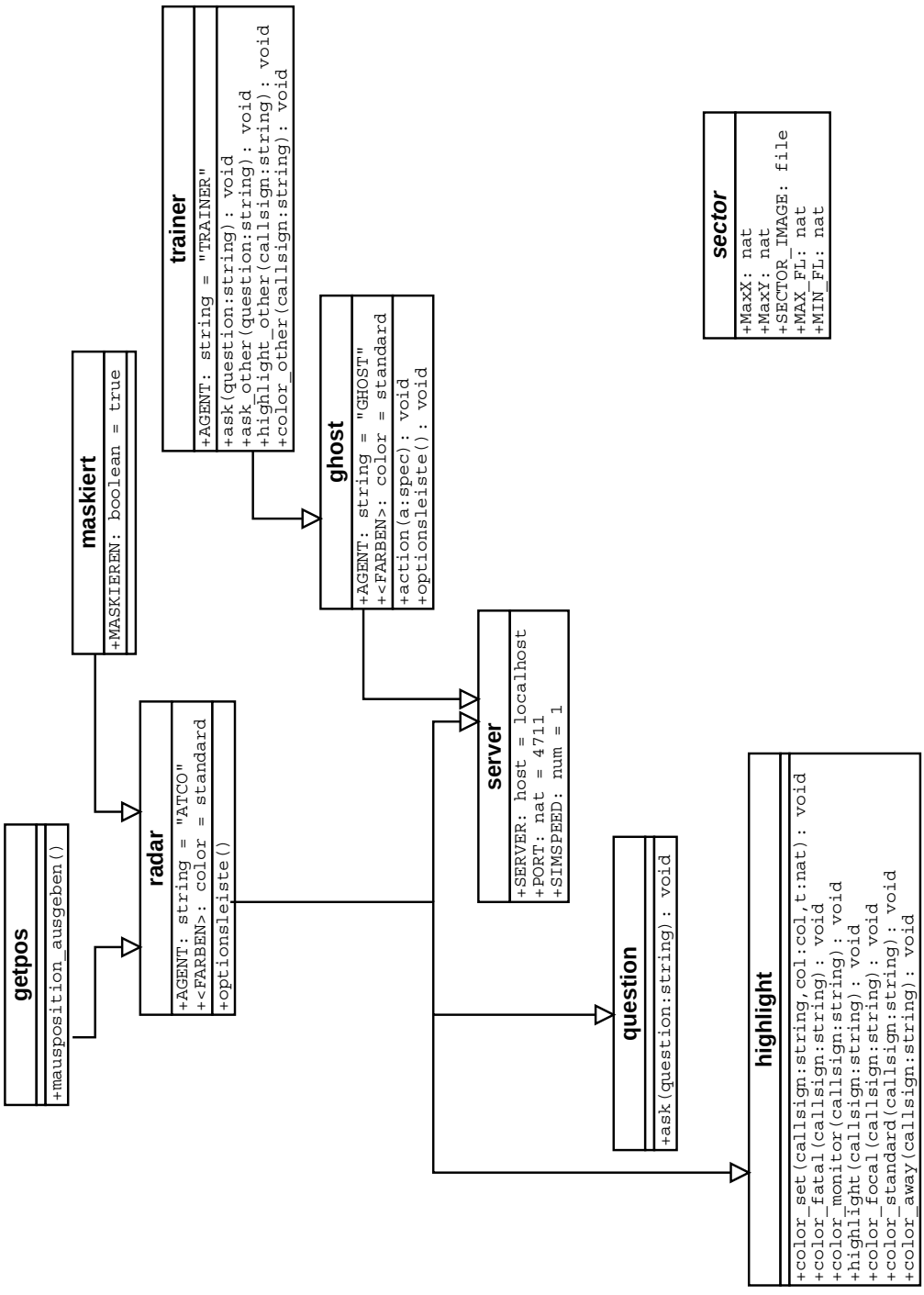

Abbildung 3.13: UML-Klassendiagramm zur Darstellung der Vererbungsbeziehungen der Module für die Konfiguration des Anzeigesystems.

die Wiederverwendbarkeit von Software zu erhöhen. Das geschieht durch die Beschreibung allgemeiner Rahmenarchitekturen, in denen an bewußt gewählten Punkten dokumentierte Erweiterungspunkte, die „Hooks“, vorgesehen werden.

Ein Hook ist eine Variable, die eine Liste von Referenzen auf Funktionen oder anonyme Funktionen (*lambda*-Ausdrücke in LISP) enthält. Zu bestimmten Zeitpunkten werden alle Funktionen, die ein Hook enthält, mit definierten Parametern ausgeführt. Das Rahmenwerk beschreibt dafür die allgemeine Verarbeitung und die Punkte, an denen die Hooks ausgeführt werden.

Dadurch, daß dieses Konzept auf das Anzeigesystem übertragen wurde, kann in den Modulen Funktionalität hinzugefügt werden, die zu bestimmten Zeitpunkten ausgeführt werden soll, ohne daß an diesen Stellen des Programmablaufes im Programmkern Änderungen erforderlich sind. Der Programmkern erwartet an diesen Punkten, daß den Hook-Variablen neue Funktionalität hinzugefügt wurde, die ja von überallher erreichbar sind. Die neuen Funktionen werden dann vom Programmkern an diesen Stellen ausgeführt.

Die Rahmenstruktur des Kerns sieht am Anfang eine allgemeine Initialisierung und den Aufbau der Fensterelemente des Radarfensters vor. Danach wird eine Socket-Verbindung zum Simulationsserver aufgebaut. Von dieser Verbindung werden dann in einer Schleife zeilenweise Kommandos gelesen, die mit einem auf regulären Ausdrücken basierenden Konzept parsiert werden.

Innerhalb dieser Schleife werden einige im Kern vorgesehene Kommandos erkannt und umgesetzt. Für alle nicht im Kern erkannten Kommandos wird angenommen, daß Tcl/Tk-Code empfangen wurde, der direkt ausgeführt wird. Dadurch lassen sich neue Kommandos als Funktionen in den Modulen definieren, was z.B. für alle bis jetzt implementierten Trainingsfunktionen genutzt wird. Fehler bei der Evaluation der empfangenen Tcl/Tk-Ausdrücke werden vom Ausnahmemechanismus in tcl abgefangen und unterdrückt.

Die wichtigste eingebaute Funktion ist das Update der Positionen und Parameter der Luftfahrzeuge. Es wird eine globale Liste verwaltet, in der alle dargestellten Luftfahrzeuge mit ihrem Rufzeichen, das innerhalb eines Simulationsszenarios einmalig sein muß, vermerkt sind. Alle Parameter der Luftfahrzeuge werden in Hash-Variablen mit den Rufzeichen als Schlüssel gespeichert.

Für jede dieser Variablen ist ein Tcl-*Variablen-trace* gesetzt, der bewirkt, daß bei einer Änderung des Variablenwertes die Darstellung im Radarfenster entsprechend angepaßt wird. Die Verarbeitung beim Update in der Hauptschleife beschränkt sich deshalb auf das Setzen der entsprechenden Variableninhalte. Dieses Konzept ist dem Smalltalk-Design-Pattern „Model-View-Controller“ entlehnt, bei dem je ein Objekt zur internen Repräsentation (*model*), Darstellung (*view*) und Änderung (*controller*) benutzt wird.

Es gibt in diesem Rahmen Hooks, die an folgenden Stellen ausgeführt werden:

- beim Initialisieren der Parameter,
- beim Initialisieren der Fensterinhalte,
- beim Update
- jeweils beim Darstellen der einzelnen grafischen Elemente für ein Luftfahrzeug.

In der implementierten Version wird nur der erste Hook benutzt, um die Daten aus einem *sector*-Modul zu lesen. Die anderen sind für spätere Konfigurationserweiterungen vorgesehen.

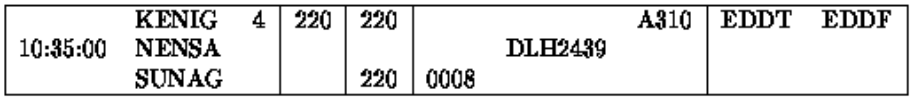

Abbildung 3.14: Darstellung eines vom Trainingssystem aus den Simulationsdaten generierten Flugstreifens

3.3.3 Flugstreifengenerator

Das Flugstreifensystem ist nicht als interaktive Schnittstelle zur laufenden Simulation implementiert. Es werden stattdessen mit einem Reportgenerator die relevanten Informationen aus einer Flugplan-, bzw. Simulationsskript-Datei ausgelesen, transformiert und als LaTeX-Datei ausgegeben, die dann nach PostScript konvertiert und ausgedruckt wird. Die Papier-Flugstreifen können dann nach den Sortierungspunkten geordnet werden.

Zur Aufbereitung wird ein einfaches Perl-Skript benutzt, das den Eingabestrom hauptsächlich aufgrund von regulären Ausdrücken parsiert. Die so gewonnenen Informationen werden mit LaTeX weiterverarbeitet und mit dvips nach PostScript konvertiert. Ein Makefile steuert die Verknüpfung dieser Skripte.

