Karlsruher Institut für Technologie

Fraunhofer IOSB

Einsatz künstlicher Intelligenz in adaptiven Lernsystemen

BACHELORARBEIT

KIT - KARLSRUHER INSTITUT FÜR TECHNOLOGIE
FRAUNHOFER IOSB - FRAUNHOFER-INSTITUT FÜR OPTRONIK, SYSTEMTECHNIK UND BILDAUSWERTUNG

Julius Busch

31. August 2018

Erster Gutachter:	Prof. Dr.-Ing. Jürgen Beyerer
Zweiter Gutachter:	Prof. Dr.rer.nat. H. Steusloff
Betreuender Mitarbeiter:	Dipl.-Inf. Alexander Streicher

Interaktive Echtzeitsysteme (IES)

Erklärung der Selbstständigkeit

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Karlsruhe, den 31. August 2018 Julius Busch

Kurzfassung

Eines der aktiven E-Learning Forschungsfelder befasst sich mit der individuellen Wissensvermittlung. Lernende sollen hierbei im Idealfall die gleiche adaptive Unterstützung zur Überbrückung von Verständnisengpässen genießen, wie es ihnen ein menschlicher Tutor ermöglichen könnte. Intelligente Tutoring Systeme (ITS) versuchen, dieses Ziel mittels Techniken der künstlichen Intelligenz zu erreichen. Die Entwicklung derartiger Systeme ist aufgrund der benötigten Modellierung des Expertenwissens der jeweiligen Domäne kostspielig. In dieser Arbeit wird die Erweiterung schon bestehender webbasierter Lernsysteme zu einem ITS aufgezeigt. Hierbei wurde der Fokus auf die Analyse von Benutzerinteraktion mithilfe der Modellierung menschlichen Denkens in kognitiven Architekturen gelegt. Als Grundlage diente das am Fraunhofer IOSB entwickelte Serious Game EXercise TRAiner (EXTRA). In dem Spiel wurde ein Adapter integriert, welcher nach aktuellen Interooperabiliätstandards den Activity Stream in xAPI Statements generiert. Die in dieser Arbeit entwickelte User Model Artificial Intelligence (UMAI) nutzt diese Daten wiederum zur dynamischen Generierung eines ACT-R Modells, um Aktionen auf ihre Frequenz und das zeitliche Vorkommen hin zu analysieren. Hierbei werden für jede Aktion deklarative Chunks in ACT-R modelliert. Die während einer Simulation produzierten Aktivitätsdaten dieser Chunks werden in einem Front-End visualisiert. Dies kann Lehrenden helfen, den Schüler adaptiv zu unterstützen. Entwickler können die Daten nutzen, um die Theorien und Funktionsweisen von ACT-R besser zu verstehen.

Abstract

An active e-learning research field is devoted to the individual transfer of knowledge. Ideally, learners should enjoy the same adaptive support to overcome understanding bottlenecks as a human tutor could provide. Intelligent Tutoring Systems (ITS) try to achieve this goal by means of various techniques of artificial intelligence. The development of such systems is a costly undertaking due to the need to cognitivly model the expertise and knowledge of each domain. The aim of this work is to demonstrate the feasibility of extending already existing web-based learning systems to an ITS, which can support students individually. Here, the focus has been placed on the analysis of user interaction by modeling human thinking in cognitive architectures. The underlying learning system was the Serious Game EXercise Trainer (EXTRA), developed at Fraunhofer IOSB. In the game the user generates xAPI data, which is forwarded to an LRS. The User Model Artificial Intelligence (UMAI), developed in this work, uses this data in turn to dynamically generate an ACT-R model in order to analyze actions according to their frequency and temporal occurrence. For each action declarative chunks are modeled in ACT-R. The activity data of these chunks produced during a simulation is visualized in a frontend. This can help teachers to adaptively support the student. Developers can use the data in order to better understand how ACT-R works.

Inhaltsverzeichnis

1

Einführung

Das “Klassenzimmer der Zukunft” könnte davon profitieren, dass jeder Lernende einen eigenen digitalen Lehrer zugewiesen bekommt. In ersten Versuchen auf dem Gebiet wurden zunächst Systeme entwickelt, die Anweisungen in kurzen Segmenten präsentierten und während des Unterrichts häufig Fragen stellten. Dabei wurde unmittelbares Feedback zu Antworten der Lernenden gegeben [Kul16]. Die adaptive Unterstützung operierte jedoch noch auf Basis trivialer Wissensrepräsentation. So wurden Fehlerkataloge und fest kodierte Hilfestellungen für bestimmte Aktionen eines Nutzers eingesetzt [Mil07]. Neuere Systeme, in der Fachliteratur intelligente Tutoringsysteme (ITS) genannt, versuchen individuellere adaptivere Wissensvermittlung mithilfe künstlich intelligenter Modelle umzusetzen. Ein adaptives Bildungssystem muss Personalisierung auf die spezifischen Bedürfnisse, Kenntnisse und den Hintergrund jedes einzelnen Studenten hin zur Verfügung stellen, da Studenten auf unterschiedliche Art und Weise lernen [JA13]. Das hoch gesteckte Ziel eines digitalen Tutors ist eine vom menschlichen Experten nicht zu unterscheidende Interaktion mit dem Schüler, um bestmöglich auf ihn eingehen zu können. Kognitive Architekturen sind Anwärter Artificial General Intelligence (AGI) umzusetzen, was im Groben bedeutet, menschliches Denken zu simulieren. Bezogen auf den digitalisierten Tutor können kognitive Architekturen dabei helfen das ITS mehr und mehr auf die Ebene menschlicher Kognition zu bringen. Für die Verbesserung dieser Systeme ist es wichtig zu erforschen, welche künstlich intelligenten Modelle den größten Fortschritt im Hinblick auf den im Vorigem motivierten digitalisierten Tutor bieten können. Intelligente Tutoringsysteme sind teuer in der Entwicklung [Cor97].Deshalb stellt sich bezogen auf die Komplexität dieser System die Frage, wie Entwickler schneller zu akzeptablen Ergebnissen kommen können. Diese Arbeit widmet sich diesen Fragestellungen und Problemen. Dabei wird vor allem der Einsatz kognitiver Architekturen untersucht.

1.1 Motivation

Wie einführend beschrieben ist die Qualität der Adaptivität eines Lernsystems direkt abhängig von der Ausprägung der verwendeten künstlichen Intelligenz. Fraglich ist, wie aufwändige Entwicklung komplexer intelligenter Tutoringsysteme (ITS) umgangen werden kann, ohne dabei schlechtere Ergebnisse zu erzielen. Des Weiteren ist zu evaluieren, welche Intelligenz speziell im Hinblick auf Serious Games geeignet ist, den Spieler adaptiv zu unterstützen. Dabei ist von grundsätzlicher Bedeutung, wann ein adaptives System eingreift, um beispielsweise das Vergessen erlernter Inhalte zu verhindern. Kognitive Architekturen könnten sich hierfür eignen, da sie menschliches Denken

simulieren und somit direkt auf die kognitiven Aspekte der Interaktion eingehen. Für die in Abschnitt 1.3 vorgestellte ELAI-Architektur wird der Einsatz kognitiver Architekturen in Betracht gezogen. Vor dem Hintergrund des hohen Aufwands der Modellierung des gesamten Wissens einer Domäne in kognitiven Architekturen, ist zu untersuchen, ob sich diese zum direkten Einsatz der Analyse von Benutzerinteraktion eignen.

1.2 Ziele und Abgrenzung der Arbeit

Diese Arbeit informiert über die verschiedenen Forschungsergebnisse auf dem Gebiet der digitalen personalisierten Wissensvermittlung. Um den Einstieg in das Themengebiet zu erleichtern, wird umfassend aufgezeigt, wie kognitive Architekturen in einem ITS eingesetzt werden können. Dabei wird zuzüglich anhand einer Beispielarchitektur eine Machbarkeitsstudie durchgeführt, welche ein Grundgerüst eines ITS auf Basis eines schon bestehenden Lernsystems realisiert. Wie in Abschnitt 1.3 beschrieben, orientiert sich die Idee des Projekts an der ELAI-Architektur. In der Arbeit wurden bereits erforschte Techniken auf ihre Eignung, Nutzen und Umsetzbarkeit untersucht. Um zukünftigen Entwicklungsaufwand zu reduzieren, wurde eine Demonstrationsarchitektur entwickelt, die als Startvorlage dienen kann. So sollen beispielsweise weitere Lernanalysetechniken eingebaut werden können oder weitere künstlich intelligente Modelle zur Analyse von Benutzerinteraktion integriert werden. Außerdem wird in einer Nutzwertanalyse aufgezeigt, welche kognitiven Architekturen für derartige Projekte geeignet sind, um Entwicklern ähnlicher zukünftiger Projekte eine Orientierung zu geben, welche Richtung sie in ihrer Arbeit einschlagen können.

1.3 Projektumfeld

Diese Arbeit baut auf dem am Fraunhofer Institut für Optronik, Systemtechnik und Bildauswertung (IOSB) in der Abteilung für “Interoperabilität und Assistenzsysteme” entwickelten E-Learning Artificial Intelligence (ELAI) auf. ELAI ist eine Rahmenarchitektur, welche interoperable Adaptivität für beigefügte Game-Engines, Spiele oder auch Lernsysteme und Simulationen ermöglicht [Str17].

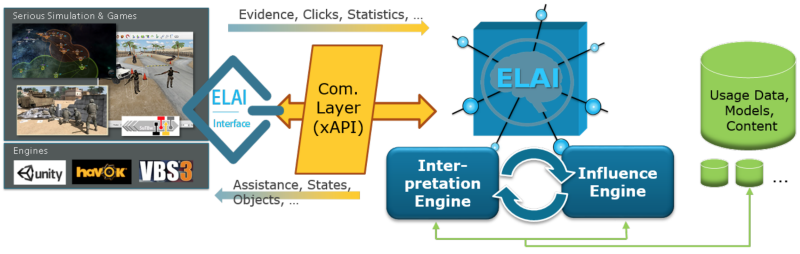

Abbildung 1.1: ELAI Architektur [Str17]

Als Kommunikationsinstrument und gleichzeitiges Datenformat zur Aufzeichnung der Benutzerinteraktionen benutzt ELAI das xAPI Protokoll, welches im Abschnitt 3.2 näher thematisiert wird. Wie in Abbildung 1.1 erkenntlich gemacht, steht im Kern das ELAI-Kontrollmodul, das zum Einen aufgezeichnete Daten eines Lernsystemens analysiert und zum Anderen mögliche unterstützende Eingriffe und Hilfestellungen bestimmt. Um die Interaktionsdaten aufzeichnen zu können, gilt es

abhängig der jeweiligen Infrastruktur des verwendeten Systems, einen Adapter zu integrieren. Insgesamt entsteht auf diese Weise eine dezentralisierte Architektur, die es ermöglicht, adaptiv und personalisiert auf den Benutzer einzugehen.

Als grundlegendes Lernsystem diente das ebenfalls am Fraunhofer IOSB entwickelte Serious Game EXercise TRAiner (EXTRA) [Gun16]. EXTRA behandelt das Problem, dass Prozesse in großen multinationalen NATO Übungen für *Joint Intelligence, Surveillance and Reconnaissance* (JISR)[1] schwer verständlich sind. Verschiedenste Teilnehmer in unterschiedlichen Rollen und mit ungleichem Kenntnisstand müssen die Prozesse und den Informationsfluss zwischen den beteiligten heterogenen Hardwaresystemen und Software Anwendungen verstehen [Str16]. In EXTRA wird der Informationsfluss zwischen den einzelnen Einheiten in Spielszenarien übersetzt dargestellt. Hierbei sind einzelne Spielelemente repräsentativ für die realen involvierten Informationsverarbeitungseinheiten der jeweiligen Prozesse.

1.4 Aufbau und Gliederung

Im ersten Kapitel werden zwei Varianten der Modellierung des Wissens einer Domäne aufgezeigt. Adaptive Lernsysteme bauen auf dieser Modellierung auf und nutzen sie, um Aktionsschritte der Schüler als richtig oder falsch zu bewerten.

Darauf werden dem Leser die für die Thematik notwendigen Grundlagen näher aufgezeigt. Hierbei wurde wie einleitend schon erwähnt der Fokus auf kognitive Architekturen gerichtet.

Im nächsten Kapitel ist die Nutzwertanalyse thematisiert, in welcher eine für dieses Projekt geeignete kognitive Architektur ausgewählt wurde.

Auf Basis dieser wurden einzelne exemplarische Aspekte von EXTRA modelliert und eine nach der ELAI-Architektur nachempfundene Webarchitektur implementiert. Die dafür entworfenen Konzepte sind im Entwurfskapitel dokumentiert.

Details des Ergebnisses sind im darauf folgenden Implementierungskapitel festgehalten.

In dem darauf folgenden Kapitel wird zunächst ein realer Anwendungsfall anhand von Systeminteraktion aufgezeigt. Anschließend werden die in dieser Arbeit erreichten Erkenntnisse diskutiert.

Abschließend wird die Arbeit im letzten Kapitel zusammengefasst und ein Ausblick gegeben.

1 US-Joint-Publication: `https://fas.org/irp/doddir/dod/jp2_0.pdf`

2

Varianten der Umsetzung adaptiver Lernsysteme

In diesem Kapitel werden die zwei bekanntesten Entwurfsmuster adaptiver Lernsysteme vorgestellt, die in der Fachliteratur auch intelligente Tutoring Systeme (ITS) genannt werden. Diese bilden den Stand der Forschung und Technik adaptiver Lernsysteme, die personalisierte individuelle Hilfestellung unterstützen. Beide Varianten basieren auf einer klassischen Struktur, welche das System in verschiedene Modelle untergliedert. Das Domänenmodell (engl. Domain-Model) repräsentiert das zu erlernende Wissen und das studentische Modell (engl. Student-Model) den Wissensstand des Lernenden. Auf diese Modelle wird in einem Tutorenmodell (engl. Tutor-Model) zugegriffen, um aus zusammengeführten Erkenntnissen Hilfestellungen für den Schüler herzuleiten. Diese Struktur ist in Abschnitt 3.4 detaillierter erklärt. Die Umsetzungsvarianten unterscheiden sich vor allem in der Modellierung des Wissens im Domain-Model. Im Folgenden wird zunächst die Modellierung der Domäne in kognitiven Architekturen thematisiert. Dieser Ansatz ist auf dem Forschungsgebiet am stärksten anerkannt, da das resultierende System auf die individuelle Kognition eines Studenten eingeht. Die Entwicklung derartiger Systeme ist jedoch im Vergleich zu der zweiten Varianten mit erhöhtem Aufwand verbunden. Hier wird die Domäne mit sogenannten Constraints modelliert, die Regeln in Form von Bedingungen darstellen. Der Einsatz künstlich intelligenter Modelle konzentriert sich im Gegensatz zur ersten Variante auf das Student-Model. In dieser Arbeit wurde keiner der beiden Ansätze verfolgt, da Ersterer für den Rahmen der Arbeit zu komplex war, der Fokus jedoch auf den Einsatz kognitiver Architekturen lag. Dies wird in Abschnitt 7.3 ausführlich diskutiert.

2.1 Kognitive Modellierung der Domäne

In der ersten Variante der Umsetzung von Adaptivität in Lernsystemen kommen kognitive Architekturen zum Einsatz, um das Wissen der Domäne zu modellieren. Im Folgenden werden Systeme vorgestellt, die auf diese Art und Weise konzeptioniert sind. Zuzüglich wird die automatisierte Generierung der Modellierung thematisiert.

Kognitive Architekturen in intelligenten Tutoringsystemen

In intelligenten Tutoringsystemen (ITS) wird meistens die ACT-R Architektur zur Modellierung der Domäne benutzt, welche in Abschnitt 5.1 vorgestellt wird [Mil07]. Im Kern dieser Modellierung

steht die Unterteilung des Wissens in deklarative und prozedurale Einheiten. Wie in Abschnitt 3.3 an einem Beispiel erklärt, stellt deklaratives Wissen atomare Fakten dar, während Produktionen einzelne zielorientierte Problemlösungsschritte einer größeren Aufgabe abstrahieren. Ziel der Modellierung ist es, menschliche Denkweisen der Herangehensweise an Probleme realitätsnah wiederzugeben [Ma14]. Die Produktionsregeln umfassen hierbei alle möglichen Lösungswege, die ein Schüler durchlaufen kann. Die grundlegende Technik kognitiver Tutoren wird in der Fachliteratur **Model-Tracing** genannt. Diese ermöglicht die Interaktion des Tutors mit dem Schüler in Echtzeit [Bev09]. Im Model-Tracing wird jede Aktion des Schülers zeitgleich in der kognitiven Architektur nachsimuliert. Anschließend kann die Vorgehensweise des Schülers durch den Vergleich mit dieser Simulation als richtig oder falsch eingestuft werden. Auf Basis dessen können folgende Hilfestellungen und Interaktionen unterstützt werden [And00]:

1. Falls der Schüler Hilfe anfragt kann passend zur Stelle des Problemlösungs-Prozesses individuelle Hilfe gegeben werden.

2. Sofern Schüler korrekte Lösungsstrategien anwenden, kann dies erkannt werden und die Aufgaben an das Wissensniveau des Schülers angepasst werden.

3. Wenn Schüler einen scheinbar erkennbaren Fehler begehen, ist es möglich einzugreifen und entsprechende Anweisungen zu geben.

Ein Beispiel für ein Model-Tracing-System ist der “Pump Algebra Tutor” (PAT), für den Problemlösungsstrategien linearer Algebra in ACT-R modelliert wurden [And00]. In dem Programm wurde der Fokus vor allem darauf gelegt, mehrere Repräsentationen der Problematiken zu erlernen [Bev09]. Wie in Abbildung 2.1 ersichtlich, dient ein Szenario-Bereich der Aufgabenstellung und Verfügungsstellung verschiedener Grafiken. In dem Worksheet-Bereich sollen Studenten Antworten und Ergebnisse zu diesen Fragen eingeben.

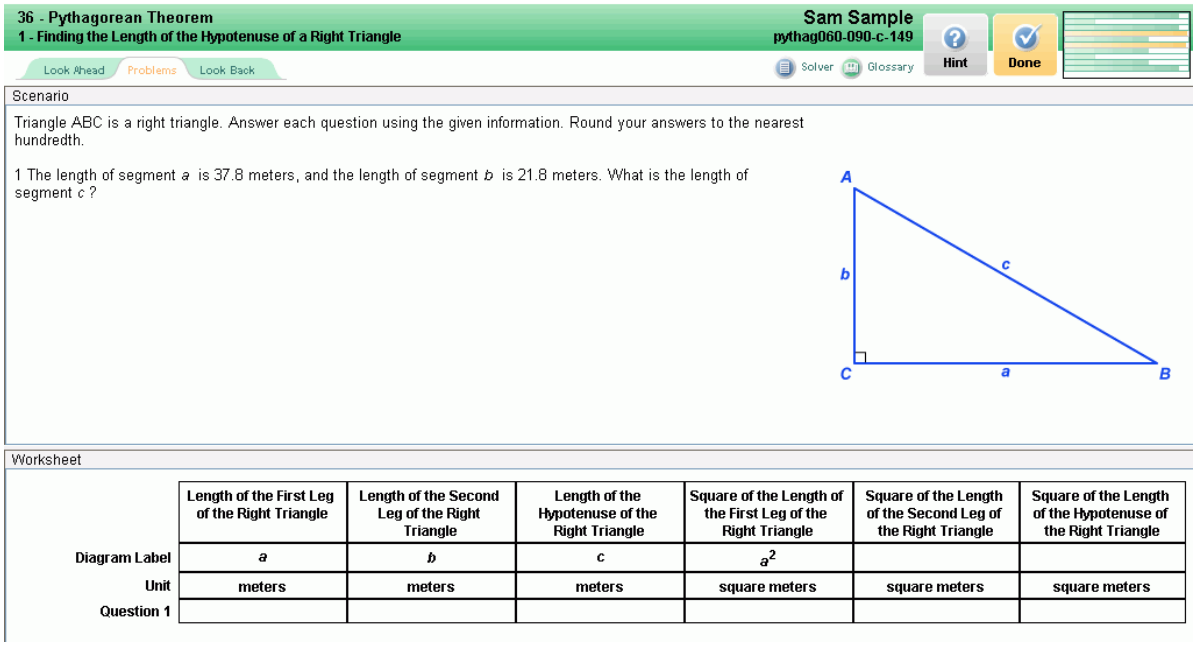

Abbildung 2.1: PAT Linear Algebra Benutzeroberfläche [Car08]

Des Weiteren können Inhalte mit dem Grapher-Werkzeug interaktiv erlernt werden, das in Abbildung 2.2 im rechten unteren Bereich gezeigt ist. In diesem Beispiel sollen Studenten die richtigen Punkte in dem Graph ablesen und anschließend im Worksheet eintragen. Im oberen rechten Bereich können Lernende eine Hilfestellung anfragen, die aus den vom Schüler getätigten Aktionen in der kognitiven Architektur errechnet wird. Darüber hinaus wird dem Schüler eine Bewertung seines Wissens angezeigt. Das im rechten oberen Eck der Abbildungen 2.1 und 2.2 angezeigte “Skilllevel” wird mithilfe von **Knowledge Tracing** berechnet. Hierbei wird für alle Produktionen nach jedem Schritt des Schülers wahrscheinlichkeitstheoretisch bestimmt, ob dieser die Regel beherrscht. Die Wahrscheinlichkeiten des so genannten “Mastery of Knowledge” werden in einem Hidden-Markov-Modell errechnet. In dem Modell wird angenommen, dass der Wissensstand für jede Produktion binär ist [Cor95]. Somit ist das studentische Modell in diesen Systemen darin konstituiert, Wahrscheinlichkeiten auf den Produktionssatz des kognitiv modellierten Domain-Models abzubilden.

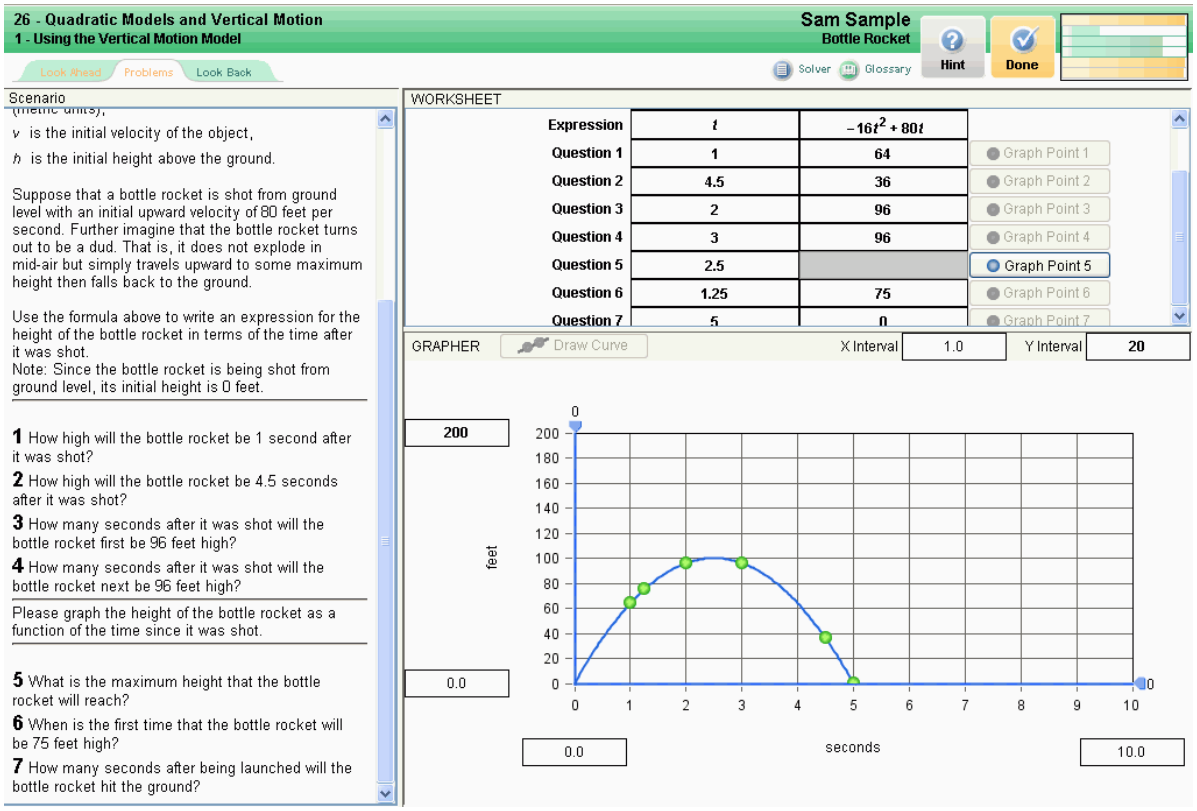

Abbildung 2.2: PAT Linear Algebra Benutzeroberfläche [Car08]

Kognitive Architekturen in Serious Games

In einem weiteren System wurde ACT-R eingesetzt, um einen digitalen Agenten zum Einsatz in Trainingssimulationen für den militärischen Einsatz in städtischen Gebieten (MOUT) zu entwickeln [Bes02]. In dem 3D basierten Serious Game wurde ein komplexer Produktionssatz modelliert, um das autonome Handeln des Agenten zu erreichen. In Abbildung 2.3 ist eine Beispielansicht der Umgebung gezeigt, in welcher der Agent gegen menschliche Gegner spielt.

Abbildung 2.3: Spieleransicht eines Agenten in der MOUT Simulation [Bes02]

Im Folgenden sind Beispiele des Produktionssatzes aufgeführt, nach welchem der Agent seine Aktionsschritte auswählt, falls ein Gegner in seinem Sichtfeld erscheint [Bes02]:

1. Wenn ein Feind in Sichtweite ist und es keinen Fluchtweg gibt, dann schieße auf den Feind.
2. Wenn ein Feind in Sichtweite ist und es einen Fluchtweg gibt, dann setze ein Ziel, um entlang dieser Route zu entkommen.
3. Wenn es ein Ziel gibt, entlang einer Route zu entkommen, und ein Feind in Sichtweite ist, schießen Sie auf den Feind und ziehen Sie sich entlang der Route zurück.

Die kognitive Architektur wurde in diesem System noch nicht um das Student-Model erweitert, wie es in dem zuvor vorgestellten Tutor der linearen Algebra umgesetzt wurde. Des Weiteren müsste hier der Produktionssatz so erweitert werden, dass alle Spielszenarien und Interaktionen abgedeckt sind, um die Leistung der Spieler einordnen zu können. Dies ist in Abschnitt 7.3 genauer thematisiert. Somit ist mit dieser Art von Modellierung noch keine Adaptivität garantiert. Jedoch kann mit der Modellierung veschiedener Spielszenarien autonome Spielweise erreicht werden.

Automatisierte Generierung der Modellierung

Wie einleitend motiviert, sind Entwickler adaptiver Lernsysteme, welche kognitive Architetkuren als Grundlage nehmen, mit der komplexen Modellierung des Domain-Models konfrontiert. Dabei stellt die Arbeit nicht nur eine quantitative Herausforderung dar. Eine gute Modellierung, beispielsweise in ACT-R, setzt Expertenwissen und Erfahrung voraus [Sal03]. Um diese Problematik zu behandeln, werden Systeme entwickelt, die von einer höheren Abstraktionsebene ausgehen und automatisiert

das komplexere Modell daraus entwerfen. Beispiele hierfür sind ACT-Simple und G2A, die auf dem Framework GOMS aufbauen [Sal03][SA05]. GOMS steht für ***G**oals, **O**perators, **M**ethods und **S**election rules*. Das Framework wurde dazu konzeptioniert Vorhersagen zu treffen, welche Methoden und Operationen Nutzer in digitalen Systemen anwenden, um bekannte Aufgaben zu erledigen [RO90].

In dieser Arbeit musste der Einsatz dieser Systeme ausgeschlossen werden, da die in den veröffentlichen Publikationen angegebenen Webseiten nicht mehr verfügbar sind.

2.2 Constraint-Modellierung der Domäne

Eine andere Art studentisches Wissen zu repräsentieren, ist das von Stellan Ohlsson entwickelte Constraint-Baised Modeling (CBM) [Ohl94]. Ohlsson erkannte die Komplexität der Entwicklung des Produktionssatzes der in Abschnitt 2.1 beschriebenen model-tracing Systeme. Für einige Domänen könnte es sogar unmöglich sein, diese auf passende Produktionsregeln abzubilden [Mit01]. Vor allem in geisteswissenschaftlichen Fächern soll vieles implizit erlernt werden, um eine subjektive Meinungsbildung zu vermitteln. Produktionen verlangen jedoch eine klare logische Struktur, die vor allem in mathematischen Fächern anzufinden ist [Ant03]. Die Theorie des CBM geht davon aus, dass es ausreicht, das Domänenwissen in Form der wichtigsten Grundlagen darzustellen, die alle Bedingungen an eine richtige Lösung abdecken. Ein ITS basierend auf CBM erfordert somit keine genaue Übereinstimmung der Expertenschritte zu den studentischen Aktionen und keine Modellierung aller fehlerhaften Lösungswege [Bev09]. Somit eignet sich diese Art von Modellierung vor allem für Systeme, in denen das zu vermittelnde Wissen nicht ausreichend beschrieben werden kann, Fehler nicht klar identifiziert werden können oder zu viele Möglichkeiten verschiedener Lösungswege existieren.

Die Idee des CBM stammt aus der von Ohlsson 1996 entwickelten “Theory of learning from performance errors” [Ohl96]. In dieser beschreibt Ohlsson, dass das Erlernen deklarativer Inhalte erst dann erfolgreich abgeschlossen ist, wenn Studenten während der Problemlösung ihre eigenen Fehler erkennen können. Schüler gehen demnach oft davon aus Aufgaben richtig gelöst zu haben, obwohl sie basierend auf dem deklarativen Wissen in einem gewissen “State” noch fehlerhaft handeln. Ohlsson beschreibt hier den von diesem “State” des deklarativen Wissens abhängigen Prozess der Fehlererkennung und anschließender Behebung. Somit ist in CBM nicht von Interesse, welche Interaktion, Handlungen oder Problemlösungen vom Schüler angewandt wurden, sondern in welchem “State” des deklarativen Wissens sich dieser befindet. Der Schüler kann somit alle Aktionen ausführen, außer jene, die in dem jeweiligen Problembereich als falsch gelten [Ant03].

Constraints repräsentieren die Applikation eines Teils deklarativen Wissens in einer expliziten Situation [Bev09]. Mit anderen Worten werden äquivalente Klassen an “Problem-States” gebildet, die miteinander in Form einer Bedingung verbunden werden. Die generelle Auffassung eines Contraints wird folgendermaßen beschrieben:

> *IF <relevance condition> is true*
> *THEN <statisfaction condition> will also be true*

Sofern eine studentische Lösung auf die Relevanzbedingung passt, muss diese alle Begleichungsbedingungen der selben Problemklasse erfüllen. Für alle Constraints einer Problemklasse wird demnach das gleiche Feedback generiert. Folgendes Constraint ist ein Beispiel aus der Mathematik [Bev09]:

Falls folgende Addition dem Schüler als Aufgabe vorliegt

$$\frac{a}{b} + \frac{c}{d}$$

und seine Lösung folgende Form annimmt

$$\frac{(a+c)}{n}$$

kann die studentische Lösung nur richtig sein, falls $b = d = n$ gilt. Sofern dies nicht der Fall ist, ist die Bedingung verletzt. Die zu der Problemklasse passende Hilfestellung zeigt dem Schüler das Missverständnis auf. Auf Basis dieser Regeln können Domänen somit effizient modelliert werden.

Diese Konzepte der Modellierung von Constraints wurden von Chriss Mills 2007 in der Entwicklung eines ITS umgesetzt, das auf einem 3D Serious-Game aufbaut [Mil07]. Dieses nennt sich StuntRobot und bringt Lernenden Newtonsche Physik bei. Das System nutzt zuzüglich Multi-Agent und maschinelle Lerntechniken, um den Spieler adaptiv zu unterstützen. In Multi-Agent-Systemen werden einzelne Bereiche von einem explizit dafür entwickelten Agent übernommen, der die jeweilige Teilaufgabe des ITS übernimmt. So wird am Ende die bestmögliche Adaptivität nicht durch die direkte Implementierung eines einzigen Bausteins, sondern durch die Orchestrierung verschiedener Techniken erreicht. In Abbildung 2.4 ist das Grundgerüst der Architektur dargestellt.

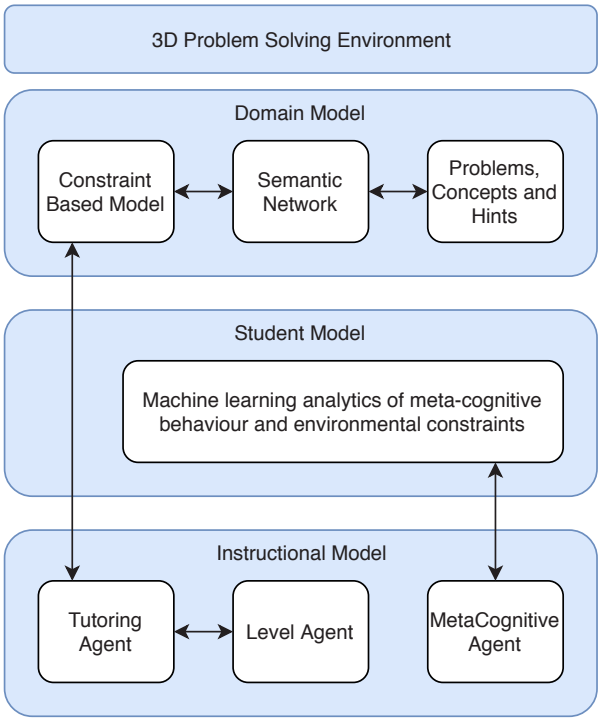

Abbildung 2.4: Systemarchitektur des Lernsystems StuntRobot [Mil07]

Im Kern des Systems steht die Constraint-basierte Modellierung des Domänenwissens. Zusätzlich werden Constraints durch ein “Semantic Network” organisiert, das Lerneinheiten und ihre

Abhängikeiten untereinander in einem Graph organisiert [Sow92]. Constraints und ihre jeweiligen Problemklassen bilden auf zugehörige Hilfestellungen und Konzepte zur adaptiven Unterstützung ab. Das Design der Constraints verfolgt hierbei den im Vorherigem beschriebenen Ansatz.

Hilfestellungen werden für die jeweiligen zu erlernenden Konzepte von Tutoring-Agents geleistet. Zuzüglich verwendet der Tutoring-Agent während des Systemtests maschinelle Lerntechniken, um Vorhersagen bezüglich der Antwortzeiten und Erfolgsraten von Schülern bei der Lösung von Teilproblemen zu generieren. So kann die durchschnittliche Zeit vor der nächsten erfolgreichen Schüleraktion kalkuliert werden. Wenn die tatsächliche Zeit die vorhergesagte Zeit überschreitet, kann der Tutoring-Agent eingreifen, indem er ein geeignetes Feedback oder einen relevanten Hinweis in Verbindung mit einem geeigneten Konzept für das aktuelle Lernziel bereitstellt.

Der Mata-Cognitive-Agent ermöglicht die Personalisierung auf den jeweiligen Student mithilfe maschineller Lerntechniken. Der Agent wertet Reaktionen auf Hilfestellungen und die strukturelle Spielweise aus, um die Performanz und den Wissensstand des Schülers zu verfolgen. Ergänzend dazu passt ein Level-Agent die Schwierigkeit des Levels an die Leistung des Schülers an. [Mil07]

3

Grundlagen

In diesem Kapitel werden die Grundlagen der Arbeit dieses Projekts thematisiert. Hierbei liegt der Fokus vor allem auf den Konzepten und Mechanismen kognitiver Architekturen. Zu Beginn wird die Strategie der in 4 angewandten Nutzwertanalyse vorgestellt. Außerdem wird auf die Themen Microservices und Experience-API eingegangen, da diese die praktische Grundlage der Implementierung des entwickelten Demonstrators bilden.