Ein Beispiel-Flugstreifen ist in Abb. 3.14 dargestellt. Dieser Fugstreifen bezieht sich auf den Sortierungspunkt NENSA (1. Spalte, Mitte). Das Flugzeug mit dem Callsign DLH2439 (4. Spalte Mitte), ein Airbus A310 (4. Spalte, oben), wird nach der Planung um 10:35:00 Uhr NENSA überfliegen (1. Spalte, links). Das Funkfeuer im Flugplan vor NENSA ist KENIG (1. Spalte, oben Mitte). Zwischen dem Überflug von KENIG und NENSA werden etwa vier Minuten verstreichen (1. Spalte, rechts oben). Das Funkfeuer nach NENSA ist SUNAG (1. Spalte, unten). Die Daten in der zweiten und dritten Spalte bedeuten (von links oben nach rechts unten): Die Einflughöhe in den Sektor ist FL 220, ebenso wie die Höhe über NENSA und die Ausflughöhe. Der Squawk, ein Funkidentifizierungscode ist 0008 (4. Spalte, unten). Die DLH2439 startete am Flughafen mit dem Code EDDT und landet in EDDF (5. Spalte).

3.3.4 Trainer

Als coachender Trainer nimmt ein Paket am Trainingsszenario teil, das mit CommonLISP, bzw. dem darauf aufsetzenden ACT-R implementiert ist. Es enthält folglich ACT-R als Komponente, auf die die anderen beiden Komponenten zurückgreifen: MoFL in einer angepaßten Version und Trainingsfunktionen, die den Zustand der Simulation von MoFL benutzen.

MoFL wurde entsprechend dem Entwurf in Abschnitt 3.2.4 angepaßt. Dabei mußte wegen der Eingriffe des Lotsenschülers, die außerhalb der Kontrolle der Simulation des kognitiven Modells stehen, Vorsorge für ein Wahrheitserhaltungssystem im Sinne des nicht-monotonen Schließens getroffen werden. Dazu werden lediglich entsprechende Chunks aus dem Arbeitsgedächtnis bei Eingriffen gelöscht und eine Neubearbeitung angestoßen. Die Eingriffe werden von der Kopplungskomponente des Simulationsserver-Paketes an das kognitive Modell weitergemeldet.

Das *picture* der Simulation des kognitiven Modells fungiert als Repräsentation der aktuellen Relevanz der Objekte und Konstellationen in der Verkehrssituation, auf das Trainingsfunktionen aus der Trainingskomponente dieses Paketes zugreifen. Aufgrund des Zustandes des *pictures* werden zu entsprechenden Zeitpunkten über die Kopplungskomponente des Simulationsservers entsprechende Kommandos an die Anzeigeeinheit des Lerners gesendet.

Als Kommandos werden die im *highlight*-Modul vereinbarten tcl-Funktionen des Verkehrsdarstellungs-Paketes benutzt, die an den Simulationsserver gesendet, an die Verkehrsdarstellungen weitergereicht und dort ausgeführt werden.

Die Kopplung des LISP-Prozesses an den Simulationsserver über Sockets geschieht über ein allgemeines Konzept. Bei Programmierumgebungen, deren Hauptzweck die Formulierung von Algorithmen ist, und die i.A. nicht zur systemnahen Programmierung eingesetzt werden, fehlen oft Schnittstellen zur Programmierung von Netzwerkkommunikation. Dies ist bei LISP der Fall, so daß zur Socketkommunikation das folgende Konzept angewendet wird.

Es kommen zwei unterschiedliche Programmiersprachen zum Einsatz: eine systemnahe, mit der Socketverbindungen aufgebaut werden können und mit der interne Datenstrukturen durch Systemaufrufe manipuliert werden können, und LISP, mit dem das Trainer-Paket implementiert ist. Die systemnahe Sprache öffnet den Socket und ändert Einträge in der Dateideskriptorentabelle (fd-table), so daß danach Lese- und Schreiboperationen von StandardEingabe und nach StandardAusgabe auf die Socketverbindung umgelenkt werden. Danach wird in diesem so manipulierten Prozeß das LISP-Programm gestartet. Durch die Art des Aufrufes bleibt die Socketumgebung erhalten, so daß der LISP-Prozeß nun einfach auf den Socket zugreifen kann. Im folgenden Abschnitt wird diese Art des Programmaufrufes und die Manipulationen an den Dateideskriptoren ganau erläutert.

Das Konzept nutzt die Struktur der Prozeßverwaltung in UNIX-Systemen aus. Jeder Prozeß läuft hier in einem Kontext ab. Der Kontext ist eine Datenstruktur im Speicher des Betriebssystems. Er hat einen statischen und einen dynamischen Teil, in dem Informationen zur Prozeßkontrolle und zum Zustand der sequentialisierten Abarbeitung der Prozesse gespeichert sind.

Im statischen Teil des Prozeß-Kontextes wird zwischen dem Benutzerkontext, in dem Programmcode, Daten und Stack des Prozesses repräsentiert sind, und dem Systemkontext unterschieden. Der Systemkontext beinhaltet einen Prozeßtabelleneintrag, der Informationen über den Prozeßzustand, Steuerinformationen und einen Verweis in den dynamischen Teil des Prozeßkontextes enthält, den u-Bereich und die p-Regionstabelle.

Die p-Regionstabelle enthält Verweise auf die für den jeweiligen Prozeß gültigen Teile des Benutzerkontextes. Der u-Bereich enthält insbesondere Verweise auf die Dateitabelle (fd-table in Abb. 3.15 und 3.16). Beim Benutzen des Systemaufrufes exec wird ein neuer Benutzerkontext angelegt und die p-Regionstabelle entsprechend geändert (s. Abb. 3.16). Der Effekt dieses Systemaufrufes ist, daß das Programm, das in dem aufrufenden Prozeß abläuft, durch ein anderes Programm ersetzt wird. Dies wird z.B. bei der Programmierung von Shells ausgenutzt, die beim Aufruf eines Programms einen Kindprozeß mit fork erzeugen. Im Kindprozeß ist am Anfang das Programm des Elternprozesses, also das der Shell. Das Shellprogramm im Kindprozeß erkennt, daß es ein Kind ist, und ruft den exec-Systemaufruf auf, mit dem das eigentlich zu startende Programm in den Prozeß des Kindes geladen wird. Danach arbeitet der Kindprozeß das aufgerufene Programm ab.

Eine Eigenschaft dieser Methode ist, daß der u-Bereich beim exec-Systemaufruf nicht geändert wird. Dadurch bleiben Modifikationen an der Dateitabelle erhalten. Die Dateitabelle enthält Verweise auf geöffnete Dateien. Sie ist als Array implementiert. Zugriffe auf Dateien sind über den Index des entsprechenden Eintrages in der Dateitabelle zugreifbar. Wird eine Datei mit dem Systemaufruf close geschlossen, bleibt der Eintrag in der Dateitabelle leer, bis eine neue Datei geöffnet wird. Ihr Dateideskriptor wird in die erste leere Zelle der Dateideskriptortabelle geschrieben.

Der Systemaufruf dup bewirkt, daß ein Dateideskriptor, der über den Index in der Dateitabelle angegeben wird, dupliziert wird und seine Referenz auf die Datei in die erste leere Zelle der Dateitabelle kopiert wird.

In einer systemnahen Programmiersprache, die Funktionen zur Netzwerkkommunikation bereitstellt, kann nun ein kleines Programm geschrieben werden, das eine Socketverbindung öffnet. Durch Öffnen des Sockets und geschickte Manipulation der Einträge in der Dateitabelle mit den Systemaufrufen close und dup werden Dateideskriptoren so verbogen, daß vor dem Aufruf des exec die Standardeingabe des Prozesses vom Socket liest und die Standardausgabe auf den Dateideskriptor des Sockets schreibt (s. Abb. 3.15). Denn Sockets werden ähnlich wie Dateien behandelt. Sie werden auch in der Dateitabelle mit Dateideskriptoren eingetragen.

Da alle Programmiersprachen Primitive zum Lesen und Schreiben von, bzw. nach Standard-Eingabe und -Ausgabe haben, kann nun mit exec das Programm in der systemfernen Programmiersprache, z.B. LISP, gestartet werden. Für dieses Programm werden nun alle Zugriffe auf die Standardkanäle auf den Socket umgelenkt, da lediglich die Daten und Code-Segmente im Benutzerkontext und die p-Regionstabelle geändert wurden, nicht jedoch der u-Bereich, in dem die Dateitabelle (fd-table) liegt (s. Abb. 3.16).

Der Kommunikationsaufbau, die Manipulation der Dateitabelle und des u-Bereiches erfolgen im Trainer-Paket durch ein kleines Perl-Programm, das danach den LISP-Interpreter mit exec aufruft. Der LISP-Interpreter wird mit entsprechenden Kommandozeilenoptionen gestartet, so daß er ACT-R, MoFL und die Trainingsfunktionen nachlädt. Dieses LISP-Programm benutzt dann Standardoperationen, um die Standardkanäle zu lesen, bzw. zu beschreiben. Da die Dateitabelle entsprechend geändert wurde, werden diese Zugriffe auf den Socket umgeleitet, der diesen Prozeß mit der Kopplungskomponente des Simulationsservers verbindet. Das Trainer-Paket kann dadurch Informationen über die Verkehrssituation aufnehmen und Anweisungen an die Anzeigeeinheit des Lotsenschülers senden, um die Darstellung wichtiger Flugzeuge zu ändern.

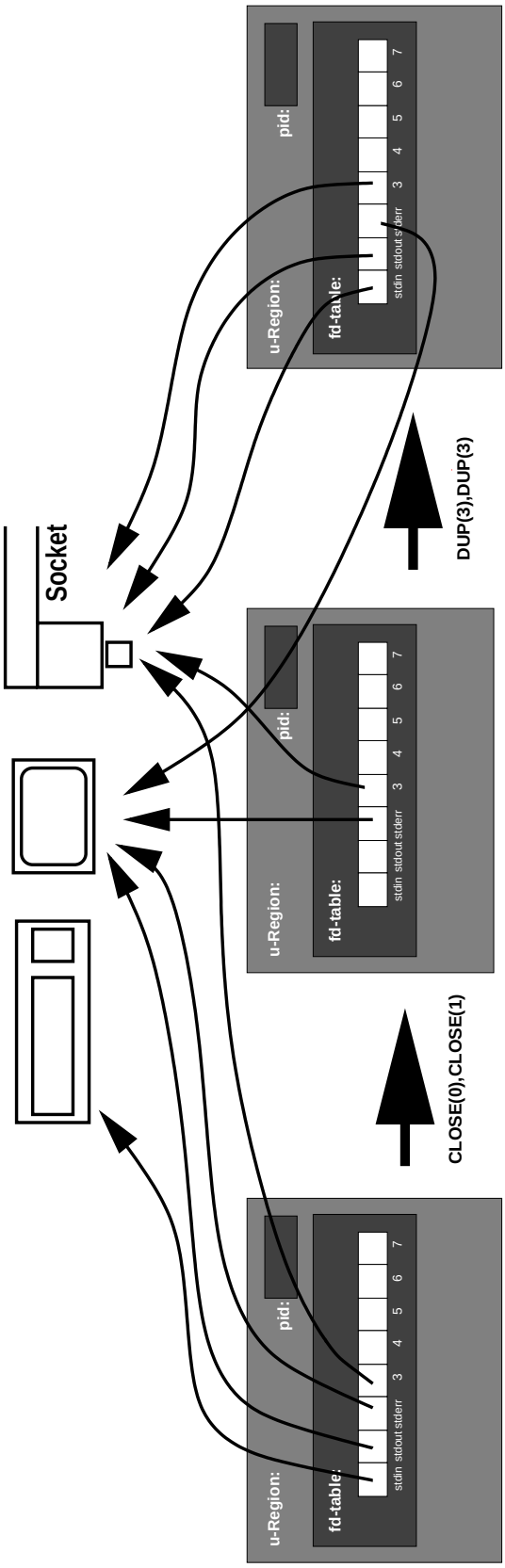

Abbildung 3.15: Aufruf der Systemaufrufe close und dup zur Manipulation der Dateideskriptorentabelle (fd-table). Nach Ausführen dieses Schemas sind die Standard-Kanäle mit dem Socket verbunden.

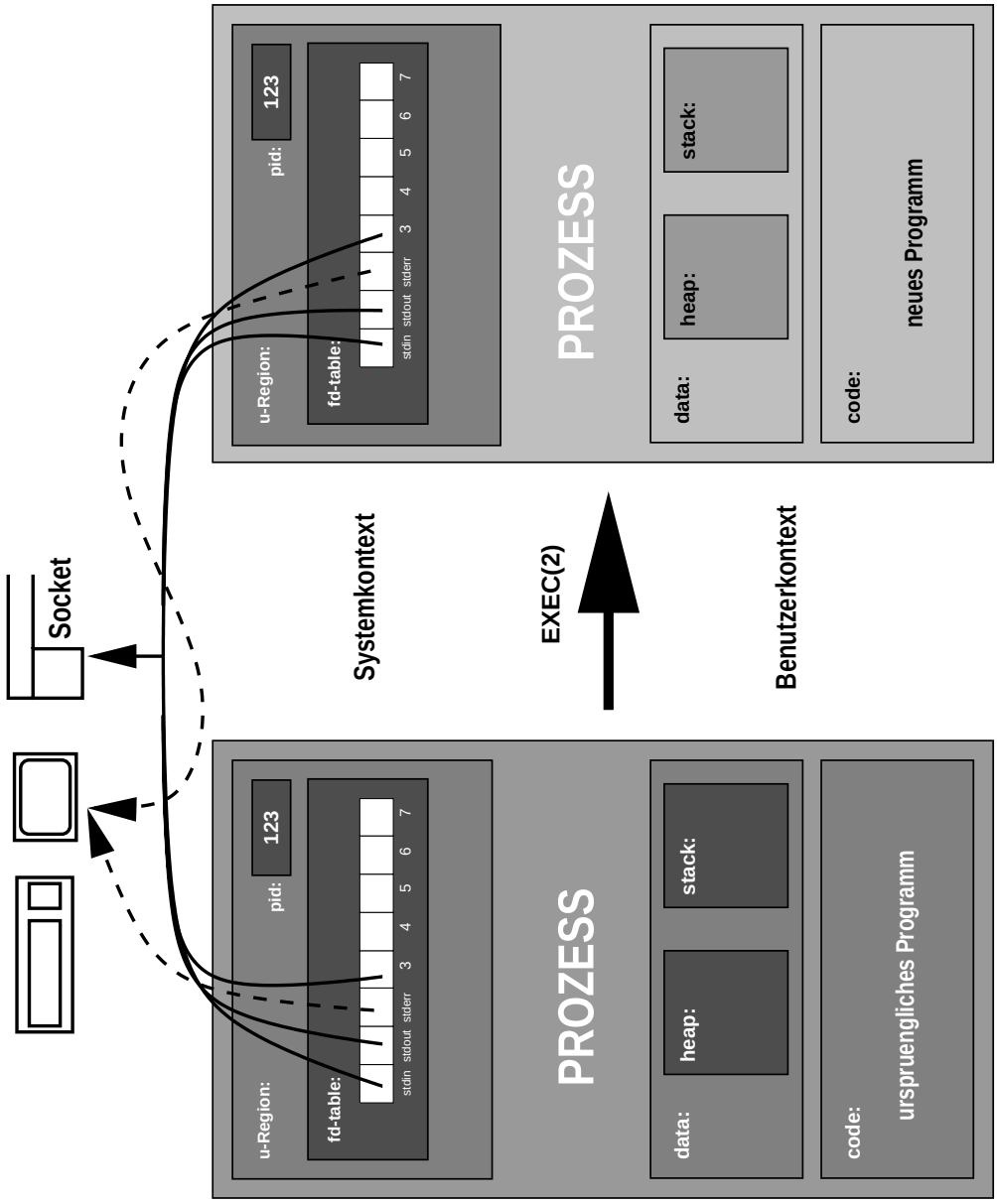

Abbildung 3.16: Aufruf der Systemfunktion exec, mit dem der Benutzerkontext des Prozesses mit einem neuen Programm überschrieben wird. Die u-Region, in der die Dateideskriptoren (fd-table) liegen, bleibt erhalten.

4 Zusammenfassung und Ausblick

Es wurden theoretische Konzeption, Entwurf und Umsetzung eines computergestützten Trainingssystems für die Flugsicherung vorgestellt.

Das Konzept basiert auf der Integration eines kognitiven Modells in ein Simulationssystem zum Training der Streckenflugkontrolle. Das kognitive Modell ersetzt teilweise den Ausbilder der gegenwärtigen Trainingssituation. Das Ziel ist eine Flexibilisierung für den Lotsenschüler und eine bessere Ressourcenauslastung für den Träger der Ausbildung.

Das Trainingssystem kann zum Selbststudium von Fluglotsenschülern benutzt werden, die in ihrer Ausbildung bereits theoretische Kenntnisse über die Streckenflugkontrolle erworben haben. Es ersetzt nicht das von einem Ausbilder angeleitete Training an den aufwendigen Simulationseinrichtungen des Ausbildungsträgers, sondern hilft als flankierende Maßnahme, das praktische Training zu vertiefen.

Das entwickelte Trainingssystem besteht aus mehreren Komponenten, die als eigenständige Prozesse mit interpretierten Skriptsprachen implementiert wurden. Die Prozesse kommunizieren über das Internetprotokoll und können auf verschiedenen Rechnern verteilt ablaufen.

Zwei wesentliche Neuerungen zu bestehenden Systemen, auch aus anderen Domänen, wurden dabei eingeführt:

- Das kognitive Fluglotsenmodell MoFL wird dazu benutzt, eine aktuelle Repräsentation der Verkehrssituation aufzubauen. Format und Inhalt der Repräsentation spiegeln die Aufmerksamkeitsverteilung eines erfahrenen Fluglotsen wieder. Durch unterschiedliche Einfärbungen der Flugzeuge in der Radardarstellung des Fluglotsenschülers wird dessen Aufmerksamkeit auf relevante Objekte und Konstellationen gelenkt.

  Dadurch wird ein Teil der Unterstützung des Fluglotsenausbilders während des Simulatortrainings nachgebildet.

- Durch den Einsatz von interpretierten Skriptsprachen und die Modularisierung in einzelne Prozesse erreicht die Implementierung eine hohe Flexibilität im Einsatz. Eine Umkonfiguration ist im laufenden Betrieb und ohne Eingriff in die Programme möglich.