3.1 Nutzwertanalyse

In Kapitel 4 wurde auf Basis einer Nutzwertanalyse eine geeignete kognitive Architektur ermittelt. Diese wurde aus einer Vorauswahl nach subjektiver Kriterienbewertung selektiert. Das Verfahren orientiert sich an der von Szentes, Bargel und Streicher empfohlenen Vorgehensweise zur Auswahl einer Game-Engine [Sze12]. Der Ansatz resultiert in einer individuellen auf diese Arbeit zugeschnittenen Lösung. Im Kontrast dazu steht die allgemein bekannte ökonomische Kosten-Nutzen-Analyse (engl. Cost Benefit Analysis), welche den Nutzen zugehörigen Kosten gegenüberstellt. Andererseits wird mit der Nutzwertanalyse eine emotionale subjektive Entscheidung getroffen. Als Ergebnis entsteht eine Rangliste, die sich nicht auf andere Projekte transferieren lässt [Sze12].

Die Vorgehensweise dieser Methodik lässt sich in folgende Schritte unterteilen:

1. Alternativenselektion
2. Kriterienselektion
3. Gewichtung der Kriterien
4. Evaluation jedes Kandidaten
5. Kalkulation des finalen Ergebnisses

Zunächst wird eine Vorauswahl getroffen, sodass die Kandidaten auf eine für das Projekt realistische Anzahl eingegrenzt werden. Hierbei spielen die in der Nutzwertanalyse verwendeten Kriterien noch keine Rolle.

Im nächsten Schritt wird ein Kriterienbaum aufgestellt. In dem Entwurf werden alle Anforderungen an die zu verwendende kognitive Architektur abgedeckt. Auf der ersten Ebene des Baumes werden

zunächst die Hauptkriterien bestimmt, anschließend zugehörige Unterkriterien auf der zweiten Ebene.

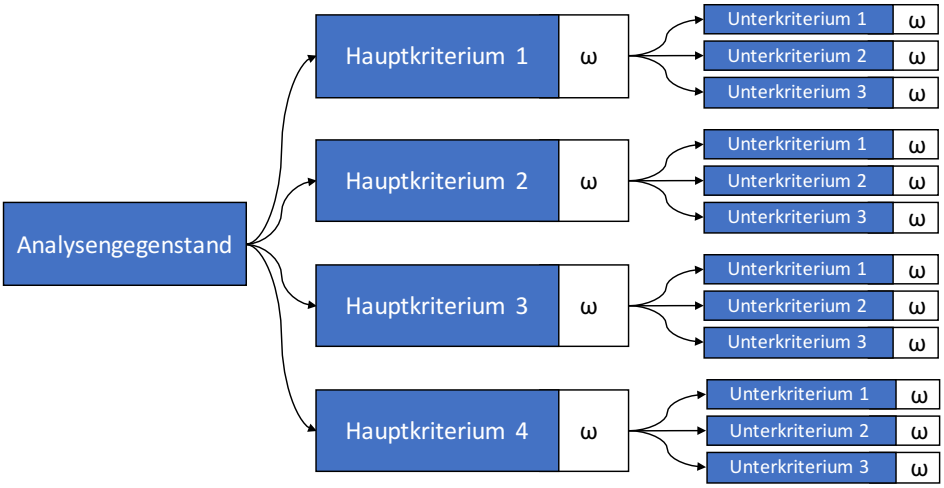

Abbildung 3.1: Kriterienbaum mit zugehöriger Gewichtung ω

Im weiteren Verlauf werden alle Knotenpunkte des entstandenen Kriterienbaums, wie in Abbildung 3.1 dargestellt, mit ω gewichtet. Hierfur bietet sich beispielsweise eine Skala von 0 bis 10 an und ist davon abhängig, wie genau die Benotung sein soll.

Sobald die Gewichtung der Kriterien erfolgt ist, werden alle Kandidaten evaluiert. Eine Bestimmung von Zielwerten, die die Benotung im Groben charakterisieren, kann dabei helfen, dass der Leser die Analyse aufgrund transparenterer Benotung besser nachvollziehen kann. Wie bei Szentes et al. vorgeschlagen eignet sich hierfür eine tabellarische Auflistung aller Kriterienbeschreibungen der einzelnen Zielwerte [Sze12].

Zur Auswertung der Analyse wird zunächst jeder einzelne Kandidat evaluiert. Die Benotung der Hauptkriterien ergibt sich aus dem normalisierten Durchschnitt der zugehörigen Unterkriterien. Der normalisierte Durchschnitt dieser Ergebnisse bildet wiederum das finale Gesamtergebnis.

3.2 Experience API und Lernanalyse

Die xAPI, auch Tin-Can API oder Experience API genannt, ermöglicht, Daten über die vielfältigen Erfahrungen eines Benutzers in Lernsystemen zu sammeln. Dabei erfasst die API Daten über die Aktivitäten der Benutzer aus verschiedenen Anwendungsbereichen in einem konsistenten Format [Rus18]. Das Vokabular der xAPI unterteilt sich in Substantive, Verben und Objekte. Die daraus resultierende Struktur ermöglicht es, die verschiedensten Aktivitäten und Interaktionen zu beschreiben. Die einzelnen xAPI Statements, welche jeweils eine Interaktion aufzeichnen, werden im JSON-Format erfasst. Ein Beispiel für ein einfaches Statement könnte folgendermaßen die Aktivität “John Doe experienced example activity” beschreiben:

```
{
  "actor": {
    "name": "John Doe",
    "mbox": "mailto:john.doe@example.com"
  },
  "verb": {
    "id": "http://adlnet.gov/expapi/verbs/experienced",
    "display": { "en-US": "experienced" }
  },
  "object": {
    "id": "http://example.com/activities/example-activity",
    "definition": {
      "name": { "en-US": "example activity" }
    }
  }
}
```

Source Code 3.1: xAPI Beispiel-Statement

Diese Statements werden, sobald die Interaktion des Benutzers erfolgt ist, mit weiteren Informationen wie einem Zeitstempel oder mit der Aktivität einhergehenden Werten versehen [Vid15]. In den meisten Systemen wird hierfür ein Adapter eingesetzt, der auch die Aufgabe der Weitergabe der Statements an einen sogenannten Learning Record Store (LRS) übernimmt, wie in Abbildung 3.2 veranschaulicht.

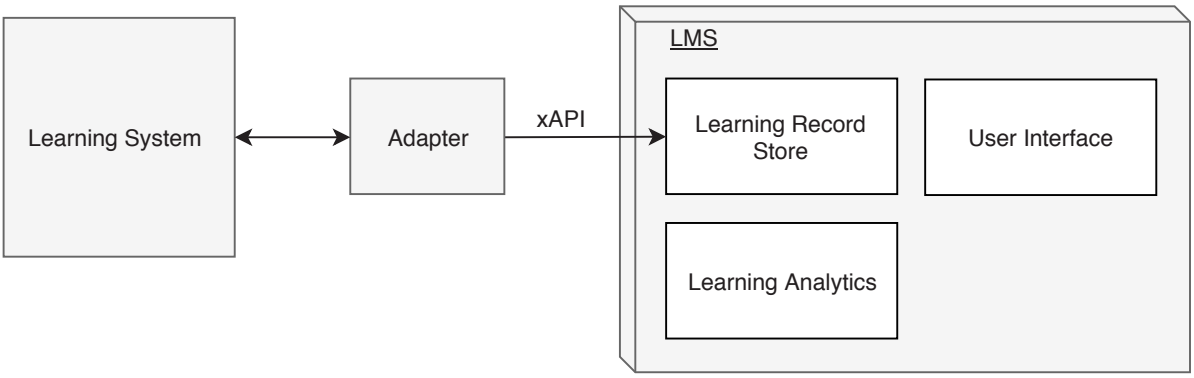

Abbildung 3.2: Kommunikation zwischen Lernsystem und LRS

Ein LRS kann zusammen mit weiteren Systemen zu einem Learning Management System (LMS) erweitert werden. Es wäre beispielsweise möglich auf Grundlage der Daten des LRS ein LMS bestehend aus Dashboards, Lernanalysen und weiteren Applikationen umzusetzen. In dieser Arbeit wurde das System von Learning Locker[1] verwendet und auf einem Server als LMS installiert.

1 https://learninglocker.net/

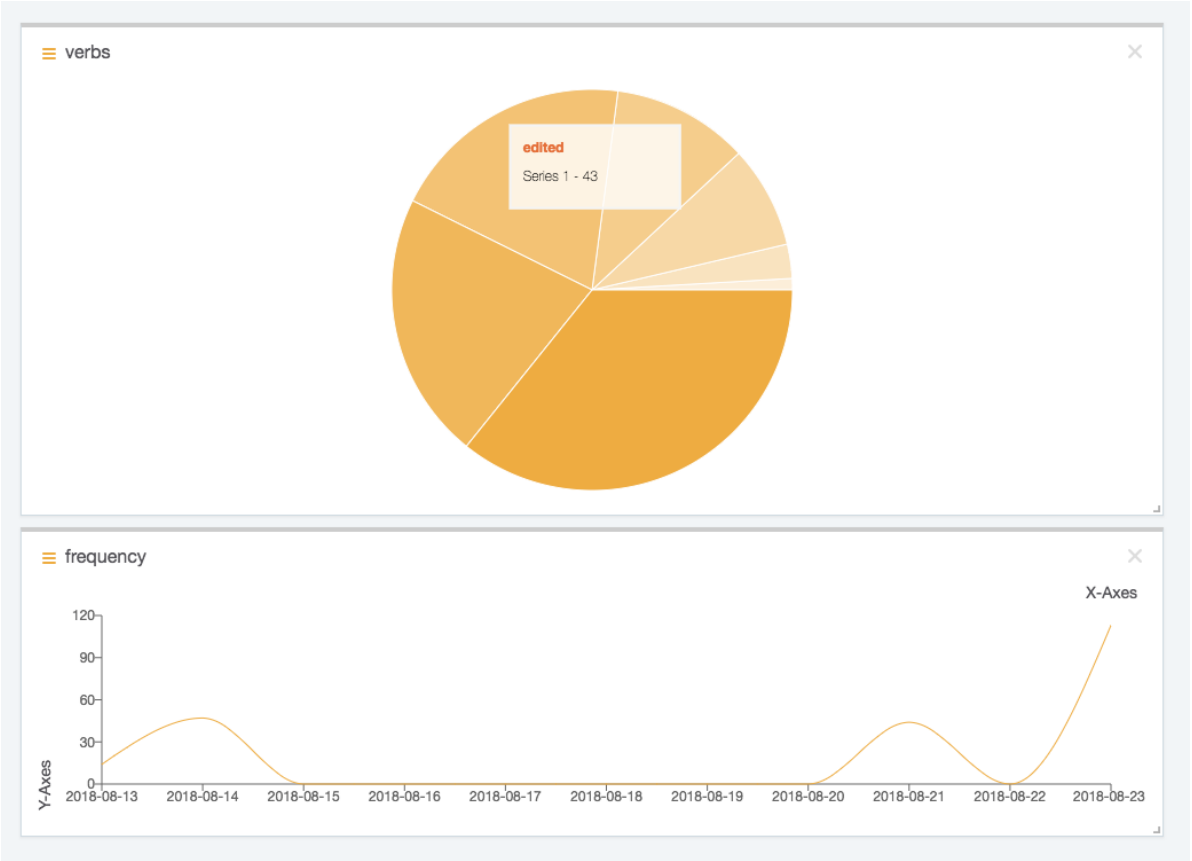

Abbildung 3.3: Learning Locker Dashboard

Dem Benutzer des LRS ist dabei ermöglicht mit einem Editor Queries auf der xAPI-Datenbank auszuführen. Beispielsweise könnten die xAPI-Statements so gefiltert werden, dass nur jene eines bestimmten Benutzer angezeigt sind und so explizit analysiert werden können. Des Weiteren werden alle xAPI-Statements aufgelistet und können in JSON-Form detailliert angezeigt werden.

Zuzüglich zum LRS wird von Learning Locker Funktionalität zur Visualisierung der xAPI-Statements bereitgestellt. In Abbildung 3.3 sind zwei Beispiele einer möglichen statistischen Darstellung aufgezeigt. Die Grafiken visualisieren die Statements, die im Zuge der Entwicklung des Demonstrators produziert wurden. In einem Pie-Chart werden die Statements nach dem verwendeten Verb gruppiert. In dem darunterliegenden Line-Chart wird die Häufigkeit der Statements in Abhängigkeit der Zeit angezeigt. Learning Locker bietet außerdem eine REST-Schnittstelle[1], welche es ermöglicht die xAPI-Statements von einem externen Service anzufordern. In Abschnitt 6.3 ist beschrieben, wie die REST-Schnittstelle in der Implementierung benutzt wurde.

1 `http://docs.learninglocker.net/http-rest/`

3.3 Kognitive Architekturen

Eine kognitive Architektur kann als eine umfassende, mechanistisch detaillierte Theorie definiert werden, wie Kognition in einem weiten Bereich von Domänen und Aufgaben operiert [O'R12]. Somit bietet eine kognitive Architektur eine Basis zur Simulation menschlichen Denkens. Hierbei werden psychologische kognitionswissenschaftliche Theorien aufgestellt, die in den Architekturen mathematisch formalisiert werden. Diese Theorien sind erst anerkannt, sobald sich die zugrundeliegende Architektur empirisch erwiesen hat. So ist das primäre Ziel der Forschung an kognitiven Architekturen die Entwicklung einer künstlich generellen Intelligenz (AGI) [Kot18]. In der Praxis werden diese Theorien auf eine bestimmte Domäne hin angewendet. Hierbei werden in der jeweiligen kognitiven Architektur Agenten modelliert, die auf Basis von Wahrnehmung und anschließender Informationsverarbeitung intelligent handeln. Eine zentrale Herausforderung der Konzeption einer kognitiven Architektur ist die Gestaltung der Repräsentation von Wissensquellen [Lan08]. Außerdem müssen Mechanismen entworfen werden, die die Auswahl einer Reaktion auf die Umgebung aus einer Menge an möglichen Alternativen bewerkstelligen. Dieser Prozess ist die zentrale Abstraktion menschlicher Kognition, wie in Abbildung 3.4 dargestellt.

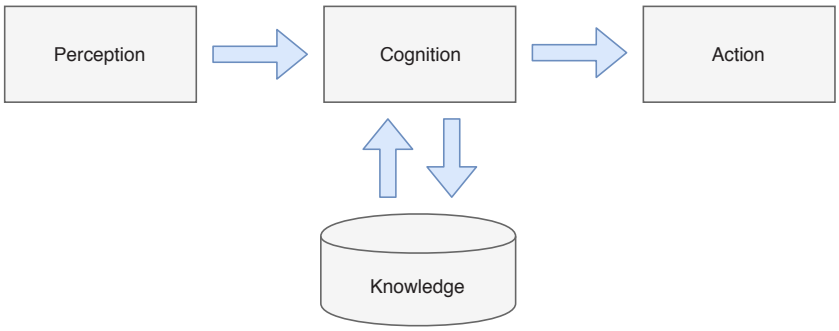

Abbildung 3.4: Informationsverarbeitung eines Agenten

Menschliche Intelligenz besteht jedoch nicht nur aus Wahrnehmung, Informationsverarbeitung und daraus resultierenden Reaktionen. Vor allem die Fähigkeit des Lernens und der Generalisierung des dabei erworbenen Wissens auf verschiedenste Anwendungsbereiche ist ein bedeutender Bestandteil dieser Intelligenz.

Kognitive Architekturen können nach verschieden Taxonomien der Simulation von Kognition kategorisiert werden [Kot18]. Diese sind im Folgenden vorgestellt.

Symbolische Systeme In der physischen Symbolhypothese wurde von Newell und Simon 1976 eine klare Auffassung symbolischer künstlicher Intelligenz beschrieben, nach der Symbole Teile einer Struktur an weiteren Symbolen sind und untereinander nach physischen Gesetzen in Beziehung stehen [New76] [Sun00]. Symbolische Strukturen werden wiederum als Ausdrücke aufgefasst, auf welchen Prozesse operieren, um weitere derartige Ausdrücke zu produzieren oder zu verändern. Dies bietet die notwendigen und ausreichenden Bedingungen, ein physisches intelligentes System zu entwickeln [Lew99]. In kognitiven Architekturen sind diese symbolischen Strukturen repräsentativ für das Wissen, auf dem mentale kognitive Prozesse operieren [Ass15].

Emergente Systeme Emergente Systeme, in der Fachliteratur auch “Connectionist” genannt, sind Netzwerke, die aus sehr vielen einfachen und gleichzeitig stark miteinander verbundenen Einheiten bestehen [Fod88]. Diese Netzwerke sind Modelle, analog zu neuronalen Netzen, bei denen der Informationsfluss durch eine Signalausbreitung von den Eingangsknoten repräsentiert wird. Die Signaleingabe wird durch sensorische Abtastung der Umgebung ermöglicht [Kot18]. Die durch die Verteilung der Prozesse auf viele Einheiten entstehende Parallelität ähnelt der neuronalen Gehirnaktivität. Jedoch werden die resultierenden Modelle als intransparent und komplex im Vergleich zum symbolischen Ansatz angesehen [Kot18].

Hybride Systeme Die zuvor vorgestellten Ansätze sind von Domänengebieten abhängig [Sun00]. So werden emergente Systeme beispielsweise zur Bildauswertung eingesetzt, jedoch seltener in psychologischen Experimenten [Kot18]. Dies wurde auch am Beispiel der kognitiven Architektur ART in der Nutzwertanalyse in Abschnitt 4.1 ersichtlich. Somit unterstützt jeder Ansatz bestimmte Bereiche von Kognition, deckt jedoch nicht alle grundlegenden Aspekte ab [Kot18]. In Hybriden Systemen werden beide Ansätze miteinander kombiniert, um dieses Problem zu umgehen und das gesamte Spektrum der menschlichen Kognition abzudecken.

Grundlegend für die Theorie hinter kognitiven Architekturen ist die bereits erwähnte Wissensrepräsentation. Hierbei wird in der Kognitionswissenschaft zwischen zwei wesentlichen Wissensarten unterschieden. *Deklaratives* Wissen basiert auf atomaren Fakten, während *prozedurales* Wissen zielorientiert Problemlösungsschritte darstellt [Cor95]. So ist folgender Satz ein Beispiel für deklaratives Wissen:

> Die Python Funktion `reverse()` nimmt eine Liste im Argument entgegen und gibt die reversierte Liste zurück.
>
> (Beispielsweise würde `numbers.reverse()` die Liste `numbers` reversieren.)