  Die Möglichkeit zur Metaprogrammierung zwischen den beteiligten Komponenten reduziert den Entwicklungsaufwand, weil die jeweiligen Parser bereits Teil der Entwicklungsumgebungen sind und weil jeweils nur die Senderseite bei einer Änderung betroffen ist. Der Empfänger braucht vom Programmierer nicht direkt geändert zu werden, weil der empfangene Prozeß durch die Nachricht des Senders umkonfiguriert wird.

  Die übliche Entwicklungszeitersparnis beim Einsatz von Skriptsprachen von 50% – 60% wird in diesem Projekt übertroffen.

Das implementierte Simulationssystem unterscheidet sich von anderen dadurch, daß für die räumliche Repräsentation der Positionen und Bewegungen der Luftfahrzeuge auf das Pixelformat der Anzeigeeinheit statt auf geographische Koordinaten in Längen- und Breitengradangaben zurückgegriffen wurde.

Die Verwendung des Koordinatensystems der Anzeigeeinheit im gesamten System erhöht die Laufzeiteffizienz, weil alle Umrechnungen bei der Entwicklung eines Szenarios und nicht während der Simulation passieren. Die Entwicklung des Trainingssystems wurde dadurch vereinfacht.

Obwohl der Einsatz von interpretierten Skriptsprachen positive Effekte hat, zeigen sich in diesem Projekt einige Schwachstellen:

- Durch die Interpretierung der Programme ergeben sich Laufzeitverluste insbesondere im Simulationskern. Das Laufzeitverhalten der Darstellung ist hingegen unkritisch.
- Der Design-Ansatz von Perl, für jede Aufgabe mehrere alternative Konstrukte vorzusehen, und die syntaktischen Schwächen bei der Verwendung verschachtelter Datenstrukturen erlauben nicht wirklich die Implementierung und Wartung eines komplexen Systems mit möglicherweise mehreren Entwicklern („Programmierung im Großen“).
- Der Mix mehrerer Implementierungssprachen (ACT-R, LISP, Perl, Tcl/Tk) erhöht den Einarbeitungsaufwand für die Wartung und erschwert die Verwendung eines Software-Engineering-Werkzeuges.

Zu der Wirksamkeit des vorgeschlagenen Trainingskonzeptes und des implementierten Software-Trainers und der Belastung eines möglicherweise beteiligten menschlichen Trainers liegen noch keine Erfahrungen vor. Es ist also auch noch nicht abschätzbar, unter welchen organisatorischen Rahmenbedingungen der Einsatz des Trainingssystems erfolgen kann. Denkbar sind sowohl der autonome individuelle Einsatz, wo statt des Ausbilders die Software-Trainingskomponente benutzt wird, als auch eine 1:1 (Trainer:Schüler) oder 1:N Situation mit oder ohne Unterstützung des Software-Trainers MoFL.

Die möglichen nächsten Arbeitsschritte betreffen sowohl das Konzept, als auch dessen Implementierung:

- Zur Vereinheitlichung der Programme sollten alle Perl- und Tcl/Tk-Komponenten mit der Programmiersprache Python reimplementiert werden. Die Trainingskomponente hingegen basiert auf ACT-R und ist deshalb nicht mit vernünftigem Aufwand außerhalb von LISP zu realisieren.

  Python bietet Konstrukte der Objektorientierung und unterstützt unter anderem dadurch die „Programmierung im Großen“. Da auch das Tk-Toolkit in Python benutzt werden kann und alle verwendeten Eigenschaften wie reguläre Ausdrücke oder Variablen-*traces* von Perl und Tcl auch in Python vorhanden sind, ist der Portierungsaufwand gering.
- Da das Laufzeitverhalten des Simulationskerns bei sehr großen Szenarien auf langsamen Rechnern nicht befriedigend ist, sollte er in C neu implementiert werden und in das Simulationsskript eingebunden werden. Die Verwendung des SWIG-Rahmens und -Werkzeuges ermöglicht eine Skriptsprachen-unabhängige Entwicklung sowohl für Perl, Python, Tcl/Tk und scheme.

Während die Umsetzung dieser beiden Arbeitspakete nicht wirklich zwingend ist, sondern „nur“ einer besseren Wartbarkeit und einem optimierten Laufzeitverhalten dient, ist eine Evaluation des Trainingssystems von herausragender Bedeutung für eine Fortsetzung der Arbeit in diesem konzeptionellen Rahmen. Es muß dabei sowohl der Nutzen des implementierten Software-Trainers, als auch die Belastung eines möglichen menschlichen Trainers untersucht werden. Als Maß für die Güte des Software-Trainers könnten Fehler der Trainees, also Separationsunterschreitungen, und Eingriffszeitpunkte bei wechselnder Verkehrslast erhoben und mit einer Situation ohne Trainerunterstützung verglichen werden.

Wenn Erfahrungen zu der Wirksamkeit des Software-Trainers vorliegen, können Modifikationen oder Erweiterungen der Trainingsfunktionen entworfen werden. Denkbar sind z.B. die selektive Benutzung des *velocity leaders* und ein Übergang zu einem Lernpfad-orientierten Trainingssystem.

Der Lernpfad eines konventionellen CBT-Systems ist die Verkettung einzelner Bildschirmmasken, mit denen Wissenseinheiten vorgestellt werden. In diesem Tutor-System könnte auch ein Lernpfad eingebaut werden, der jedoch nicht statisch verknüpft wäre, wie in konventionellen CBT-Systemen. Es müßten vielmehr zu jeder Bildschirmmaske, die zusätzlich zum Radarbild eingeblendet wird, Bedingungen im *picture* von MoFL spezifiziert werden. Möglicherweise sollten auch Verknüpfungen zwischen den Masken möglich sein.

Bei der Entwicklung dieses Konzeptes wären einige Entscheidungen zu treffen, die möglicherweise nach einer erneuten Evaluation revidiert werden müssen:

- Soll die Simulation angehalten werden, während eine Maske angezeigt wird? Soll das Anhalten erst nach einer gewissen Anzeigedauer passieren?
- Soll der Schüler zwischen unterschiedlichen Masken navigieren können, oder soll die Anzeige ausschließlich durch den Zustand des *pictures* gesteuert werden?
- Welche Art von Lerninhalten soll vermittelt werden?
- Soll das dynamische Radarbild in die Masken integriert werden?

Ein weiterer Ausbau des Trainingssystems ist durch die Auswertung von Demaskierungsvorgängen möglich. Sie spiegeln in begrenztem Rahmen die Interaktion des Lernenden mit der Aufgabenumgebung wieder. Es könnte ein Konzept entwickelt werden, das den Zustand des *pictures* in MoFL mit den Interaktionen des Schülers verknüpft. Dadurch wäre es möglich, ein wirkliches Lerner-Modell aufzubauen und Abweichungen vom Zustand des *pictures* von MoFL zu erkennen, damit der Lerner gezielt unterstützt werden kann.

Das vorgestellte Konzept kann auf das Training der Kontrolle in anderen Domänen übertragen werden. Dabei muß jedoch als Voraussetzung ein implementiertes kognitives Modell vorhanden sein. Das kognitive Modell muß ein Weltbild aufbauen und aktuell halten. Aufbau und Struktur dieses Weltbildes können dann für Trainingsfunktionen genutzt werden.

Der besondere Vorteil dieses Trainingssystems liegt jedoch gerade in seiner Domäne begründet, da es bei der Flugsicherung nicht möglich ist, Interaktionen in ausreichendem Maß zu messen, um ein Schülermodell aufzubauen. In anderen Domänen können mehr Informationen über den Schüler durch die Beobachtung seines Verhaltens gewonnen werden.

Die implementierte Software ist zum großen Teil spezifisch für die gewählte Anwendungsdomäne. Der Simulationsserver mit seiner Kopplungskomponente und seinem Rahmen, der das

luftfahrtspezifische Simulationskern-Modul einbindet, ist auch in anderen Domänen einsetzbar, in denen diskret simuliert werden kann. Für die anderen Komponenten können lediglich die Prinzipien übernommen werden. Dazu gehört in erster Linie der Austausch von Nachrichten im Format der Programmiersprache des Empfängers, die dann beim Empfang gleich ausgeführt werden können, wodurch eine flexible Metaprogrammierung zwischen den Komponenten möglich wird.

Glossar

ACT-R *adaptive control of thought — rational*. Theorie, ↑ Architektur und Implementierungsumgebung zur ↑ kognitiven Modellierung auf der Basis von Produktionensystemen. ACT-R enthält deklaratives und prozedurales Wissen auf einer symbolischen Ebene. Auf der subsymbolischen Ebene steuern Zahlen-Parameter das Verhalten der Verarbeitung auf der symbolischen Ebene. ACT-R beinhaltet Mechanismen zum Vergessen, zum Fehlermachen, zum Lernen und zur nicht exakten Verarbeitung von Wissen und versteht sich daher als Anwärter für eine ↑ UTC. In neueren Entwicklungen wurde die Theorie vom Bereich des Problemlösens auf Wahrnehmung und Motorik erweitert.

Adaptivität In diesem Zusammenhang wird die Anpassung eines Trainingssystems an den Schüler betrachtet. Sie kann auf unterschiedlichen Ebenen passieren. Bei der Makroadaption wird die gesamte Trainingssituation an die Bedürfnisse des Schülers angepaßt. Die Verwendung von Lernsoftware paßt z.B. die Lernsituation an die Wünsche des Lerners an, zu welchem Zeitpunkt und wie lange er trainiert wird, im Gegensatz zum Unterricht mit einem menschlichen Lehrer. Die Mikroadaption geschieht während des Trainings. Das Wiederholen einer Lektion bei mangelndem Lernerfolg des Schülers ist dafür ein Beispiel. Zum Erreichen von Mikroadaption in einem Softwaresystem ist die Diagnose des Lerners notwendig.

Architektur zur kognitiven Modellierung Im Rahmen der Annahme, Kognition ließe sich mit Algorithmen simulieren, legt eine solche Architektur die Struktur, Verbindung und Kapazitzät der beteiligten Prozesse oder Speicher fest, die als gegeben angesehen werden.

Callsign ↑ Rufzeichen

CBSM *context based state monitoring*. Eine Methode, um den Zustand eines technischen Systems zu überwachen. Dazu werden Aktivitäten definiert, die jeweils ein Ziel haben. Für jede Aktivität werden initiale Systemzustände spezifiziert. In Situationsknoten, die in einem Graphen initiale Zustände mit dem gewünschten Zielzustand verbinden, werden markante Zwischenzustände repräsentiert. Die Zustandsübergänge bilden dann den Weg der Zielerreichung ab. Die Gesamtheit der beteiligten Knoten wird Erwartungsnetzwerk genannt. Der Kontext ergibt sich aus der aktuellen Position im Netzwerk, bzw. der Reihe von Zustandsübergängen, mit der die aktuelle Situation vom Zielzustand aus erreicht wurde.

CBT *computer based training/teaching*. Softwaresysteme zur Vermittlung von Wissen. Die am weitesten verbreitete Ausprägung von CBT ist mit Bildschirmmasken realisiert, in denen Informationen — inzwischen vorwiegend als multimediale Darstellungen — präsentiert werden. Die Masken sind miteinander verbunden. Der Lerner kann wie in einem Hypertext zwischen den vorhandenen Masken navigieren, um passende Informationen dargeboten zu bekommen. Am Ende einer Lektion finden sich oft Lernerfolgskontrollen. Werden bei diesen Kontrollen Defizite diagnostiziert, kann zu den entsprechenden Masken zurückgesprungen werden. Eine andere Form von CBT sind ↑ ITS.

CDI *climb descent indicator*. Ikonische Information im Radarbild bei der Flugsicherung, die angibt, ob das dazugehörende Flugzeug steigt, sinkt oder die Höhe nicht ändert (*level*-Flug).

COGENT *cognitive objects in an graphical environment*. Implementierungsumgebung

für kognitive Modelle im Rahmen von ↑ Mikrotheorien.

CommonLISP Standardisierter Sprachumfang von LISP (*list processing language*). Hybride Programmiersprache, mit der im funktionalen, imperativen und objektorientierten Paradigma gearbeitet werden kann. Es gibt mehrere Implementierungen der CommonLISP-Sprachdefinition.

CORBA *common object request broker architecture*. Eine Middleware-Architektur, die in mehreren Implementierungen vorliegt. Sie dient zur Verbindung von Prozessen in einem heterogenen Rechnernetzwerk. Clientobjekte können Methoden entfernter Serverobjekte aufrufen. Den Verbindungsaufbau zwischen beiden und den Aufruf der Methoden übernimmt der *object request broker* (ORB). Die Klassen der beteiligten instantiierten Objekte können in unterschiedlichen objektorientierten Programmiersprachen implementiert sein. Mit der *interface definition language* wird deshalb die abstrakte Aufrufschnittstelle definiert, so daß der ORB Nachrichten und Typen entsprechend in die Konventionen der Serverobjekt-Programmiersprache umsetzen kann.

DFS *Deutsche Flugsicherungs GmbH*. Flugsicherungsorganisation in der Bundesrepublik Deutschland.

Dictionary Container-Datenstruktur, bei der der Zugriff auf die enthaltenen Elemente über einen alphanumerischen Schlüssel erfolgt.

DIS *distributed interactive simulation*. Effizientes Kommunikationsprotokoll, das für die Echtzeit-Verbindung von Komponenten in verteilten Simulationsverbünden gedacht ist. Der Ursprung dieses Protokolls ist militärisch. An Adaptionen für den nichtmilitärischen Einsatz von DIS wird gearbeitet. Es gibt eine Variante für die Simulation in der zivilen Flugsicherung (ATCSP — *air traffic control simulation protocol*). Andere mögliche Einsatzgebiete sind die Katastrophenplanung und Computerspiele.

en-route In der ↑ Flugsicherung ist der zu kontrollierende Luftraum sowohl horizontal, als auch vertikal in Sektoren aufgeteilt. Während die Sektoren in niedriger Höhe kleiner sind und zur Koordination des An- und Abfluges von Flughäfen dienen, haben sie in größerer Höhe eine weitere Ausdehnung. In diesen höheren „en-route“ Sektoren haben die Flugzeuge i.A. Reisehöhe und -geschwindigkeit erreicht. Der in dieser Arbeit benutzte Sektor WR2 ist im en-route Bereich. Er erstreckt sich in einer Höhe von 13.500 ft bis 28.000 ft.

EPIC *executive process — interactive control*. ↑ Kognitive Architektur auf der Basis eines Produktionensystems. Anwendungsbereich der Implementierungsumgebung ist die Bewertung von Entwurfsprototypen für Arbeitsumgebungen. Besondere Beachtung wird deshalb Wahrnehmungsprozessen gewidmet. EPIC ist jedoch keine ↑ UTC, weil viele kognitive Prozesse, insbesondere Lernen, nicht berücksichtigt werden.

Flugsicherung Das Ziel der Flugsicherung ist, den Luftverkehr sicher und ökonomisch zu regeln. Dazu werden alle Flugbewegungen im voraus geplant und kontrolliert. Zur Gewährleistung der Flugsicherheit wird der gesamte Luftraum in Zuständigkeitsbereiche unterteilt, die von jeweils einem Lostenteam kontrolliert werden. Die Fluglotsen geben Anweisungen, die von den Piloten umgesetzt werden müssen.

Flugstreifen Routen, Flughöhen und -geschwindigkeiten aller kontrollierten Flüge werden im voraus geplant. Einige dieser Planungsinformationen werden Fluglotsen auf Papier-Flugstreifen zur Verfügung gestellt. Sie erhalten sie etwa 20 Minuten vor Eintritt des betreffenden Flugzeuges in den Sektor.

Framework Ein Rahmenmodell für die Entwicklung eines Systems. Im Bereich des Softwareengineerings beschreiben solche Modelle das Zusammenspiel vorgefertigter oder allgemein gebräuchlicher Objekte. In der ↑ kognitiven Modellierung wird hingegen nicht eine Architektur wie ↑ ACT-R oder ↑ SOAR als Rahmen angesehen, sondern die ihnen zugrundeliegende allgemeine Vorstellung über die Modellierung von Kognition, also in diesen Beispielen Wissensverarbeitung mit Regel- bzw. Produktionensystemen.

Geisterpilot Person, die in einem Flugsicherungs-Simulationsszenario die Rolle mehrerer oder

aller Piloten übernimmt. Sie nimmt Anweisungen von Lotsen über eine simulierte Sprechfunkverbindung entgegen und greift über eine Schnittstelle zur Simulation den Anweisungen entsprechend in den Verkehr ein.

Hash-List ↑ Dictionary

Hook In Softwaresystemen sind „Haken“ Erweiterungspunkte, an denen neue Funktionen eingefügt werden können. Ein Hook ist eine Variable, die eine Liste von Referenzen auf Funktionen enthält. Jeder Haken wird zu einem definierten Punkt in der Ablauflogik des Programms ausgeführt, d.h. alle Funktionen, die von ihm referenziert werden, werden aufgerufen. Der Punkt liegt normalerweise am Anfang oder am Ende eines größeren Verarbeitungsschrittes. Dadurch kann der Verarbeitungsschritt erweitert werden, ohne daß in das Programm des Verarbeitungsschrittes eingegriffen werden muß.

Idiom In der Softwaretechnik ist ein Idiom ein in einer bestimmten Programmiersprache übliches Vorgehen, um eine gegebene Funktionalität zu erreichen. Es hat nicht die Allgemeingültigkeit eines ↑ Patterns, ist jedoch flexibler als ein Makro. Es gilt i.a. nur in einer Programmiersprache. Bspw. ist das Parsieren eines Strings mit „Pointerarithmetik“ ein Idiom in der Programmiersprache C.

Inferenz Schlußmechanismus, um aus vorhandenem Wissen neues zu erzeugen. Dabei wird häufig die Regelverknüpfung (Modus ponens) angewandt:

Vorbedg. 1	wenn p dann q
und Vorbedg. 2	p
dann Schluß	deshalb q

ITS *intelligent turoring systems*. ↑ CBT-Systeme, die mit wissensbasierten Methoden ein Modell des Schülers aufbauen. Das Ziel ist eine Verbesserung der ↑ Adaption an den Lerner. Es sind weitere wissensbasierte Module vorhanden, die Wissen über den Lehrstoff und Lehrmethoden enthalten. Die Verknüpfung von Wissen über Schüler, Lehrmethoden und Lehrstoff ermöglicht eine an den aktuellen Kenntnisstand des Lerners angepaßte Aufbereitung und Darstellung des Lehrstoffs.