Prozedurales Wissen beinhaltet die Ausführungsschritte der Anwendung dieses Wissens und tritt in Form von mehreren miteinander verbundenen Regeln auf. Diese werden auch Produktionen genannt und nehmen meistens eine “IF-THEN” Form an. Bezogen auf das deklarative Beispiel könnte folgende Produktion formuliert werden:

> Falls das Ziel ist, einen Ausdruck zu schreiben, der die Liste `numbers` reversiert,
>
> dann schreibe zunächst `numbers` und setze ein neues Ziel, um darauf die `reverse`-Funktion mit dem Punktoperator anzuwenden.

In Abschnitt 5.1 wird eine umfangreichere Produktion auf Basis der hybriden kognitiven Architektur ACT-R aufgezeigt.

3.4 Intelligente Tutoring Systeme

Das Ziel intelligenter Tutoringsysteme (ITS) ist es, die Schüler in eine anhaltende Denkaktivität zu bringen und, basierend auf einem tiefen Verständnis des Verhaltens der Schüler, mit diesen zu interagieren [Cor97]. Hierbei vereint ein ITS mehrere verschiedene Techniken und Mechanismen, um dieses Ziel zu realisieren. Im Verlauf der Recherche dieser Arbeit wurde ersichtlich, dass in der Literatur ein einheitlicher Konsens über Struktur, Funktionsweise und Techniken in diesen

Systemen besteht. In folgender Abbildung 3.5 sind die Schichten eines ITS und ihre Abhängigkeiten untereinander dargestellt.

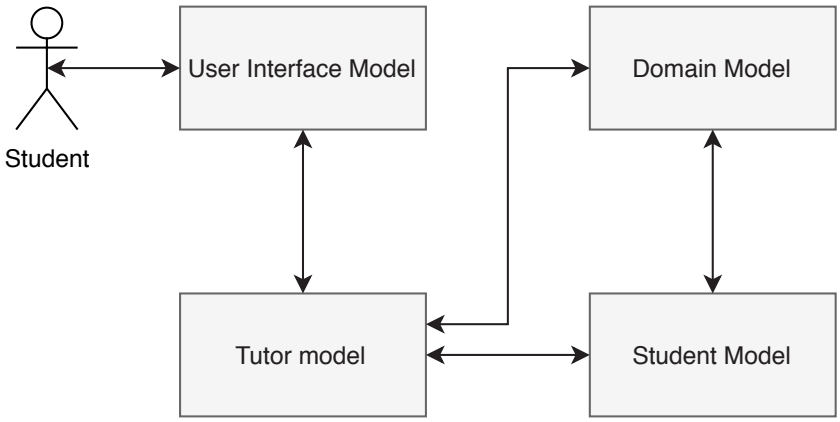

Abbildung 3.5: Struktur eines intelligenten Tutoring Systems

Domain Model Das Domänenmodell, auch Expertenmodell genannt, erfüllt in einem ITS zwei Aufgaben. Erstens dient es als Quelle für das Wissen, das dem Schüler präsentiert wird, was das Generieren von Fragen, Erklärungen und Antworten einschließt. Zweitens bietet es einen Standard für die Bewertung der Leistung des Schülers [Nwa90]. Somit empfindet das Domänenmodell Problemelösungen der behandelten Domäne dem menschlichen Denkvermögen nach. In der Literatur wird zwischen verschiedenen Ausprägungengsgraden des Domain-Models unterschieden: Ein **Black-Box-Model** beschreibt und löst Probleme der Domäne auf grundsätzlich andere Art und Weise, als ein menschlicher Student [Cor97]. Ein **Glass-Box-Model** ist ein fortgeschritteneres Modell, das auf die gleichen Domänenkonstrukte wie ein menschlicher Experte zurückgreift [Cor97]. Für die Entscheidungsfindung wird jedoch eine andere Kontrollstruktur verwendet. Ein Beispiel hierfür ist das MYCIN System, das aus aus einer großen Menge an "IF-THEN" Regeln besteht. Diese bilden wahrscheinlichkeitstheoretisch Krankheitszustände zu Diagnosen ab [Sho76]. In den Regeln wird die gleiche Denkweise menschlicher Diagnosenfindung widergespiegelt, jedoch werden die Diagnosen in MYCIN mit Hilfe eines "depth first backward-chaining control-scheme" bestimmt [Bev09][Sho76]. Das fortgeschrittenste Model wird **Ideal-Student-Model** genannt und darf nicht mit der im Folgenden thematisierten Student-Model-Schicht verwechselt werden. Hierbei wird auch das Wort "Student" verwendet, um zu signalisieren, dass der Experte auf der gleichen kognitiven Ebene, wie ein menschlicher Student agiert. Das Modell beinhaltet das ideale Wissen, welches vom Studenten akquiriert wird [Cor97]. Dies beinhaltet alle Lösungswege der Domäne, was auch die Falschen mit einschließt. In Abschnitt 2.1 wird ein kognitiver digitaler Tutor der linearen Algebra vorgestellt, der ein Ideal-Student-Model verwendet.

Student Model Das studentische Modell hat vor allem die Aufgabe das Wissen und die Performanz eines Schülers zu jedem Zeitpunkt aufzuzeichnen und zu bewerten. Es ist dem Entwickler frei gestellt, wie dies realisiert wird. Das Modul dient in der Umsetzung kognitiver ITS als sogenanntes "Overlay" zum Domänenmodell, da es eine Kopie des Domänenmodells beinhaltet. In dieser wird jeder Wissenseinheit, die zu einem bestimmten Zeitpunkt eingeschätzte studentischen Performanz

hinzugefügt. Es soll also zuzüglich in das Domänenmodell gespeichert werden, wie gut der Student jede explizite Wissenseinheit erlernt hat. Zuzüglich kann ein Fehlerkatalog eingesetzt werden, welcher eine Reihe von Missverständnissen oder falschen Regeln beinhaltet, die jeweils einen Hinweis darauf enthalten, ob der Schüler ein Missverständnis erworben hat [Cor97]. Systeme verwenden im Student-Model auch maschinelle Lerntechniken und statistische Verfahren, um den Wissenstand des Schülers zu bewerten [Mil07].

Tutor Model Das Tutor-Model kombiniert die Erkenntnisse aus dem Domain-Model und Student-Model, um auf den Schüler einzugehen. Die Art und Weise, wie dieses Modul gestaltet ist, hat direkten Einfluss auf die Qualität des ITS. Das Model generiert Feedback, das den positiven Einfluss auf den studentischen Lernprozess bestimmt. Zu Beginn der Entwicklung eines ITS sollte der Fokus auf dieses Modul gerichtet sein und überlegt werden, welche Art von Domain-Model und zugehörigem Student-Model für die benötigte Unterstützung des Schülers notwendig ist. Hierbei bieten sich dem Entwickler zwei verschiedene Möglichkeiten an, welche in den Abschnitten 2.1 und 2.1 dokumentiert werden.

User Interface Model Das User Interface Model beinhaltet das Front-End des jeweiligen Lernsystems und dient als Schnittstelle zum Schüler. In dieser Arbeit wurde hierfür das bestehende Serious Game Extra verwendet, das in Abschnitt 1.3 näher beschrieben wurde.

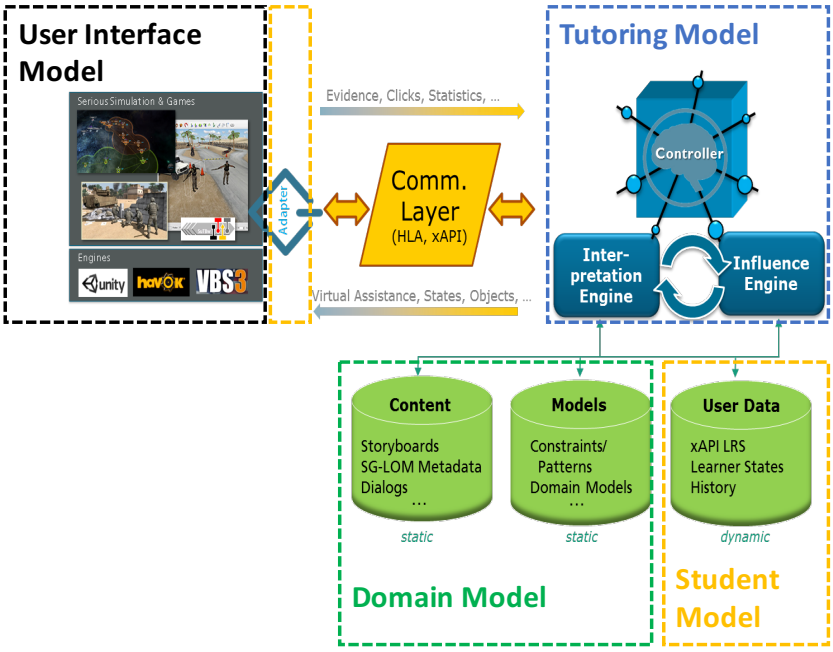

Abbildung 3.6: ELAI Architektur ITS

In Abbildung 3.6 sind die in Abschnitt 1.3 thematisierten Komponenten der ELAI-Architektur auf die bereits vorgestellten Schichten des ITS bezogen. Hierbei ist der in dieser Arbeit für EXTRA implementierte Adapter Teil des Student-Models. Des Weiteren wird kein Domain-Model implementiert, da dies, wie im Kapitel über den Stand der Forschung und Technik ersichtlich wurde, für diese Arbeit mit zu großem Aufwand verbunden wäre. Der Demonstrator dieser Arbeit und das installierte LRS

sind auch in das Student-Model einzuordnen. Um ein vollständiges ITS zu entwickeln, wäre es folglich von Nöten die Domäne von EXTRA mittels einer der zwei Ansätze zu modellieren, die in Kapitel 2 vorgestellt wurden. Hierbei wäre die erwünschte Funktionalität des Tutor-Models ausschlaggebend für die Art und Weise der Umsetzung.

4

Systematische Auswahl kognitiver Architekturen

Dieses Kapitel dokumentiert die Ergebnisse der Nutzwertanalyse kognitiver Architekturen im Hinblick auf die Implementierung des Demonstrators. Die Auswahl der Architektur teilt sich in zwei Abschnitte auf. Im ersten Teil wurde die generelle Architektur ausgewählt, im zweiten die Implementierung dieser bestimmt.

4.1 Vorauswahl kognitiver Architekturen

Als Ausgangspunkt für die in diesem Kapitel thematisierte Nutzwertanalyse war es notwendig, zunächst eine Basisauswahl festzulegen. Hierbei wurde nach Bekanntheitsgrad und Wichtigkeit der Architekturen entschieden. Bezogen auf die Anzahl an Erwähnungen in Veröffentlichungen ist ACT-R die bekannteste Architektur [Kot18]. Auch SOAR, ART, Leabra, Clarion und LIDA sind noch so populär, dass sie über einen Wikipediaeintrag verfügen.

Im Folgenden werden diese Architekturen zunächst grob beschrieben.

ACT-R Die Hauptmotivation von ACT-R besteht darin, ein Modell der menschlichen Kognition zu entwickeln, wobei empirische Daten verwendet werden, die aus Experimenten in der kognitiven Psychologie stammen [Cho07]. Auf Basis deklarativer und prozeduraler Kognitionsmodellierungen wird eine schrittweise Simulation des menschlichen Verhaltens für ein detailliertes Verständnis der menschlichen Wahrnehmung entwickelt. In Abschnitt 5.1 wird diese Architektur detailliert vorgestellt.

SOAR Soar steht für **S**tate, **O**perator, **A**pply, **R**esult. In der Architektur kommen prozedurale Regeln in Kombination mit relationalen Graphenstrukturen zum Einsatz [Lai87]. Hierbei ist die Unterteilung in ein Langzeitgedächtnis und ein Arbeits- oder auch Kurzeitgedächtnis zentral [Cho07].

CLARION CLARION speichert sowohl aktionsorientiertes als auch nicht aktionsbezogenes Wissen in impliziter Form unter Verwendung mehrschichtiger neuronaler Netze und in expliziter Form unter Verwendung symbolischer Produktionsregeln [Sun01]. Kurzzeitspeicher enthalten Aktivierungen für symbolische Knoten in neuronalen Netzen, die die Architektur an Regelstrukturen anpasst [Cho07].

Zusammenfassend werden in dieser Architektur neuronale Netze mit “Chunk” Speichern und Regeln kombiniert.

LEABRA Leabra steht für **L**ocal, **E**rror-driven and **A**ssociative, **B**iologically, **R**ealistic, **A**lgorithm [O’R00]. Die Architektur basiert direkt auf der zugrunde liegenden Biologie des Gehirns. Im Kern werden eine Reihe von biologisch inspirierten realistischen neuronalen Verarbeitungsmenchanismen verwendet [O’R12].

LIDA LIDA ist ein umfassendes, konzeptionelles und computergestütztes Modell, das einen großen Teil der menschlichen Wahrnehmung abdeckt [Fra11]. Die kognitiven Module der LIDA-Architektur umfassen ein perzeptiv-assoziatives Gedächtnis, episodisches Gedächtnis, funktionelles Bewusstsein, prozedurales Gedächtnis und Handlungsselektion [Fra07]. Dabei implementiert LIDA die “Global Workspace Theory of Consciousness” [Rob09].

ART ART steht für **A**daptive **R**esonance **T**heory und basiert auf einer Reihe an Echtzeit-Neuronalen Netzen, welche sowohl überwachtes, als auch unüberwachtes Lernen nutzen [Car].

ICARUS In Icarus wird eine Abstraktion eines Langzeitgedächtnisses eingesetzt, welche probabilistische hierarchische Konzepte verwendet, um alle Wissensarten zu speichern [Lan91]. Hierbei werden zwei Arten von Wissen spezifiziert. **Konzepte** beschreiben Klassen von Umweltsituationen in Bezug auf andere Konzepte und Wahrnehmungen. **Fertigkeiten** geben an, wie Ziele erreicht werden, indem sie in geordnete Teilziele zerlegt werden [Lan08].

4.2 Kriterien

Wie im Grundlagenkapitel unter Abschnitt 3.1 beschrieben, werden in einer Nutzwertanalyse genaue Kriterien und zugehörige Zielwerte als Maßstab festgelegt. Die Analyse wurde hierbei in zwei Phasen unterteilt. Zunächst wurde im ersten Schritt die allgemeine Architektur ausgewählt. Da sich die verschiedenen Implementierungen der Architekturen wiederum vor allem in Dokumentation und Lizenzhandhabung unterscheiden, wurde im zweiten Teil die konkrete Implementierung der festgelegten kognitiven Architektur ermittelt.

4.2.1 Festlegung auf die kognitive Architektur

Wie in Abbildung 4.1 veranschaulicht unterteilen sich die Kriterien der Architekturauswahl zunächst in die beiden Hauptkriterien "Architekturmerkmale" und "Implementierungseignung". Hierbei wurde die Implementierungseignung priorisiert, um das Ergebnis des Projektes nicht zu gefährden.

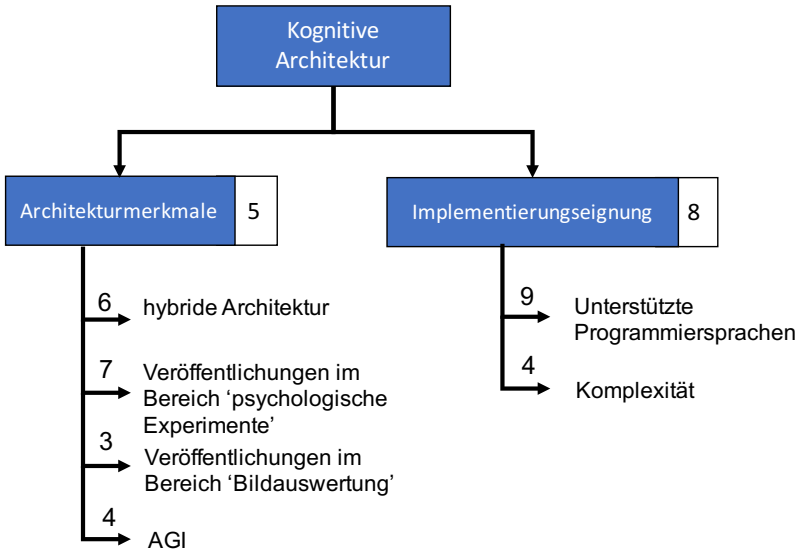

Abbildung 4.1: Kriterienbaum zur Auswahl der kognitiven Architektur und zugehörige Gewichtung

Für die Auswahl der Architektur wurden folgende Architekturmerkmale festgelegt:

Hybride Architektur Das Projekt hat den Anspruch, dass eine hybride Architektur verwendet wird. Hybride Architekturen versuchen, Elemente sowohl symbolischer, als auch emergenter Ansätze zu kombinieren [Kot18]. Dies wurde in Abschnitt 3.3 bereits thematisiert.

Veröffentlichungen im Bereich psychologischer Experimente Lernsysteme, die adaptiv den individuellen Lernprozess eines Benutzers unterstützen, sind darauf angewiesen auf die Kognition dessen einzugehen. Für die Modellierung der Kognition in Lernprozessen sind die Erkenntnisse aus der Psychologie das bedeutsamste Mittel. Aus diesen Gründen ist dieses Merkmal am schwersten gewichtet.

Veröffentlichungen im Bereich Bildauswertung Das in in dieser Arbeit zu entwickelnde System reiht sich inhaltlich auch im Bereich der Bildauswertung ein. Das Lernspiel Extra behandelt die verschiedenste Prozesse der Bildverarbeitung. Vor dem Hintergrund der Weiterentwicklung des Demonstrators und dem möglichen Einsatz der erreichten Erkenntnisse im näheren Projektumfeld wurde dieses Merkmal einbezogen.

AGI Beim Entwurf kognitiver Architekturen wird versucht künstlich generelle Intelligenz (AGI) zu erreichen. AGI wird in dem Forschungsfeld als Bezeichner für künstliche Intelligenz verwendet, die von der menschlichen Intelligenz nicht zu unterscheiden ist. Bestimmte Architekturen spezialisieren sich jedoch des Öfteren auf eine Domäne, während andere Architekturen generelleren Anspruch verfolgen. In dieser Arbeit werden Architekturen präferiert, die in den meisten Domänen einsetzbar sind und somit AGI besser umsetzen.