Kognitive Modellierung Methode in der Kognitionswissenschaft. Modelle der kognitiven Vorgänge werden mit Beschreibungsformalismen notiert und deren Verhalten mit dem Computer simuliert. Ergebnisse der Simulation helfen bei der Bewertung der Modelle und liefern Informationen für die Gestaltung empirischer Arbeit, die der Anpassung der Modelle dient.

Level-Flug ↑ CDI

LISP ↑ CommonLISP

Mensch-Maschine System Der Verbund aus technischem System und Bediener/Benutzer in der Arbeitswelt. Es wird eingebettet in ein organisatorisches Umfeld betrachtet. Der Zweck der Beschäftigung mit Mensch-Maschine Systemen ist die Verbesserung der Bedienbarkeit oder Erlernbarkeit der Schnittstelle des technischen Systems zum Benutzer.

Metaprogrammierung Programmiertechnik, bei der ein Programm ein anderes Programm erzeugt, wie z.B. bei einem Compiler. In dieser Arbeit ist die dynamische Erzeugung eines Unterprogramms für den generierenden oder einen anderen Prozeß gemeint: Viele vorwiegend interpretierte Programmierprachen bieten als Sprachkonstrukt den Aufruf eines Interpreters an, der die eigene Sprache akzeptiert. Als Eingabe werden Strings oder andere von Programmen generierbare Datenstrukturen akzeptiert. Erzeugt ein Programm nun ein weiteres Programm als Text, kann es direkt in demselben Prozeßkontext ausgeführt oder an einen anderen Prozeß geschickt und dort interpretiert werden. Ein Programm hat dann die Möglichkeit, sich selbst oder einen anderen Prozeß dynamisch umzuprogrammieren.

MIDAS *man-machine integration design and analysis system*. Dieses System der NASA kombiniert CAD-Systeme zum Prototyping von Cockpit-Ausstattungen mit der dynamischen Simulation des Verhaltens und der Belastung einer Crew mit dem Ziel, den Entwurf neuer Cockpits zu unterstützen und zu bewerten.

Mikrotheorie Als Mikrotheorie soll in diesem Zusammenhang ein ↑kognitives Modell verstanden werden, daß ausschließlich einen bestimmten Aspekt von Kognition zu erklären versucht und das nicht in eine allgemeine ↑kognitive Architektur eingebettet ist, die den Anspruch erhebt, eine ↑UTC zu sein.

MMS ↑Mensch-Maschine System

MoFL *Modell der Fluglotsenleistungen*. Konzeptionelles und implementiertes Modell der kognitiven Vorgänge und Strukturen bei erfahrenen Fluglotsen in der Streckenflugkontrolle auf der Basis von ↑ACT-R.

NATS *national air traffic services*. Flugsicherungsorganisation in Großbritannien.

nicht-monotones Schließen ↑Inferenzmechanismus, bei dem berücksichtigt wird, daß bereits geschlossenes Wissen durch neue Erkenntnisse, z.B. aus der dynamischen Umgebung, ungültig werden kann.

Pattern Entwurfsmuster im Softwareengineering. Ein Pattern ist die Beschreibung der Struktur und der inneren Zusammenhänge einer erfolgreichen Familie von Lösungen eines wiederkehrenden Problems, das in einem zu spezifizierenden Kontext und einem System von Randbedingungen auftritt. Entwurfsmuster haben einen Namen, unter dem sie in einer halbformalen Sprache dokumentiert werden.

Perl *practical extraction and report language*. ↑Skriptsprache, die bei einer sehr flexiblen Syntax Eigenschaften der Shell, von *awk* und von *sed* vereint. Es existiert eine Interpreter-Implementierung und ein experimenteller Compiler. Es gibt eine Schnittstelle zum Betriebssystem. Seit einigen Jahren existieren Mechanismen zur Erweiterung durch Module und objektorientierte Techniken.

picture Mit diesem Begriff bezeichnen Fluglotsen ihr mentales Bild der Verkehrssituation.

Pseudopilot ↑Geisterpilot

Python Moderne objektorientierte ↑Skriptsprache, die in einer Interpreter-Implementierung vorliegt. Es existieren vielfältige Module für den Einsatz üblicher Mechanismen, wie z.B. für die Benutzung regulärer Ausdrücke, von Internetprotokollen oder betriebssystemnahen Systemaufrufen. Es gibt vordefinierte Datentypen für Zahlen, Zeichenketten, Tupel, Listen und ↑Dictionaries. Python unterstützt die Arbeit an großen Projekten durch Mechanismen der Objektorientierung, Modularisierung und Paketbildung.

Rahmenarchitektur ↑Framework

Rufzeichen Eindeutige Kennung jedes Flugzeuges. Sie besteht aus wenigen Buchstaben und Ziffern.

scheme ↑LISP-ähnliche funktionale Programmiersprache, die in unterschiedlichen Interpreter-Implementierungen vorliegt. Es gibt scheme-Libraries. Sie können zu einem Programm hinzugelinkt werden, so daß das Programm mit scheme-Skripten erweitert werden kann. Daher wird scheme als ↑Skriptsprache betrachtet.

Skriptsprache Klasse von Programmiersprachen. Ihre Programme werden interpretiert. Skriptsprachen sind für die Einbettung in andere Programme konzipiert. Bei der Einbettung werden wichtige Funktionen des Programms dem Funktionsumfang der Skriptsprache hinzugefügt, so daß das um die Skriptsprache erweiterte Programm mit Skripten, also kurzen Programmen, die jeweils eine neue begrenzte Funktionalität durch die Komposition vorhandener hinzufügen, erweitert und konfiguriert werden kann. Skriptsprachen erlauben auch die Verwendung von *stand-alone* (nicht eingebetteten) Skript-Interpretern. Skript-Interpreter können durch Hinzufügen externer Module um Funktionen erweitert werden. Allen Skriptsprachen gemeinsam ist eine große Ausdrucksmächtigkeit, wofür mächtige Typen wie z.B. ↑Dictionaries und eine große Auswahl vorgefertigter Funktionen verfügbar sind. Die Entwicklung von Programmen geschieht in Skriptsprachen z.T. wegen ihrer Mächtigkeit und z.T. wegen des Wegfalls von Compilieren/Linken in kürzerer Zeit. Sie werden deshalb oft zum Prototyping und zur Rechner-Administration eingesetzt.

SOAR *state, operator, apply and result*. Wie ↑ACT-R eine Theorie und Implementierungsumgebung für die ↑kognitive Modellierung. Sie formalisiert kognitive Vorgänge,

insbesondere Problemlösen, in einem Produktionensystem als fortwährende Transformation von Zuständen, die dem deklarativen Wissen entsprechen, mit Operatoren, die das prozedurale Wissen darstellen. Lernen wird als Verknüpfung von Operatoren abgebildet.

Socket Bidirektionaler Kommunikationskanal zwischen zwei Prozessen auf demselben oder unterschiedlichen Rechnern in einem Netzwerk. Als Transportprotokoll wird fast ausschließlich TCP/IP, ein Internet-Protokoll, verwendet.

Squawk Vierstellige Oktalzahl, die jedem Flugzeug von der Flugsicherung angewiesen wird. Sie wird von einem Transponder im Flugzeug an die Flugsicherung gesendet. Durch eine flugsicherungsinterne Verknüpfung von Squawk und ↑ Callsign kann das ↑ Rufzeichen im Label des Radarbildes angezeigt werden.

SWIG *simplified wrapper and interface generator*. Software-Entwicklungs-Werkzeug, mit dem Programme oder Bibliotheken, die mit C, C++ oder Objective C entwickelt wurden, mit ↑ Skriptsprachen wie ↑ Perl, ↑ Tcl/Tk oder ↑ Python verbunden werden können.

Tcl/Tk *tool command language/toolkit*. ↑ Skriptsprache, deren Design auf bestmögliche Einbettung und Erweiterung ausgelegt ist. Die interne Verarbeitung ist ähnlich der von ↑ LISP, wobei jedoch als Basisrepräsentation Strings verwendet werden. Tk ist ein X-Windows Toolkit, also eine Sammlung von ↑ Widgets, das in tcl integriert wurde und die Entwicklung grafischer Oberflächen unter X11 vereinfacht. Tcl/Tk ist inzwischen von UNIX-Plattformen nach Windows und MacOS portiert worden.

UML *unified modelling language*. Methode und Notation für die objektorientierte Modellierung (Analyse und Design). UML ist die Nachfolgerin von den Methoden von Booch, OMT und OOSE, die sie in einer einheitlichen Notation vereint.

UTC *unified theory of cognition*. ↑ Architektur zur kognitiven Modellierung, mit der ein möglichst weiter Bereich kognitiver Phänomene, z.B. Lernen, Problemlösen und Wahrnehmen, adäquat erklärt werden kann.

Widget *windowing gadget*. Grafisches Funktions-, Eingabe- oder Anzeigeelement in fensterorientierten Software-Benutzungsoberflächen, z.B. Button, Slider, Menü.

Literaturverzeichnis

[1] Aasmann, J. (1995). *Modelling Driver Behaviour In Soar*. Leidschendam: KPN Research.

[2] Albayrak, S. & Bussmann, S. (1993). Kommunikation und Verhandlungen in Mehragenten-Systemen. In: J. Müller (Hrsg.), *Verteilte Künstliche Intelligenz. Methoden und Anwendungen.* Mannheim: BI-Wissenschaftsverlag. 55 – 81.