Unterstützte Programmiersprachen Im Verlauf der Recherche wurde erkenntlich, dass bestimmte Architekturen in Programmiersprachen entwickelt sind, die für dieses Projekt nicht verwendet werden sollten, da sie in der Praxis rar sind. Dies kann zu Problemen in der Weiterführung der Projektarbeit führen.

Komplexität Die Modellierung in kognitiven Architekturen ist ein komplexes Unterfangen. In der Analyse wird mit diesem Kriterium darauf geachtet, dass der Rahmen des Projektes eingehalten wird.

4.2.2 Festlegung auf die Implementierung der kognitiven Architektur

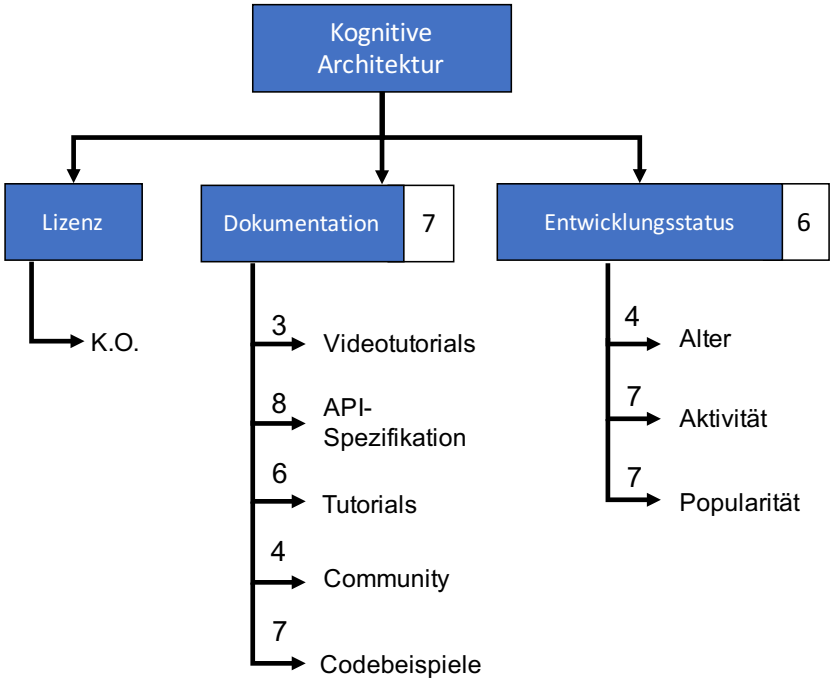

Abbildung 4.2: Kriterienbaum zur Auswahl der Implementierung der kognitiven Architektur und zugehörige Gewichtung

Wie einleitend in diesem Kapitel bereits erwähnt, wurde im zweiten Teil der Analyse die Implementierung der im ersten Teil ausgewählten Architektur bestimmt. Hierbei ist die Lizenz zunächst ein k.o. Kriterium. Das heißt, Architekturimplementierungen können a priori auf Grund urheberrechtlichen Gründen oder proprietärer Lizenzhandhabung von der Auswertung ausgeschlossen werden. Des Weiteren decken die Merkmale 'Dokumentation' und Entwicklungsstatus alle relevanten Kriterien an die Implemenitierungseignung ab. Im Folgenden sind diese genauer aufgeführt:

Dokumentation Die Dokumentation einer Software erklärt die Funktionsweise detailliert, sodass diese für Entwickler nutzbar wird. Eine Dokumentation kann auf verschiedene Art und Weise umgesetzt werden.

Videotutorials Videotutorials können den Entwicklungsprozess beschleunigen. Mithilfe einer schnelleren Wissensbeschaffung kann das Projekt einen größeren Mehrwert bieten.

API-Spezifikation Das wichtigste Kriterium der Dokumentation ist die API-Spezifikation. Sofern diese Lücken aufzeigt, kann es zu erheblichen Verzögerungen und Entwicklungsproblemen kommen. In einer API-Spezifikation sind alle Erklärungen und für Entwickler benötigte Hinweise zur Nutzung einer API aufgezeigt.

Tutorials Ein Tutorial eignet sich als interaktiver Wissenstransfer, um Inhalte schneller und besser zu verstehen. Tutorials verwenden hierbei beispielorientierte Methoden, um die zu erlernende Domäne oder das Wissen, wie eine bestimmte Problemstellung zu lösen ist, klarer darzustellen.

Community Gruppen (engl. Community) sind in der Software-Entwicklung von Bedeutung, da es dort zum regen Austausch bei Problemlösungsengpässen kommt. Sofern keine größere Community für ein bestimmtes Themengebiet existiert, ist es für Entwickler schwieriger Hilfestellungen zu finden. Des Weiteren ist eine kleine Community ein Indiz für mangelndes Interesse an an der in dieser Gruppierung thematisierten Thematik. Eine Nebeneffekt einer großen Community ist eine rege Beteiligung in Internetforen wie beispielsweise Stackoverflow.

Codebeispiele Die Veröffentlichung von Codebeispielen ist für Entwickler wichtig, um die Konzepte und Techniken eines Gebietes näher nachvollziehen zu können. Der Begriff Codebeispiele beschriebt hierbei kurze Codefragmente, die alle grundlegenden Inhalte der zu zeigenden Technologie prägnant und verständlich vereinen.

Entwicklungsstatus Der Entwicklungsstatus einer Software informiert über die Lauffähigkeit des Systems. Des öfteren kennzeichnet der Entwicklungsstatus, dass die Software noch nicht fehlerfrei läuft.

Alter Das Alter einer Software ist dahingehend interessant, als dass ältere Systeme viele Entwicklungszyklen durchlaufen haben und sich auf Grund der längeren Fortführung des Projektes in der jeweiligen Domäne somit bewert haben. Folglich sind diese Systeme in der Regel weniger fehleranfällig.

Aktivität Eine erhöhte Aktivität um eine bestimmte Software ist ein Merkmal erfolgreicher Projekte. Eine geringe Aktivität passiert meistens aufgrund von mangelnden Interesse an der Software und hat zur Folge, dass eine aktive Fortführung des Projekts nicht wirtschaftlich ist.

Popularität Die Popularität einer Software zielt auf den Bekanntheitsgrad ab. In dieser Analyse wurden populärere Architekturen präferiert, um das Interesse zu steigern. In den meisten Fällen erfüllt populäre Software ihren Zweck besser, als unbekannte Systeme. Hier ist auch zu beachten, dass sich neuere Systeme erst etablieren müssen und deshalb weniger populär sind.

4.3 Auswertung

Wie bereits in Abschnitt 3.1 erwähnt, basiert diese Nutzwertanalyse auf subjektiver Einschätzung. In den Tabellen 4.1 und 4.2 sind die zur Auswertung verwendeten Zielwerte dargestellt.

Hauptkriterium	Unterkriterium	Ziel		
		0 ≤ r ≤ 3 *ungenügend bis mangelhaft*	**4 ≤ r ≤ 7** *ausreichend bis befriedigend*	**8 ≤ r ≤ 10** *gut bis sehr gut*
Architektur-merkmale	Hybride Architektur	Die Architektur ist symbolisch einzuordnen	Subsymbolische Prozesse werden unterstützt	Die Architektur ist hybrid einzuordnen
	Veröffentlichungen im Bereich “psychologische Experimente”	Es existieren wenige bis keine Veröffentlichungen	Es existieren mehrere Veröffentlichungen	Es existieren viele Veröffentlichungen
	Veröffentlichungen im Bereich “Bildauswertung“	Es existieren wenige bis keine Veröffentlichungen	Es existieren mehrere Veröffentlichungen	Es existieren viele Veröffentlichungen
	AGI (Künstlich generelle Intelligenz)	Der Fokus der Architektur richtet sich auf eine bestimmte Domäne	Der Fokus der Architektur deckt mehrere Domänen ab	Der Anspruch der kognitiven Architektur ist generelle künstliche Intelligenz zu erreichen
Implemen-tierungseignung	Unterstützte Programmier-sprachen	Die unterstützten Programmiersprachen sind unbekannt und schlecht dokumentiert	Mindestens eine Implementierung in Java oder Python existiert	Mehrere bekannte Programmier-sprachen sind unterstützt
	Komplexität	Die Modellierung in der kognitiven Architektur ist mit zu großem Aufwand verbunden	Die Architektur ist komplex, jedoch lässt sich der Aufwand minimieren	Die Komplexität der Architektur ist gering

Tabelle 4.1: Kriterien der kognitiven Architektur und zugehörige Zielwerte

Hauptkriterium	Unterkriterium	Ziel		
		0 ≤ r ≤ 3 *ungenügend bis mangelhaft*	**4 ≤ r ≤ 7** *ausreichend bis befriedigend*	**8 ≤ r ≤ 10** *gut bis sehr gut*
Dokumentation	Videotutorials	Es sind keine Videotutorials zu finden	Einige wenige Videotutorials sind auffindbar, bieten jedoch geringen Mehrwert	Es sind mehrere gute Videotutorials auffindbar
	API – Spezifikation	Es existiert keine Spezifikation	Die Spezifikation ist teilweise lückenhaft	Die Spezifikation ist ausführlich und verständlich
	Tutorials	Es sind keine Tutorials zu finden oder diese sind unverständlich	Es existiert ein Tutorial	Es existieren mehrere gut verständliche Tutorials
	Community	Es existiert keine aktive Community	Es existiert eine kleine aktive Community	Es existiert eine große aktive Community
	Codebeispiele	Es sind keine Codebeispiele auffindbar	Gefundene Codebeispiele sind nicht aufschlussreich	Es existieren mehrere gute Codebeispiele
Entwicklungs-status	Alter	Die Software ist neu und noch nicht getestet	Die Software enthält mehrere Updates und ist ausreichend getestet	Die Software wird seit mehreren Jahren aktiv weiterentwickelt
	Aktivität	Der Entwicklungsstand der Implementierung liegt mehrere Jahre zurück	Der Entwicklungsstand liegt höchstens ein Jahr zurück	Die Software wird aktiv weiterentwickelt
	Popularität	Die Software ist über Suchmaschinen kaum auffindbar	Die Software ist über Suchmaschinen leicht auffindbar	Die Software wird in mehreren Veröffentlichung-en thematisiert und ist allgemein populär

Tabelle 4.2: Kriterien der Implementierung der kognitiven Architektur und zugehörige Zielwerte

4.4 Ergebnisse und Diskussion

In dem Ergebnis der Auswertung setzte sich ACT-R von den restlichen Architekturen beachtlich ab, wie in Tabelle 4.3 ersichtlich. Die Bewertung der Veröffentlichungen im Bereich “psychologische Experimente” und “Bildauswertung” stütze sich auf die von Kotseruba und Tsotsos 2018 in “A Review of 40 Years in Cognitive Architecture Research Core Cognitive Abilities and Practical Applications” veröffentlichen Werte [Kot18].

Hauptkriterien	Unterkriterien	Gewichtung	ACT-R	LEABRA	SOAR	CLARION	LIDA	ICARUS	ART
Architekturmerkmale	Hybride Architekturen	6	10	2	10	10	10	3	2
	Veröff. Im Bereich 'psych. Experimente'	7	10	7	3	7	6	0	3
	Veröffentlichungen im Bereich 'Bildauswertung'	3	0	4	0	0	0	0	10
	AGI	4	9	4	9	3	5	8	7
	Total	7	8,30	4,45	5,85	6,05	6,01	2,5	4,55
Implementier-ungseignung	Komplexität	4	2	4	3	3	1	3	1
	Unterstützte Programmiersprachen	9	7	5	5	5	7	4	5
	Total	4	5,46	4,69	4,38	4,38	5,15	3,69	3,77
Gesamtbewertung			**7,27**	**3,98**	**4,54**	**5,44**	**5,7**	**2,93**	**4,27**

Tabelle 4.3: Ergebnisse der Auswahl der kognitiven Architektur

In dem Kriterium Veröffentlichungen im Bereich “Bildauswertung” schnitten bis auf ART alle kognitiven Architekturen schlecht ab. Dies ist damit zu begründen, dass in diesem Bereich eher maschinelle Lerntechniken zum Einsatz kommen, wie sie beispielsweise in ART verwendet werden. ART ist vor allem auf diese Domäne spezialisiert und für diese Arbeit ungeeignet. Von Vorteil ist die große Bandbreite der Reimplementierungen der Architekturen in Programmiersprachen wie Java oder Python. Dies kann das Verständnis der grundlegenden Strukturen und Prozesse für Entwickler mit verschiedensten Programmierkenntnissen erleichtern.

In Tabelle 4.4 ist das Ergebnis der Auswahl der Implementierung von ACT-R dargestellt. Alle Implementierungen werden nicht mehr aktiv fortgeführt. Desweiteren gibt es keine größeren Communities oder Ansprechpartner, die den Einstieg in die APIs erleichtern könnten. Ausschlaggebend für das Ergebnis waren vor allem die guten Codebeispiele von Python ACT-R, welche die grundlegenden Mechanismen einprägsam darstellen. Diese sind auf einer Website[1] des Entwicklers mit weiteren Beschreibungen und Ausführungen veröffentlicht.

1 https://sites.google.com/site/pythonactr/

Hauptkriterien	Unterkriterien	Gewichtung	Java ACT-R	Python ACT-R	PyACT-R
Dokumentation	Videtutorials	3	0	0	0
	API-Spezifikation	8	4	5	3
	Tutorials	6	2	5	4
	Community	4	1	2	2
	Codebeispiele	7	3	8	5
	Total	7	2,46	4,79	3,25
Entwicklungsstatus	Alter	4	2	1	1
	Aktivität	7	1	3	2
	Popularität	7	1	3	2
	Total	4	1,23	2,56	1,76
Gesamtbewertung			**2,01**	**3,98**	**2,71**

Tabelle 4.4: Ergebnisse der Auswahl der Implementierung der kognitiven Architektur

5

Entwurf

Gegenstand der Arbeit war es, einen Microservice zu entwerfen, welcher den Activity-Stream von EXTRA ausnutzt, um Schüleraktionen auf Basis einer kognitiven Architektur zu analysieren. Dieser Microservice wurde als User Model Artificial Intelligence (UMAI) benannt. In diesem Kapitel werden die Entwurfsentscheidungen von UMAI dargelegt. Des Weiteren werden alle für die Implementierung verwendeten Technologien und die Interaktionen der einzelnen Komponenten des entstandenen Systems erklärt. Zu Beginn des Kapitels eird zumächst die kognitive Architektur ACT-R vorgestellt, welche in der Nutzwertanalyse in Abschnitt 4 ausgewählt wurde.

5.1 Vorstellung der kognitiven Architektur ACT-R

ACT-R versucht mit einem Modulkonzept die formale Abstraktion menschlichen Verhaltens umzusetzten. Dabei wird voraus gesetzt, dass Erkenntnis durch die Interaktion eines prozeduralen Gedächtnisses mit einem deklarativen Speicher und unabhängigen Modulen für externe Wahrnehmung und Handlungen entsteht [Taa04]. Der exakte Mechanismus dieser Module ist mathematisch festgelegt und wird fortlaufend erforscht. Jedes Modul ist mit einem Buffer ausgestattet, der die temporäre Speicherung und Informationsweitergabe der Module untereinander ermöglicht.

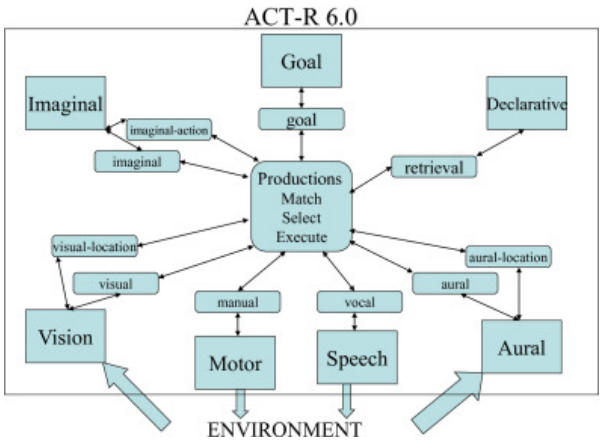

Abbildung 5.1: ACT-R Architektur [Tho15]

In Abbildung 5.1 sind die Kommunikationswege der aktuellen Version 6.0 dargestellt. Im Kern liegt das prozedurale Modul, in dem Produktionen gespeichert werden. Produktionsregeln abstrahieren Regeln, mit denen Teile einer Aufgabe abgehandelt werden. In dieses Modul fließt die in den umliegenden Modulen generierte Information ein und wird anschließend verarbeitet, sodass der nächste Aktionsschritt bestimmt werden kann. Produktionsregeln dienen in ACT-R somit als Schaltungsfunktionen, die bestimmte Informationsmuster in den Speichern der Module, genannt Buffern, auf Änderungen der Buffer-Inhalte abbilden [And06]. Mit anderen Worten erklärt, wird eine Produktion aus der Menge aller möglichen Produktionsregeln dann aufgerufen, wenn diese auf die Belegung der Buffern zu einem bestimmten Simulationschritt passt. Innerhalb der Produktionen kann die Information der Buffern überschrieben werden, was wiederum Prozesse innerhalb der jeweiligen Module auslöst. Die Simulation der einzelnen Module erfolgt parallel, jedoch kann jedes einzelne Modul nur seriell einzelne Aktionen ausführen. Beispielsweise ist es dem deklarativen Modul nur ermöglicht ein Element zu einem bestimmten Zeitpunkt abzurufen oder das visuelle Modul kann seine Aufmerksamkeit nur auf ein Element im visuellen Feld richten [And06]. Produktionen sind syntaktisch als "IF-THEN" Regeln realisiert. In folgendem Pseudocode ist die Funktionsweise verdeutlicht: .

```
IF      X is the current goal (goal buffer)
AND     Y is retrieved (declerative buffer)
AND     Z is seen (visual buffer)
THEN    A is executed (manual buffer)
AND     B is set as new goal (goal Buffer)
```

Wie in diesem Beispiel veranschaulicht dient der IF-Teil in einer Produktion der Spezifizierung einer Reihe von Bedingungen. Diese müssen in den jeweiligen Buffern der Module zu einem bestimmten Zeitpunkt gespeichert sein, sodass die Produktion ausgewählt werden kann. Eine Utility-Rate ermöglicht dem Modellierer, bestimmte Produktionen zu priorisieren, sofern ein Ziel auf mehrere Produktionen passen sollte [Bot04]. Im THEN-Teil werden die Ausführungsaktionen der Produktion abgehandelt, wie beispielsweise das Setzen eines neuen Ziels im Goal-Buffer. Dies hätte zur Folge, dass eine nächste Produktion ausgewählt wird, die dem neuen Ziel entspricht. Wird in einer Produktion kein neues Ziel spezifiziert, kann es vorkommen, dass die Simulation in eine Endlosschleife gerät, in der nach jedem Schritt die gleiche Produktion wieder und wieder ausgeführt wird. In jeder Modellierung wird eine initiale Produktion verwendet, welche ein erstes Ziel setzt. Um die Simulation zu beenden, wird am Ende eine "Stop" Produktion aufgerufen.

Die meisten Anfragen der Produktionen werden an das deklarative Modul gerichtet. Dieses dient als Speicher für Wissen in Form von Fakten. Einzelne Fakten werden als Chunks gespeichert und können während der Simulation von Produktionen generiert oder abgerufen werden. Diese Chunks sind mit einem Activation-Attribut versehen, das den Nutzen des jeweiligen Chunks numerisch einordnet. Hierbei erhalten diejenigen Chunks eine erhöhte Activation, die des Öfteren abgerufen werden oder zeitnah benutzt wurden [And06]. Die Activation eines Chunks i berechnet sich aus folgender Formel:

$$A_i = B_i + \sum_j W_j S_{ji}$$

Hierbei steht B_i für die **Baselevel Activation**, die widerspiegelt, wie oft der Chunk i in der Vergangenheit vorkam und wann er initialisiert wurde. W_j repräsentiert die Erhöhung der Aktivitätsrate derjenigen Chunks, die die gleichen Werte besitzen, wie die aktuellen Belegung des Goal-Buffers. In S wird zuzüglich bestimmt, wie hoch der Assoziationsgrad der einzelnen Chunks untereinander

ist. Das Ergebnis des zweiten Summanden der Formel wird in ACT-R **Spreading Activation** genannt und wurde in der Modellierung dieser Arbeit nicht aktiviert, da dies für die serielle Analyse der Aktionsschritte in dieser Arbeit keinen Mehrwert bieten konnte. Grund hierfür ist, dass jeder Chunk genau eine Aktion abspeichert und somit auch nur mit einem Ziel assoziiert wird. Sofern eine Aktion zu einem bestimmten Simulationschritt im Goal-Buffer steht, kann folglich nur eine Beziehung zu einem einzigen Chunk hergestellt werden.

Die Berechnung von B_i basiert auf der Annahme, dass die Notwendigkeit einer Erinnerung an einen bestimmten Fakt abhängig vom Vorkommen dessen in der Vergangenheit ist [And91]. Um dies darzustellen wird in ACT-R folgende Formel genutzt:

$$B_i = ln \sum_{j=1}^{n} t_j^{-d}$$

Hierbei steht die Variable t_j für die Zeit, seit der j-ten Anwendung und d für den Degenerierungsgrad der Funktion [And06]. In der Formel wird die psychologische Theorie "Power Law of forgetting" angewandt. Diese besagt, dass Performanz über die Zeit nach einer Potenzfunktion degeneriert [Rub96]. Dies ist in der Baselevel-Funktion durch den negativen Exponenten $-d$ umgesetzt.

5.2 Szenario

Zu Beginn der Entwicklung des Demonstrators wurde zunächst ein Szenario festgelegt, das die gewünschte Interaktion und benötigte Funktionalität bestmöglich beschreibt. Die involvierten Rollen sind im Use-Case-Diagramm in Abbildung 5.2 dargestellt. Zuallererst spielt der Nutzer EXTRA. Hierbei werden xAPI Daten zu seinen Aktionen generiert und an das LRS geschickt. Nach jeder Spiel-Session kommen zwei mögliche Rollen eines Reviewers in Betracht. Ein Tutor kann die Visualisierungen des UMAI-Front-Ends und die des LRS nutzen, um die Schüleraktion zu analysieren. Ein Entwickler kann außerdem Konfigurationseinstellungen der Simulation in ACT-R vornehmen. Somit kann die Visualisierung des Front-Ends auch genutzt werden, um die Theorien und Mechanismen des deklarativen Moduls von ACT-R besser zu verstehen.

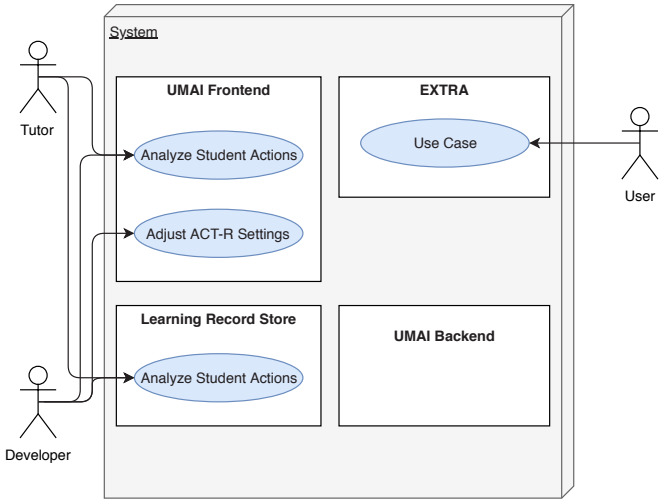

Abbildung 5.2: Use Case Diagramm

5.3 Umsetzungsvarianten

Zu Beginn der Entwurfsphase sind zwei mögliche Umsetzungsvarianten in die engere Auswahl genommen worden. Diese sollen zunächst näher beschrieben werden. In der in Abbildung 5.3 dargestellten Variante wird im ersten Schritt nach jeder Aktion des Benutzers ein xAPI Statement im Adapter generiert, das die Aktion beschreibt. Anschließend wird das xAPI-Statement zum LRS geschickt. Dies wird für jede Aktion so lange wiederholt bis der Benutzer das Spiel beendet hat. Anschließend greift UMAI auf alle aufgezeichneten Statements ab dem Spielbeginn zu. Anschließend wird eine rückwirkende Simulation in der kognitiven Architektur ausgehend vom Front-End gestartet. Diese Simulation kann in verschiedenen Konfigurationsvarianten beliebig oft wiederholt werden. Nach jeder Simulation werden in der kognitiven Architektur angefallene Daten im Front-End grafisch dargestellt.

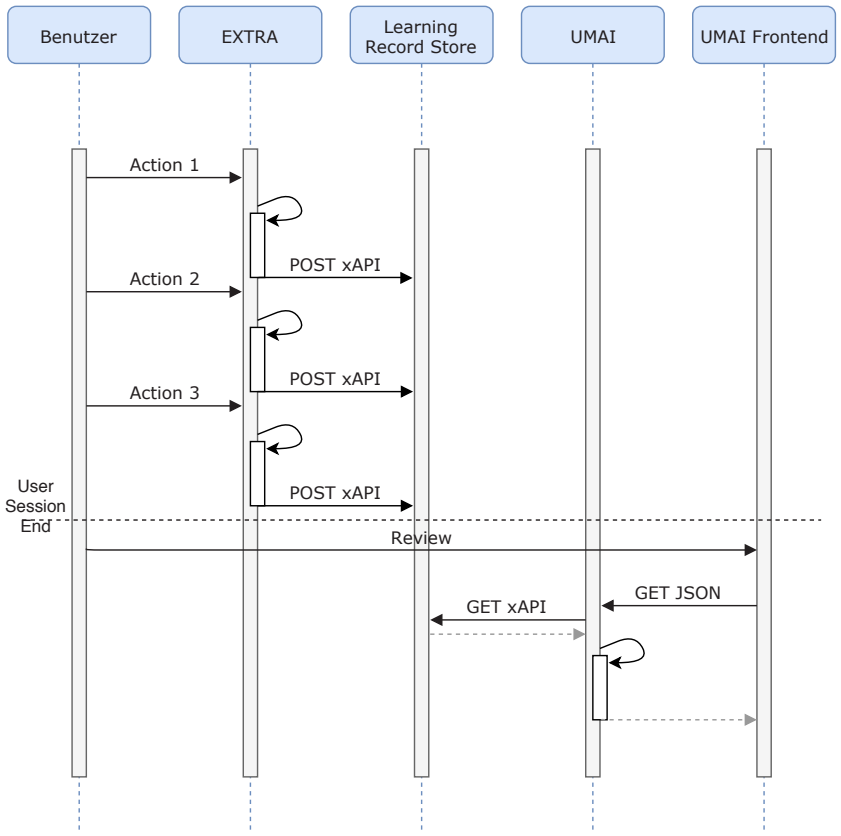

Abbildung 5.3: Sequenzdiagram der Systeminterkation in Variante 1

In der zweiten Variante, die in Abildung 5.4 dargestellt ist, wird nach jeder Aktion zusätzlich ein Ping vom EXTRA Server an UMAI geschickt. So wird jede Aktion des Benutzers direkt in der kognitiven Architektur nachsimuliert. Anschließend wird die Simulation bis zum nächsten Ping angehalten. Nach Beendigung der Spiel-Session wird wie in der ersten Variante auf die von UMAI produzierten Daten vom Front-End aus zugegriffen.

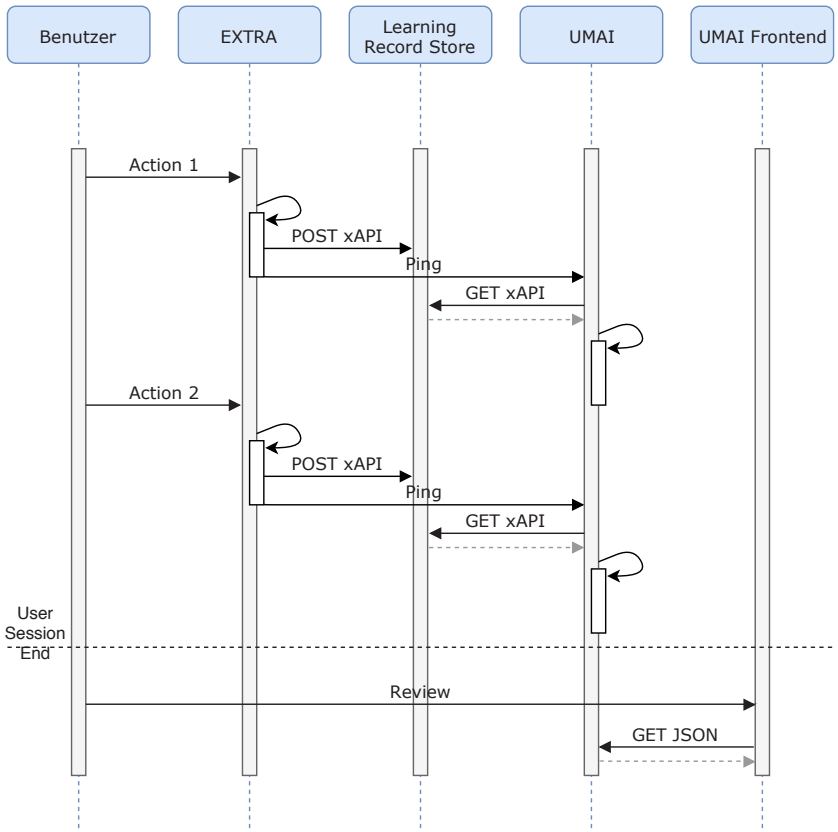

Abbildung 5.4: Sequenzdiagram der Systeminterakation in Variante 2

5.4 Serverseitige Technologien

Im Folgenden sind die in der Implementierung verwendeten Technologien näher beschrieben. Python ACT-R wurde in der unter Abschnitt 4 dokumentierten Nutzwertanalyse aus einer Auswahl bestimmt. Zur Beschleunigung der Entwicklung der API von UMAI, wurde das Framework Flask-RESTPlus für Python benutzt, das automatisiert eine Swagger UI generiert.

5.4.1 Flask-RESTPlus

Flask-RESTPlus[1] ist eine Erweiterung für Flask, das Unterstützung für die Erstellung von REST-APIs bietet. Flask-RESTPlus unterstützt "Best Practices" mit minimalem Setup [Hau14]. Flask[2] ist ein für Python entwickeltes Microframework. In folgendem Codeauszug ist eine vollständig funktionsfähige "hello World" API in Flask-RESTPlus implementiert:

1 `https://flask-restplus.readthedocs.io/en/stable/`
2 `http://flask.pocoo.org/`

```
1  from flask import Flask
2  from flask_restplus import Resource, Api
3
4  app = Flask(__name__)
5  api = Api(app)
6
7  @api.route('/hello')
8  class HelloWorld(Resource):
9      def get(self):
10         return {'hello': 'world'}
11
12 if __name__ == '__main__':
13     app.run(debug=True)
```

Source Code 5.1: 'Hello World' Programm in Flask-RESTPlus

In dem Programmtext wird eine Ressource mit zugehöriger Route erstellt, auf die mit der http-Methode `GET` zugegriffen werden kann. Die Rückgabedaten sind in Zeile 10 spezifiziert. Flask-RESTPlus generiert zuzüglich zu jeder Route automatisch eine Swagger-Dokumentation, die im nächsten Abschnitt 5.4.2 näher erläutert wird.

5.4.2 Swagger-UI

Swagger[1] hat sich als der Standard für die Modellierung und Beschreibung von REST-APIs etabliert [Hau]. Swagger ermöglicht eine Synchronisierung der Dokumentation mit allen Änderungen an den REST-Services während der Implementierung [Var15]. Die generierte Dokumentation ist interaktiv und dient dem Entwickler zum Testen der einzelnen Services. Die Dokumenation wird auf Basis einer OpenAPI-Spezifikation generiert, die in den Modellierungssprachen YAML oder JSON beschrieben werden kann [Sma18]. In der Spezifikation werden alle Routen zu den Ressourcen, unterstützte Methoden, mögliche Struktur der Input- und Output-Parameter sowie Authentifizierungsmethoden festgelegt. In der Swagger-UI wird die Struktur einer Anfrage in einem Editorfenster angezeigt, sodass der Entwickler die API mit Beispielsparametern testen kann. Falls eine Anfrage abgesandt wurde, wird der Inhalt (engl. Payload) der Antwort inklusive Statuscode visualisiert.

5.4.3 Python ACT-R

Python ACT-R[2] ist eine Reimplementierung der ursprünglich in LISP geschriebenen kognitiven Architektur ACT-R. Die Intention des Python ACT-R-Projekts ist es, ein besseres Verständnis der ACT-R-Theorie zu erlangen [Ste06]. In diesem Projekt diente die Implementierung in Python grundsätzlich dazu, ACT-R im Web einsetzen zu können. Folgendes Programm nutzt Klassen der ccm-Bibliothek, die die Funktionalität von Python ACT-R bereitstellt. Dieses ist in Codeauszug 5.2 dargestellt und zeigt eine simple Modellierung eines Agenten implementiert, welcher mit drei Produktionsregeln simuliert wird [Bra15].

1 http://swagger.io
2 https://sites.google.com/site/pythonactr/

```
1  import ccm # import ccm module library for Python ACT-R classes
2  from ccm.lib.actr import *
3
4  class MyAgent(ACTR):
5      focus=Buffer() # Creating the goal buffer for the agent
6      def init(): # this rule fires when the agent is instantiated.
7          focus.set("sandwich bread") # set focus buffer to direct
           ↪ program flow
8      def bread_bottom(focus="sandwich bread"): # if focus="sandwich
       ↪ bread" , fire rule
9          print "I have a piece of bread"
10         focus.set("stop") # set focus buffer to direct program flow
11     def stop_production(focus="stop"):
12         self.stop() # stop the agent
```

Source Code 5.2: Beispiel eines Agenten in Python ACT-R

Als erstes wird in Zeile 4 ein Agent definiert, welcher von der CCM-Suite[1] bereitgestellten ACT-R Klasse erbt. Dieser ist standardmäßig mit einem Produktionsmodul ausgestattet. In Zeile 5 wird der Goal-Buffer initalisiert, der in Python ACT-R als Focus-Buffer benannt wird. Die verschiedenen Arten der Module und ihre Funktionsweisen sind in Abschnitt 5.1 beschrieben. Als Nächstes wird in Zeile 6 eine initiale Produktion definiert, die vom Agenten zum Start ausgeführt wird. Darauf wird in Zeile 7 der Focus-Buffer auf einen String-Wert gesetzt. Das zentrale Produktionsmodul prüft nach jedem Schritt in der Simulation, ob Produktionregeln existieren, die der Belegungen des Focus Buffers und allen weiteren verwendeten Buffer entsprechen. In Zeile 8 ist nun eine weitere Produktionsregel definiert, die das Ziel des Focus Buffers erfüllt, welcher in der initialen Produktion gesetzt wurde. In den Zeilen 9-12 wird nach dem gleichem Prinzip vorgegangen und der Agent in der letzten “Stop” Produktion angehalten.

5.5 Front-End Technologien

Bei der Umsetzung des Front-Ends wurde zunächst nach Technologien recherchiert, die den aktuellen Webstandards entsprechen, jedoch gleichzeitig eine beschleunigte Entwicklung ermöglichen. Hierbei wurde mit Polymer[2] eine Lösung gefunden, die den Anforderungen entsprach. Polymer ist eine von Google entwickelte Javascriptbibliothek, die auf dem Konzept der “Web-Components” aufbaut [Pol17]. Web-Components sind Elemente wie Eingabefelder oder Schaltfächen, die in jedem Front-End benötigt werden. Entwickler greifen bei der Programmierung auf diese zurück, indem nur bestimmte Metadaten zur Anpassung an die jeweilige Funktionalität angegeben werden. Eine in Polymer entwickelte Seite ist ein Baum von Elementen, die in der Wurzel an das Document Object Model (DOM) angehängt werden. Jede Benutzeransicht wird in der Regel wiederum in einem Element definiert, das von der `PolymerElement` Klasse erbt. Des Weiteren wurde zur Visualisierung

1 https://github.com/tcstewar/ccmsuite
2 https://www.polymer-project.org/

der in UMAI generierten Aktivitätsraten Google Charts[1] eingesetzt, das eine Javascriptbibliothek zur Umsetzung jeglicher Grafiken ist. Beide Bibliotheken sind kostenlos und frei zugänglich.