[3] Alhir, S.S. (1998). *UML in a Nutshell*. Cambridge: O'Reilly.

[4] Anderson, J.R. (1993). *Rules of the Mind*. Hillsdale, N.J.: Erlbaum.

[5] Anderson, J.R. & Lebiere, C. (1998). *Atomic Components of Thought*. Hillsdale, N.J.: Erlbaum.

[6] Amaldi, P. & Leroux, M. (1995). Selecting Relevant Information in a Complex Environment: the Case of Air Traffic Control. In: L. Norros (eds), *5th European Conference on Cognitive Science Approaches to Process Control (VTT Automation, Finland)*. 89 – 98.

[7] de Barros Costa, E., Lopes, M.A. & Fernenda, E. (1995). Mathema: A Learning Environment Based on a Multi-Agent Architecture. In: J. Wainer (ed), *Advances in Artificial Intelligence, 12^{th} Brazilian Symposium on Artificial Intelligence*, SBIA'9. Berlin: Springer. 141 – 150.

[8] Bass, E.J. (1998a). Architecture for an Intelligent Instructor Pilot Decision Support System. Accepted to the *1998 IEEE Conference on Systems, Man, and Cybernetics*, San Diego, October 11 – 14, 1998. Verfügbar im Internet unter `http://www.searchtech.com/articles/smc98.htm` (letzter Zugriff: 29.3.1999).

[9] Bass, E.J. (1998b). Towards an Intelligent Tutoring System for Situation Awareness Training in Complex, Dynamic Environments. In: B.P. Goettl, H.M. Halff, C.L. Redfield & V.J. Shute (eds), *Intelligent Tutoring Systems. Proceedings of the 4^{th} International Conference, ITS'98*, San Antonio, Texas, USA, August 1998. Berlin: Springer. 26 – 35.

[10] Bass, E.J., Baxter, G.D. & Ritter, F.E. (1995). Creating Models to Control Simulations: A Generic Approach. *AISB Quarterly 93*, 18 – 25.

[11] Bass, E.J., Ernst-Fortin, S.T. & Duncan, P.C. (1998). An Intelligent Debriefing System for Situation Awareness Training. In: *Proceedings of the Eleventh Florida Artificial Intelligence Research Symposium (FLAIRS 98)*, May 17 –20, 1998. Menlo Park: AAAI Press. 168 – 172.

[12] Bass, E.J., Zenyuh, J.P., Small, R.L. & Fortin, S.T. (1996). A Context-based Approach to Training Situation Awareness. In: *Proceedings of HICS'96 — Third Annual Symposium on Human Interaction with Complex Systems, Los Alamitos: IEEE Computer Society*. 89 – 95. Verfügbar im Internet unter `http://www.searchtec.com/articles/hics96.htm` (letzter Zugriff: 29.3.1999).

[13] Baxter, G.D. & Ritter, F.E. (1997). Model-Computer Interaction: Implementing the Action Perception Loop for Cognitive Models. In: D. Harris (ed), *Engineering Psychology and Cognitive Ergonomics II. Job Design and Product Design*. Hants, England: Ashgate. 215 – 222.

[14] Beck, J., Stern, M. & Haugsjaa, E. (1996). Applications of AI in Education. *Crossroads 3* (1). Verfügbar im Internet unter `http://www.acm.org/crossroads/xrds3-1/aided.html` (letzter Zugriff: 24.3.1999).

[15] Benel, R.A. & Benel, D.C.R. (1998). A Systems View of Air Traffic Control. In: M.W. Smolensky & E.S. Stein (eds), *Human Factors in Air Traffic Control*. San Diego: Academic Press. 17 – 63.

[16] Bierwagen, T. (1996). *Programmsystem EnCoRe-PLuS: Über die Möglichkeiten der programmierbaren Luftraumsimulation*. ZMMS-Forschungsbericht 96-2, Zentrum Mensch-Maschine-Systeme, Technische Universität Berlin.

[17] Bierwagen, T., Eyferth, K., Helbing, H. & Kolrep, H. (1995). *Report on the Work Carried out 15.9.1992 to 14.9.1994*. Forschungsbericht 95-1, Institut für Psychologie, Technische Universität Berlin.

[18] Bierwagen, T., Eyferth, K. & Niessen, C. (1997). *Bericht über die Arbeit des DFG-Projektes „Modellierung von Fluglotsenleistungen in der Streckenflugkontrolle" vom 15.9.1994 bis zum*

14.9.1996. ZMMS-Forschungsbericht 97-2, Zentrum Mensch-Maschine-Systeme, Technische Universität Berlin.

[19] Booch, G. (1991). *Object-Oriented Design with Applications*. Redwood City, CA: Benjamin/Cummings.

[20] Broach, D. & Manning, C. (1998). Issues in the Selection of Air Traffic Controllers. In: M.W. Smolensky & E.S. Stein (eds), *Human Factors in Air Traffic Control*. San Diego: Academic Press. 237 – 271.

[21] Brusilovsky, P., Specht, M., & Weber, G. (1995). Towards Adaptive Learning Environments. In: F. Huber-Wäschl, H. Schauer & P. Widmayer (Hrsg.), *Herausforderungen eines globalen Informationsverbundes für die Informatik. GISI 95*. Berlin: Springer-Verlag. 322 – 329.

[22] Chappell, A.R., Crowther, E.G., Mitchell, C.M. & Govindaraj, T. (o. J.). *The VNAV Tutor: Addressing a Mode Awareness Difficulty for Pilots of Glass Cockpit Aircraft*. Online-Dokument. Verfügbar im Internet unter `http://olias.arc.nasa.gov/publications/mitchell/vnav/vnav.html` (letzter Zugriff: 29.3.1999).

[23] Cooper, R. & Fox, J. (1998). COGENT: A visual design environment for cognitive modeling. *Behavior Research Methods, Instruments & Computers 30*, 553 – 564.

[24] Cooper, R. & Shallice T. (1995). Soar and the Case For Unified Theories of Cognition. *Cognition 55*, 115 – 149.

[25] Cooper, R., Yule, P., Fox, J. & Sutton, D. (1998) COGENT: An environment for the development of cognitive models. In: U. Schmid, J.F. Krems, & F. Wysotzki (eds), *Mind Modelling — A Cognitive Science Approach to Reasoning, Learning and Discovery*. Lengerich: Pabst Science Publishers. 55 – 82.

[26] Corker, K., Pisanich, G. & Bunzo, M. (1997). A Cognitive System Model for Human/Automation Dynamics in Airspace Management. Presented at *1st U.S.A./Europe Air Traffic Management R&D Seminar*, Saclay, France. Verfügbar im Internet unter `http://atm-seminar-97.eurocontrol.fr/corker.htm` (Letzter Zugriff 3.3.1999).

[27] Corker, K.M. & Smith, B.R. (1993). An Architecture and Model for Cognitive Engineering Simulation Analysis: Application to Advanced Aviation Automation. Presented at *AIAA Conference on Computing in Aerospace*, San Diego, CA. Verfügbar im Internet unter `http://george.arc.nasa.gov/af/aff/midas/www/AIAA_Final.txt` (Letzter Zugriff 3.3.1999).

[28] DFS (o.J.). *Computer Based Training Programs for Air Navigation Services*. Broschüre: DFS Akademie, CBT Team, Bernd Fischer, Paul-Ehrlich-Str. 37–39, D-63225 Langen.

[29] Endsley, M.R. & Smolensky, M.W. (1998). Situation Awareness in Air Traffic Control: The Picture. In: M.W. Smolensky & E.S. Stein (eds), *Human Factors in Air Traffic Control*. San Diego: Academic Press. 115 – 154.

[30] EUROCONTROL (1999). *CBT. Computer Based Training*. Broschüre: Computer-Based Training, EUROCONTROL, Institute of Air Navigation Services, Email: `mailto:training.dev@eurocontrol.be`.

[31] Fisher, S.G. & Kulick, I. (1998). Air Traffic Controller Training: A New Model. In: M.W. Smolensky & E.S. Stein (eds), *Human Factors in Air Traffic Control*. San Diego: Academic Press. 273 – 298.

[32] Folk, M. (1998). *A White Paper. HDF as an Archive Format: Issues and Recommendations*. Online-Dokument in der Version vom 27.1.1998. Verfügbar im Internet unter `http://hdf.ncsa.uiuc.edu/archive/hdfasarchivefmt.htm` (Letzter Zugriff 22.7.1999).

[33] Free Software Foundation (1999). *Guile: Duct Tape For Bits. Project GNU's Extension Language.* Online-Dokument in der Version vom 1. Juli 1999. Verfügbar im Internet unter `http://www.gnu.org/software/guile/guile.html` (letzter Zugriff: 6.9.1999).

[34] Fricke M. (1998). *Mensch-Maschine-Systeme in kooperativen Systemen der Flugsicherung und Flugführung. Management von potentiellen Luftfahrzeugkonflikten durch einen kooperativen Planungsprozeß.* Zwischenbericht an die Deutsche Forschungsgemeinschaft. Institut für Luft- und Raumfahrt, Technische Universität Berlin.