1 https://developers.google.com/chart/

6

Realisierung des Demonstrators

Dieses Kapitel stellt das Ergebnis der Implementierungsarbeit des Projektes vor. Zunächst wird die Modellierung des studentischen Modells in Python ACT-R thematisiert. Hierbei wird auf die verschiedenen zugrundeliegenden ACT-R Mechanismen näher eingegangen. Anschließend wird die Funktionsweise der entwickelten API dargelegt und die Interaktion der Services untereinander beschrieben. Außerdem wird das entwickelte Front-End präsentiert.

6.1 Modellierung in Python ACT-R

Im Verlauf der Arbeit wurde ein Modell in Python ACT-R entwickelt, welches einen Mehrwert zur Analyse der Schülerinteraktion bietet. Die Modellierung ist unabhängig von der Domäne des Lernsystems und analysiert das Vorkommen und die Häufigkeit von Aktionsschritten in Abhängigkeit der Zeit. Wie in Abschnitt 5.1 beschrieben wurde ist in ACT-R jeder Chunk mit einer Aktivitätsrate versehen. Diese degeneriert über die Zeit, sofern der jeweilige Chunk nicht neu angefragt wird. In der Modellierung wird aus dem aus xAPI-Statements bestehendem Activity-Stream ein neuer Action-Stream generiert, welcher nur aus den Beschreibungen der Aktionen besteht. Dies ist genauer in Abschnitt 6.3 beschrieben. In der Modellierung werden dynamisch für jede dieser Aktionsbeschreibungen Chunks des deklarativen Speichers angelegt. Ziel der Simulation ist es die Interatkionsschritte auf Basis der Aktivitätsraten dieser Chunks zu analysieren. Sofern Aktionen mehrfach vorkommen und für diese im Verlauf der Abarbeitung des Action-Streams schon Chunks angelegelegt wurden, werden Anfragen an das deklarative Modul gestellt. Diese Anfragen können fehlschlagen, sofern die Aktivitätsrate des jeweiligen Chunks der jeweiligen Aktion unter einer Grenze, auch Threshold genannt, liegt. Diese Grenze sowie weitere Parameter können die Simulation beeinflussen und sind am Ende dieses Abschnitts genauer erklärt.

Für jede Aktion wird zunächst eine Basisproduktion generiert, welche in dem Code-Auszug 6.1 dokumentiert ist. In der Produktion wird zwischen zwei Fällen unterschieden: Im ersten Fall wurde eine Aktion zum ersten Mal vom Benutzer ausgeführt und wird deshalb dem deklarativen Speicher in Form eines Chunks hinzugefügt. Außerdem wird Die Nummer der Aktion in dem `chunk_numbers` Array gespeichert und der Fokus-Buffer auf die nachfolgende Aktion gesetzt. Sofern die Aktion zuvor vorkam, wird der `else`-Teil der Produktion aufgerufen. Für jede Aktion wird der IF-Teil der Produktion nur einmal aufgerufen und ein Chunk der Form `action_0:"<action_descriptor>"` angelegt. In Python ACT-R wird eine Anfrage an einen deklarativen Chunk gestellt, indem der Funktion `DM.request() <"action_0:?action">` übergeben wird. Bevor diese Anfrage abgeschickt werden

kann, wird die Chunk-Nummer bestimmt, wann die jeweilige Aktion das erste Mal vorkam. Im Anschluss wird der Focus nicht auf die nächste Aktion gesetzt, sondern zunächst auf die Produktion, die zu der Chunk-Nummer passt. Diese ist somit die Produktion an der Stelle, an der die Aktion das erste Mal vorkam. Dabei wird der Fokus nicht auf die Basisproduktion gesetzt, sondern auf die ebenfalls zu jeder Aktion generierten Produktionen `forget` und `remember`, welche in den nächsten Code-Beispielen erklärt werden. Im letzten Teil der Produktion werden die Aktivierungen gespeichert, die nach der Simulation im Front-End grafisch dargestellt werden.

```
1  def action_0(focus='action_0'):
2          countHelper = -1
3          if (action_stream[len(fired_actions)] in new_actions):
4              new_actions.remove(action_stream[len(fired_actions)])
5              dmstring = 'action_0:' +
                ↪  action_stream[len(fired_actions)]
6              chunk_numbers[action_stream[len(fired_actions)]]= str(0)
                ↪  # store the chunk number
7              fired_actions[str(len(fired_actions))]=
                ↪  action_stream[len(fired_actions)] # store the fired
                ↪  action and the counter time
8              DM.add(dmstring)
9              focus.set('action_' + str(len(fired_actions)))
10         else :
11             countHelper = 0
12             current_chunk_number =
                ↪  chunk_numbers[action_stream[len(fired_actions)]]
13             dmstring = 'action_' + current_chunk_number + ':?action'
14             DM.request(dmstring)
15             focus.set('action_' + current_chunk_number + '_')
16
17         #store activations
18         list_of_chunks = self.DM.find_matching_chunks('')
19         actions_temp = [str(i).split(':')[0] for i in list_of_chunks]
20         for i,chunk in enumerate(list_of_chunks):
21             chunk_activations[str(len(fired_actions)+countHelper)]
                ↪  [str(chunk[actions_temp[i]])] =
                ↪  round(self.DM.get_activation(chunk), 3)
```

Source Code 6.1: Basisproduktion einer Aktion

Für den Fall, dass die für die Aktionen generierten deklarartiven Chunks eine Aktivierung kleiner gleich dem `threshhold` besitzen und die Aktion abgerufen wird, wird die in Code-Auszug 6.2 dargestellte Produktion aufgerufen. Hier wird innerhalb der Produktionsregel einzig der Fokus auf die seriell folgende Aktion gesetzt. Ist die Aktivierung größer als der `threshhold`, so wird die `remember` Produktion aufgerufen, welche in folgendem Code-Auszug dargestellt ist. Im Falle des Aufrufs dieser Produktion kann der jeweilige Chunk nicht reaktiviert werden.

```
1 def forgot_0(focus='action_0_', DMBuffer=None, DM='error:True'):
2         # DMBuffer=none means the buffer is empty
3         # DM='error:True' means the search was unsucessful
4         print ("I forgot something")
5         if len(action_stream) == len(fired_actions) :
6             focus.set('stop')
7         else :
8             fired_actions[str(len(fired_actions))]=
               ↪ action_stream[len(fired_actions)]
9             focus.set('action_' + str(len(fired_actions)))
```

Source Code 6.2: Vergessensproduktion einer Aktion

In der in Code-Auszug 6.3 dargestellten `remember`-Produktion wird zunächst wie in der `forget`-Produktion geprüft, ob alle Aktionen schon abgearbeitet wurden. Für diesen Fall wird der Focus auf "Stop" gesetzt, damit die Simulation des Agenten im nächsten Schritt mit der `stop`-Produktion beendet werden kann. Falls der Abarbeitungsstand des Action-Streams noch nicht das Ende erreicht hat, wird der Focus auf die nächste Aktion gerichtet. Anders als in der originalen Implementierung ist es in Python ACT-R notwendig, die abgerufene Aktion dem deklarativen Speicher neu hinzuzufügen, um die Erhöhung der Aktivierung des jeweiligen abgerufenen Chunks zu erreichen. Dies wird in Zeile 6 des Code-Auszugs 6.3 erreicht.

```
1 def remember_0(focus='action_0_', DMBuffer="action_0:?action"):
2         print ("I remember action_0:",action)
3         if len(action_stream) == len(fired_actions) :
4             focus.set('stop')
5         else :
6             DM.add('action_0:?action')
7             DMBuffer.clear()
8             fired_actions[str(len(fired_actions))]=
               ↪ action_stream[len(fired_actions)]
9             focus.set('action_' + str(len(fired_actions)))
```

Source Code 6.3: Erinnerungsproduktion einer Aktion

Diese Produktionen werden in der Simulation inkrementell bis zum Ende durchlaufen. Währenddessen werden die Aktivierungen nach jedem Schritt aufgezeichnet und in einem JSON-Objekt gespeichert. Die Aktivierungen können beispielsweise aufzeigen, dass ein Benutzer eine bestimmte Aktion in dem Spiel über längere Zeit nicht ausgeführt hat. Dies kann ein Hinweis für den Entwickler des Spiels sein, dass die Aktion in der aktuellen Gestaltung eines bestimmten Levels zu wenig benutzt werden soll. Sofern der Benutzer beispielsweise einen schlechten Score hat, so kann ein Tutor falsches Spielverhalten genauer analysieren.

Die Simulationsart des Agenten wird in verschiedenen Parametern zu Beginn bestimmt. Diese beeinflussen die Aktivitätsrate der einzelnen Chunks und sind im Folgenden erklärt:

Threshold Chunks müssen eine Aktivierung haben, die größer oder gleich dem Wert ist, der im Treshold bestimmt wird. Liegt die Aktivierung unter dieser Grenze ist ein Zugriff auf den jeweiligen Chunk nicht möglich.

Noise Der Noise Parameter legt den Zufall fest, der nach jedem Schritt der Simulation unabhängig neu berechnet wird [Bra14].

Basenoise In dem Basenoise Parameter wird ein Zufallsmaß festgelegt, welches sich auf einen bestimmten Chunk bezieht. Im Gegensatz zum Noise Parameter wird dieser Zufall nicht nach jedem Simulationsschritt neu berechnet. Für gewöhnlich wird dieser Parameter mit 0 initialisiert [Bot04].

Decay Decay legt fest, wie schnell die Aktivitätsraten degenerieren sollen. Sofern menschliche Daten simuliert werden, wird der Paramter mit `0.5` initialisiert [Ste06].

Latency Mit der Latenz wird die Beziehung zwischen Aktivierung und der Dauer bis zum Aufruf eines Chunks gesteuert [Bra14].

6.2 Dynamische Generierung des Python ACT-R Modells

Für die Generierung der Simulation wird ein Filegenerator-Script aufgerufen, welches dynamisch die Modellierung auf Basis des `action_stream` umsetzt.

```
1 for i,x in enumerate(global_variables.action_stream):
2     actions_methods_stream.write(Template(action_tpl)
      ↪  .substitute(action_nr=i))
3     actions_methods_stream.write('\n')
```

Source Code 6.4: Dynamische Generierung des ACT-R Modells

In Code-Auszug 6.4 ist der Loop dargestellt, welcher über das `action_stream` Array iteriert und für jede Aktion die in Abschnitt 6.1 thematisierte Modellierung generiert. Dies ist mithilfe eines Templates umgesetzt, in welchem an bestimmten Stellen der Zähler des Loops als Nummerierung der Aktionen eingesetzt wird.

6.3 REST API

Die REST API des entstandenen UMAI-Services ist, wie bereits erwähnt, mit einer Swagger-UI dokumentiert und lässt sich mittels des in Abbildung 6.1 festgehaltenen Front-End testen.

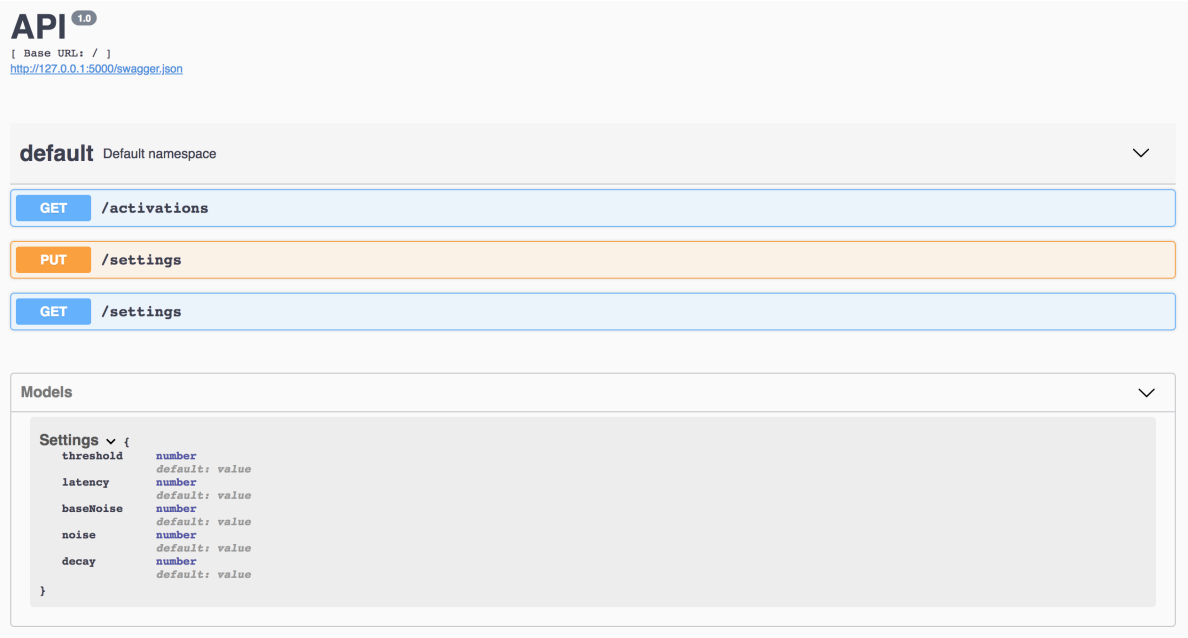

Abbildung 6.1: Swagger-UI der UMAI API

Die Funktionsweise der verwendeten Routen ist im Folgenden näher beschrieben.

Abfrage der Chunk Aktivierungen

Beim Aufruf dieser Route werden zunächst die xAPI-Statements vom LRS abgerufen. Dies teilt sich in zwei Anfragen auf, welche mit der requests API (`http://docs.python-requests.org/en/v2.7.0/`) implementiert wurden. Um nur die aktuellen Statements abzurufen, wird zunächst das letzte Statement angefragt, welches die Initialisierung des Spiels aufzeichnet. Dies ist in folgendem Code-Abschnitt 6.5 realisiert:

```
1 r = requests.get(<uri_get_initialized>,
    ↪ headers={'X-Experience-API-Version': '1.0.3'}, auth=(<username>,
    ↪ <password>))
2
3 json_intitialized_statements = r.json()
4
5 timestamp =
    ↪ json_intitialized_statements['statements'][0]['timestamp']
```

Source Code 6.5: GET Anfrage an LRS zur Speicherung des Timestamps des letzten `initialized` Statements

Mit der Anfrage in Zeile 1 werden alle xAPI-Statements darauf gefiltert, ob diese das Verb “initialized” verwenden. Darauf wird auf den Timestamp des ersten xAPI-Statements in dem zurückgegeben JSON-Array zugegriffen. Dieser wird dann in einer nächsten Anfrage dazu benutzt, alle Statements abzurufen, die nach diesem Timestamp in dem LRS gespeichert wurden. Anschließend wird über diese Statements iteriert und die jeweilige Beschreibung in einem `action_stream` Array gespeichert. Dieses Array ist in einem weiteren `global_variables` Script gespeichert und dient als Grundlage der dynamischen Generierung des ACT-R Modells.

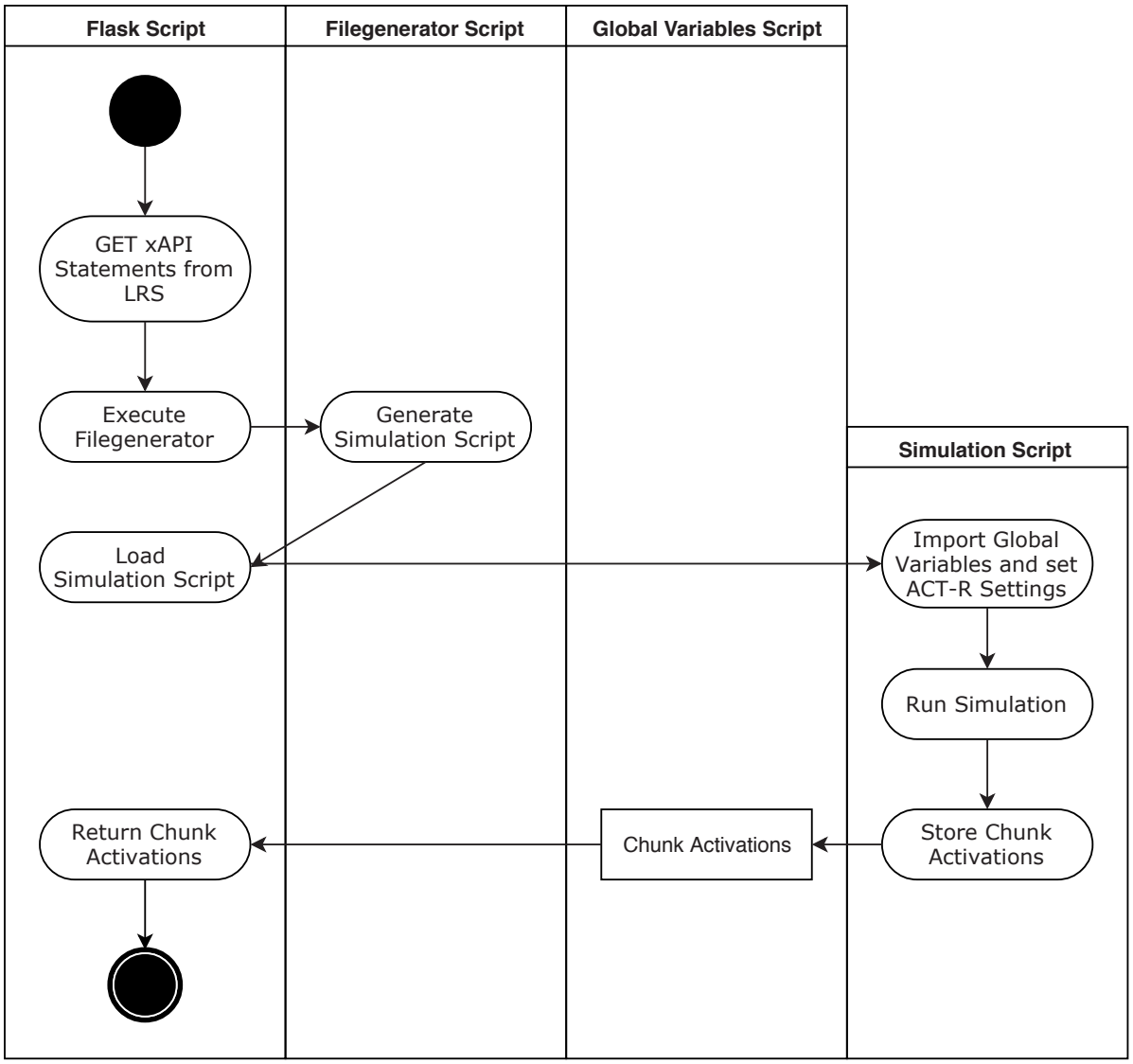

Abbildung 6.2: Abarbeitungsprozess einer `GET` Anfrage

Nachdem die aktuellen Aktionen des Spielers gespeichert sind, wird das File-Generator-Script aufgerufen, das die Simulation in Python ACT-R auf Basis der Aktionen generiert. Wie in Abbildung 6.2 dargestellt, wird das Simulations-Script nach der Ausführung des Filegenerators in des Flask Script importiert. Bei dem Versuch das Simulations-Script wiederum vom Flaskscript mit einem

`exefile()` Ausdruck auszuführen, kam es zu Fehlern in der von Python ACT-R bereitgestellten CCMSuite[1]. Der Fehler trat jedoch nicht mehr beim Importieren des generierten Scripts auf. Da in Python imports nur ein einziges mal ausgeführt werden, wird das Script im nächsten Schritt mit `reload()` aktualisiert, um neue Anfragen an die API und daraus resultierende Veränderungen des Simulationsscriptes zu unterstützen. Zuletzt werden die während der Simulation berechneten Aktivierungswerte der deklarativen Chunks als Antwort zurückgegeben.