[35] Gopher, D. (1993). The Skill of Attention Control: Acquisition and Execution of Attention Strategies. In: D.E. Meyer & S. Kornbloom (eds), *Attention and Performance* XIV*: Synergies in Experimental Psychology, Artificial Intelligence, and Cognitive Neuroscience*. Cambridge, MA: The MIT Press. 299 – 322.

[36] Goucher, G. W. & G. J. Mathews (1994). *A Comprehensive Look at CDF*. Technical Report NSSDC/WDC-A-R&S 94-07, NASA/Goddard Space Flight Center.

[37] Han, D. & Mathews, J. (1998). *NSSDC's CDF Homepage*. Online-Dokument in der Version vom 13.11.1998. Verfügbar im Internet unter `http://nssdc.gsfc.nasa.gov/cdf/cdf_home.html` (Letzter Zugriff 21.7.1999).

[38] Hitchcock, L. (1998). Air Traffic Control Simulation: Capabilities. In: M.W. Smolensky & E.S. Stein

(eds), *Human Factors in Air Traffic Control*. San Diego: Academic Press. 327 – 340.

[39] Imetrix (1998). *Imetrix RovMentor. ROV Pilot Training System*. Online-Dokument. Verfügbar im Internet unter `http://www.imetrix.com/RovMentor/RovMentor.html` (letzter Zugriff: 29.3.1999).

[40] Johnson-Laird, J.E., Newell, A. & Rosenbloom, P.S. (1987). Soar: An Architecture for General Intelligence, *Artificial Intelligence 33*, 1 – 64.

[41] Kenney, P.J. & Saito, T. (1994). *Results of a Survey ob the Use of Virtual Environment Technology in Training NASA Flight Controllers for the Hubble Space Telescope Servicing Mission*. Online-Dokument in der Fassung vom 21.6.1994. Verfügbar im Internet unter `http://www.vetl.uh.edu/Hubble/longpaper.html` (letzter Zugriff: 1.4.1999).

[42] Kieras, D.E, Meyer, D.E (1995). Predicting Human Performance in Dual-Task Tracking and Decision Making with Computational Models Using the EPIC Architecture. In: D.S. Alberts, D. Buede, T. Clark, R. Hayes, J. Hoffmann, W. Round, S. Starr & W. Vaughan (eds), *Proceedings of the International Symposium on Command and Control Research and Technology*. Washington D.C.: National Defense University. 314 – 325.

[43] Kieras, D.E., Wood, S.D. (1997). Predictive Engineering Models Based on the EPIC Architecture for a Multimodal High-Performance Human-Computer Interaction Task. *ACM Transactions on Computer-Human Interaction 4* (3), 230-275.

[44] Kierwan, B. (1998). HF Techniques in the NATS ATM System Development Process. In: EUROCONTROL. *The third EUROCONTROL Human Factors Workshop. Integrating Human Factors into the life cycle of ATM systems*. 7–9 October 1998, Luxembourg.

[45] Leutner, D. (1992). *Adaptive Lehrsysteme. Instruktionspsychologische Grundlagen und experimentelle Analysen*. Weinheim: Psychologie-Verlag-Union (Fortschritte der psychologischen Forschung; 13).

[46] Luft, C. (1998). Flexible Struktuen für ein modernes Berufsbild. Neues Ausbildungskonzept „DATS“. *transmission 6/98*, 6 – 9.

[47] Lusti, M. (1992). *Intelligente Tutorielle Systeme. Einführung in wissensbasierte Lernsysteme*. München: Oldenbourg (Handbuch der Informatik; Bd. 15.4).

[48] Meyer, D.E., Kieras, D.E. (1992). The PRP Effect: Central Bottleneck, Perceptual-Motor Limitations, or Task Strategies? Präsentiert auf dem Meeting of the Psychonomic Society, St. Louis, MO.

[49] Meyer, D.E. & Kieras, D.E. (1997a). A Computational Theory of Executive Processes and Multiple-Task Performance: Part 1. Basic Mechanisms. *Psychological Review 104* (1), 3 – 65.

[50] Meyer, D.E. & Kieras, D.E. (1997b). A Computational Theory of Executive Processe s and Multiple-Task Performance: Part 2. Accounts of Psychological Refractory-Period Phenomena. *Psychological Review 104* (4), 749 – 791.

[51] Meyer, D.E., Kieras, D.E., Lauber, E., Schumacher, E.H., Glass, J., Zurbriggen, E., Gmeindl, L. & Apfelblatt, D. (1995). Adaptive Executive Control: Flexible Multiple-Task Performance Without Pervasive Immutable Response-Selection Bottlenecks. *Acta Psychologica 90*, 163 – 190.

[52] Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, Massachusetts: Harvard University Press.

[53] Niessen, C., Eyferth, K. & Bierwagen, T. (1997). Modelling Cognitive Processes of Experienced Air Traffic Controller. In: S. Bagnara, E. Hollnagel, M. Mariani & L. Norros (eds), *Sixth European Conference on Cognitive Science Approaches to Process Control, Time and Space in Process Control*, 23.9. – 26.9.1997 Baveno, Italy.

[54] Niessen, C.; Eyferth, K. & Bierwagen, T. (1998). Modelling Cognitive Processes of Experienced Air Traffic Controllers. Submitted to the *Ergonomics Special Issue: Time and Space in Process Control*.

[55] Niessen, C. & Leuchter, S. (in Vorbereitung). Konzeption und Implementierung eines dynamischen Situationsmodells am Beispiel der Fluglotsentätigkeit. In: K-.P. Timpe & H. Kolrep (Hrsg.) *Mensch-Maschine-Systemtechnik*.

[56] Niessen, C., Leuchter, S. & Eyferth, K. (1998). A Psychological Model of Air Traffic Control and Its Implementation. In: F. Ritter & R. Young (eds), *Proceedings of the Second European Conference on Cognitive Modelling (ECCM-98)*. Nottingham Press. 104 – 111.

[57] Opwis, K. (1992). *Kognitive Modellierung: zur Verwendung wissensbasierter Systeme in der psychologischen Theoriebildung*. Bern: Huber.

[58] O'Reilly & Associates (1999). *Open Source: kurz und gut*. Köln: O'Reilly.

[59] Paiva, A. (1995). *Towards a Consensus on the Communication Between User Modeling Agents and Application Agents*. Online Dokument in der Version vom 30.11.1995. Verfügbar im Internet unter `http://www.cbl.leeds.ac.uk/~amp/MyPapers/kc.ps` (letzter Zugriff 13.9.1999).

[60] Pree, W. (1997). *Komponentenbasierte Softwareentwicklung mit Frameworks*. Heidelberg: dpunkt Verlag.

[61] Pryce, N. (1999). *Pattern: Glue Code*. Online-Dokument in der Version vom 24. März 1999. Verfügbar im Internet unter `http://scorch.doc.ic.ac.uk/~np2/patterns/scripting/glue-code.html` (letzter Zugriff: 6.9.1999).

[62] Rideau, F.-R. (1999). *Metaprogramming and Free Availability of Sources*. Online-Dokument in der Version vom Januar 1999. Verfügbar im Internet unter `http://www.tunes.org/~fare/articles/ll99/index.en.html` (letzter Zugriff: 8.9.1999).

[63] Riede, C. (1997). *GATS — Generic Air Traffic Simulation*. Technischer Bericht, Institut für Luft- und Raumfahrt, Technische Universität Berlin.

[64] Schumann, J. & Kulessa, M. (1997). Ein interdisziplinäres Konzept zur Untersuchung des Einflusses von Automatisierung auf die Mensch-Maschine-Interaktion in Flugsicherung und Flugführung. In: K.-P. Gärtner (Hrsg.), *Menschliche Zuverlässigkeit, Beanspruchung und benutzerzentrierte Automatisierung*. DGLR-Bericht 97-02. Bonn: Deutsche Gesellschaft für Luft- und Raumfahrt. 29 – 40.

[65] Seifert, K. (1997). *Evaluierung der Informationskontrolle mittels Maskierung duch Messung von Blickbewegungen*. Diplomarbeit, Institut für Psychologie, Technische Universität Berlin.

[66] Sherry, L., Kelly, J., McCrobie, D., Feary, M., Alkin, M., Polson, P., Hynes, C. & Palmer, E. (o. J.). *A Framework for the Design of Intentional Systems in Support of Cooperative Human-Machine Systems*. Online Dokument. Verfügbar im Internet unter `http://olias.arc.nasa.gov/publications/feary/osupap1/osupap1.html` (letzter Zugriff: 29.3.1999).

[67] Strube, G. (1996). Kognitive Modellierung. In: G. Strube (Hrsg.), *Wörterbuch der Kognitionswissenschaft*. Stuttgart: Klett-Cotta. 407 – 408.

[68] Upmeyer, A. & Günther, A. (1994). Der Einsatz von Intelligenten Tutoriellen Systemen beim Fernlernen. In: A. Günther (Hrsg.), *Computerbasiertes Training und Fernlernen*. Berlin: Köster. 53 – 66.

[69] Wickens, C.D., Mavor, A.S. & McGee, J.P. (eds.) (1997). *Flight To the Future. Human Factors in Air Traffic Control*. Washington: National Academy Press.

[70] Widowski, D. & Eyferth, K. (1986). Comprehending and Recalling Computer Programs of Different Structual and Semantic Complexity by Experts and Novices. In: H.P. Willumeit (ed), *Human Decision Making and Manual Control*. Amsterdam: Elsevier. 267 – 275.