Abfrage der Einstellungswerte

Die Settings Route gibt alle Werte der in dem `global_variables` Script gespeicherten Konfigurationsvariablen zurück. Initial sind diese mit Beispielswerten belegt. Sofern Änderungen im Front-End vorgenommen werden, wird eine PUT Anfrage gestellt und die neuen Werte gesetzt. Wie die einzelnen Variablen die Simulation beeinflussen, ist in Abschnitt 6.1 näher beschrieben.

6.4 Front-End

Das Front-End wurde auf Basis der Polymer-Bibliothek implementiert, welche in Abschnitt 5.5 vorgestellt wurde. Die Navigationsstruktur gliedert sich in eine Einstellungsseite, eine Anleitungseite und einer Seite zur Visualisierung der während einer Simulation angefallenen Aktivierungswerte der deklarativen Chunks.

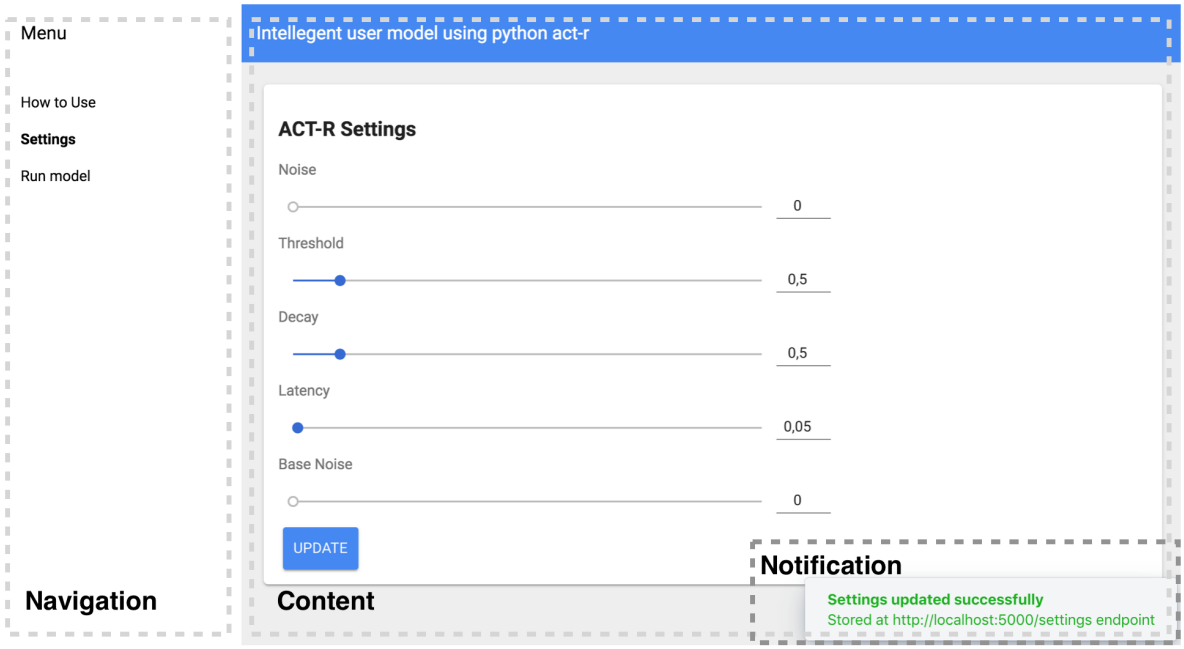

Abbildung 6.3: Einstellungsseite

1 https://github.com/tcstewar/ccmsuite

Beim Design wurde darauf geachtet, nach jeder abgesendeten REST Anfrage an das Backend, dem Benutzer den Erfolg oder mögliche Fehler in Form von Benachrichtigungen anzuzeigen. In Abbildung 6.3ist die Einstellungsseite gezeigt. Im linken Bereich befindet sich die Navigation. Das resultierende Front-End nutzt CSS, um “Responsive Design” umzusetzen. Hierbei wird die Webseite auf die Bildschirmgröße des verwendeten Browsers angepasst. In Kapitel 7.1 sind Beispiele der Interaktion aufgeführt.

7

Anwendungsfall und Diskussion

In diesem Kapitel soll das in Kapitel 5.2 im Groben beschriebene Szenario detalliert ausgeführt und technisch evaluiert werden. Somit wird die Orchestrierung der verschiedenen Technologien erklärt und verdeutlicht.

7.1 Szenario

Zunächst werden im Folgenden die involvierten Rollen des Szenarios vorgestellt.

Tutor Der Tutor David hat die Aufgabe im Rahmen des Nato JISR Teilnehmer in die verschiedenen Prozesse der heterogenen Hardware Systeme und Softwareanwendungen einzuführen. Hierbei nutzt er das am Fraunhofer IOSB entwickelte Serious Game EXTRA, um den Lernprozess seiner Schüler zu verbessern. Zusammen mit dem Entwickler John, welcher im nächstem Abschnitt vorgestellt wird, hat er Szenarien für das Spiel entwickelt. Jedes Szenario ist repräsenativ für die wesentlichen Prozesse der Informationsverarbeitung. Dabei wurde darauf geachtet, dass sich die Schwierigkeit der verschiedenen Szenarien in den jeweiligen Spiellevels wiederspiegelt. Des Weiteren wird grundlegendes Wissen in Form von wiederholender Anwendung intensiviert.

Entwickler Der Entwickler John ist ein am Fraunhofer IOSB angestellter Mitarbeiter, der die Aufgabe hat, ein ITS auf Grundlage der ELAI-Architektur zu konzeptionieren, welche in Abschnitt 1.3 vorgestellt wurde. Hierbei ist eine kognitive Modellierung der Domäne von EXTRA geplant. John befindet sich momentan in der Einarbeitungsphase in ACT-R und ist mit der Komplexität der Architektur konfrontiert, da er kein Vorwissen in kognitiver Modellierung besitzt. Deshalb nutzt er UMAI, um die Funktionsweise von ACT-R schneller zu verstehen.

Student Alan ist ein Teilnehmer des Nato JISR Programms und wurde dem Kurs von Tutor David zugeordnet. Hierbei durchläuft er die von David konzeptionierten Levels, um das Wissen spielerisch zu erlernen.

Zuallererst fängt Alan an EXTRA zu spielen, nachdem ihm das Userinterface von Tutor David erklärt wurde. Hierbei legt er als erste Aktion eine Produktionsstraße an, die von einer Fabrik zu einem Marktplatz führt. Dies führt dazu, dass die ersten Produkte über die entstandene Verbindung an die

Fabrik ausgeliefert werden. Nachdem 8 Produkte am Marktplatz eingegangen sind verkauft Alan diese im Spiel und gewinnt damit das erste Level.

Nachdem die erste Spiel-Session von Alan beendet ist, analysieren David und John die Interaktion von Alan im UMAI-Front-End. Hierbei navigiert John vorerst auf den Reiter “Run Model” und startet die Simulation. In Abbildung 7.1 ist das Ergebnis der Anfrage dargestellt.

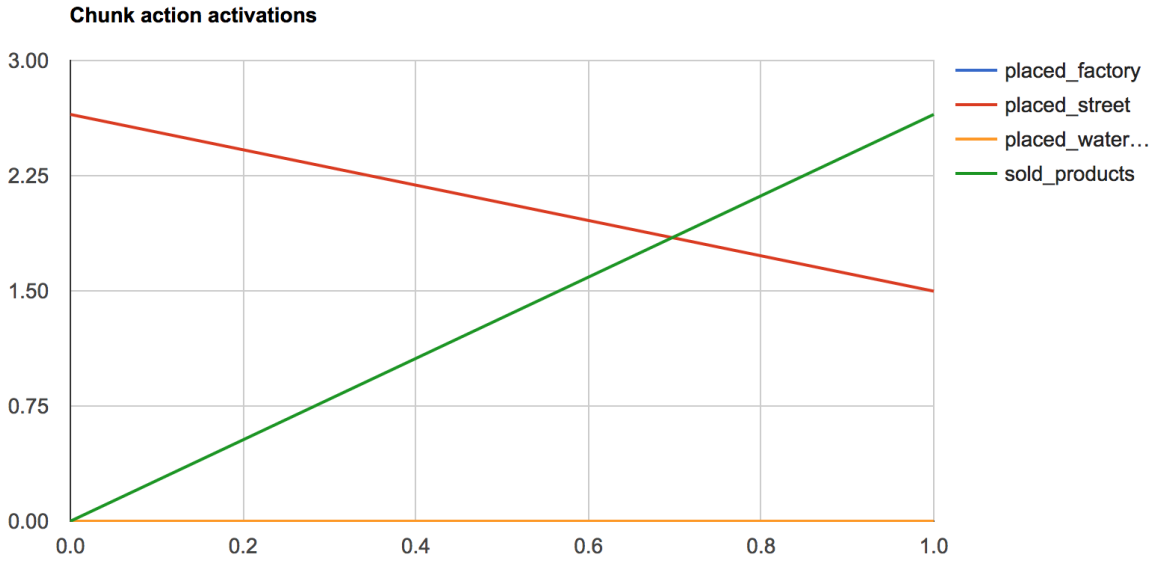

Abbildung 7.1: Baselevel-Aktivierung der ersten beiden Aktionen

John kann mit dem Diagramm noch nicht sehr viel anfangen. Ersichtlich ist nur, dass Aktivierungen zu den beiden Aktionen des Schülers Alans angezeigt werden. Hierbei nimmt die eine Aktivierung linear ab und eine weitere linear zu. Währenddessen hat Alan mit dem zweiten Level begonnen. John führt nun eine zweite Simulation aus und bekommt das in Abbildung 7.2 visualisierte Diagramm angezeigt.

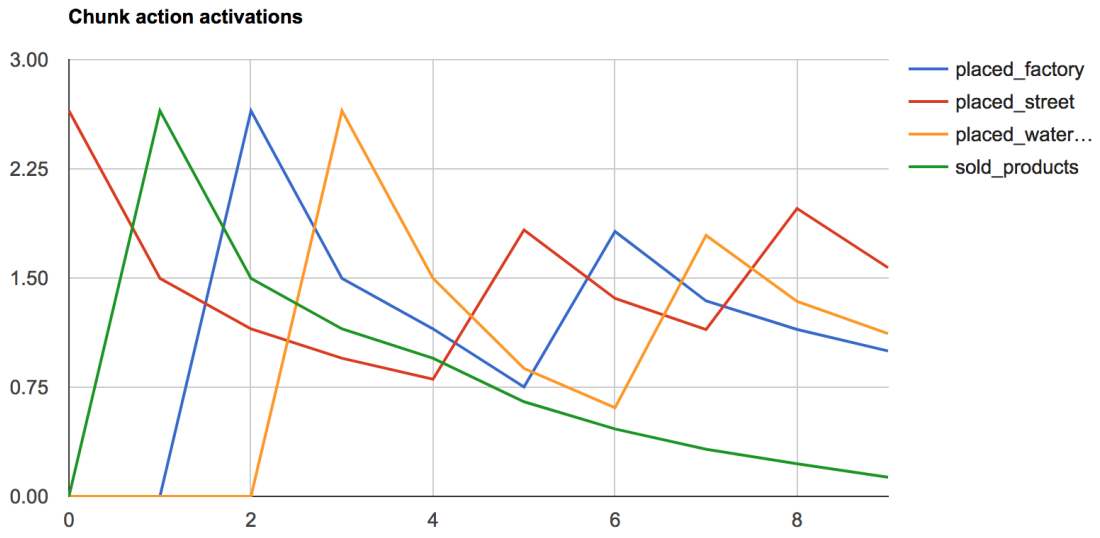

Abbildung 7.2: Baselevel-Aktivierung mehrerer Aktionen

John erkennt zuerst, dass die Aktivierungen nicht linear verlaufen, wie nach der ersten Simulation anzunehmen war. In dem Diagramm fällt sowohl dem Tutor David, als auch dem Entwickler John auf, dass sich der Aktivierungswert der Aktion “sold_products” vom Rest absetzt und einen geringeren Wert annimmt. David wendet ein, dass sein Schüler diese Aktion, jedoch während des zweiten Levels benutzt hatte. Nun wechselt David auf die Einstellungsseite, um den Threshold auf 0 zu setzen.

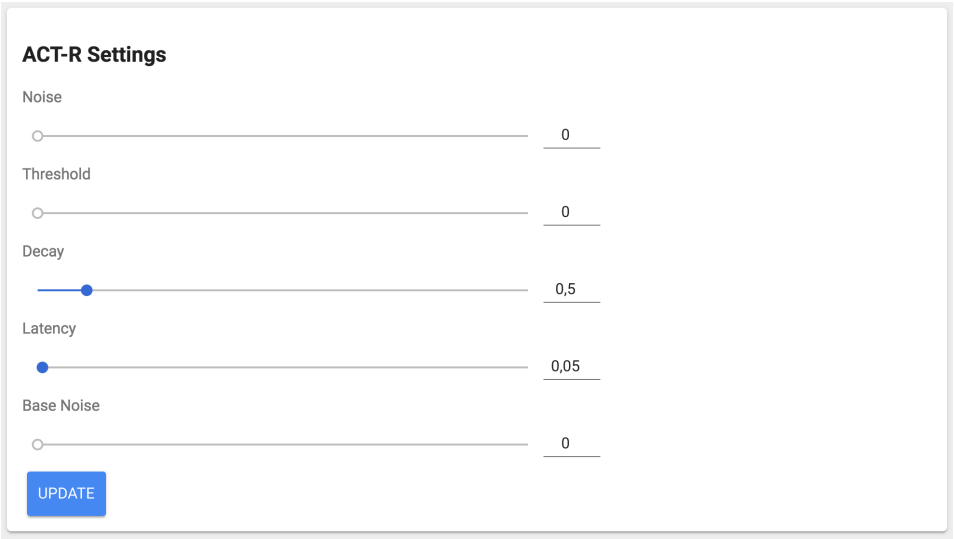

Abbildung 7.3: ACT-R Einstellungsbereich

Nachdem John die Einstellung vorgenommen hat und ihm eine Benachrichtigung anzeigt, dass die Änderung erfolgreich war, führt er wieder eine Simulation aus.

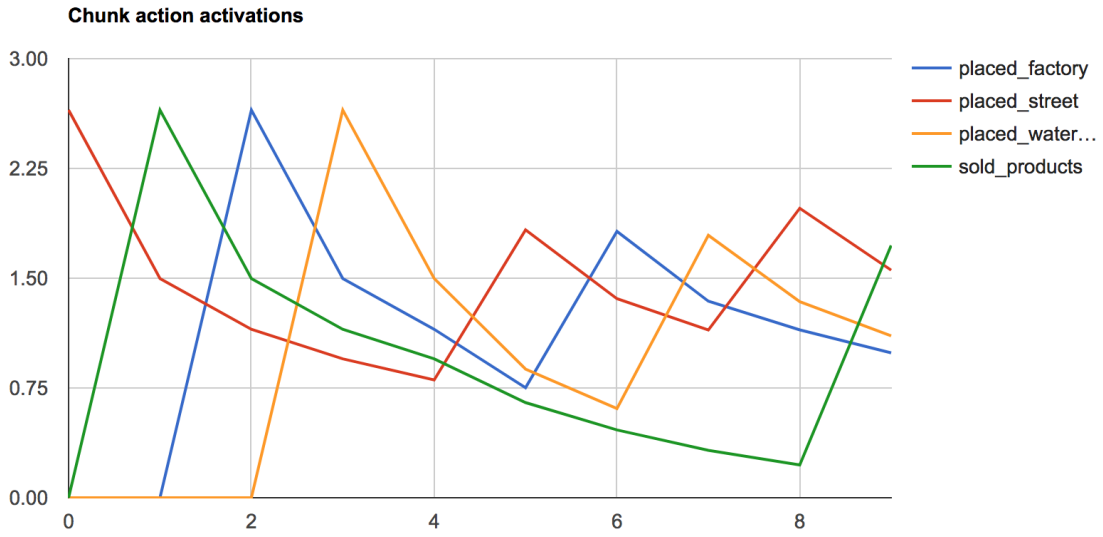

Abbildung 7.4: Baselevel-Aktivierung der Aktionen nach Anpassung des Thresholds

Mit dem in Abbildung 7.4 gezeigtem Diagramm, erkennt John, dass seine Vermutung richtig war und diese Aktion in der ACT-R Simulation vergessen wurde. David erkennt, dass diese Aktion in dem Design seiner Szenarios eventuell zu wenig vorkommt und wird dies in Zukunft beachten.

7.2 Erklärung

Grundlage der in dem Szenario analysierten Daten ist die Generierung von xAPI Statements, während der Spiel Session des Schülers Alan. Wie in der in Abbildung 7.5 gezeigten Gesamtarchitektur werden die Statements an das LRS weitergeleitet und dort gespeichert.

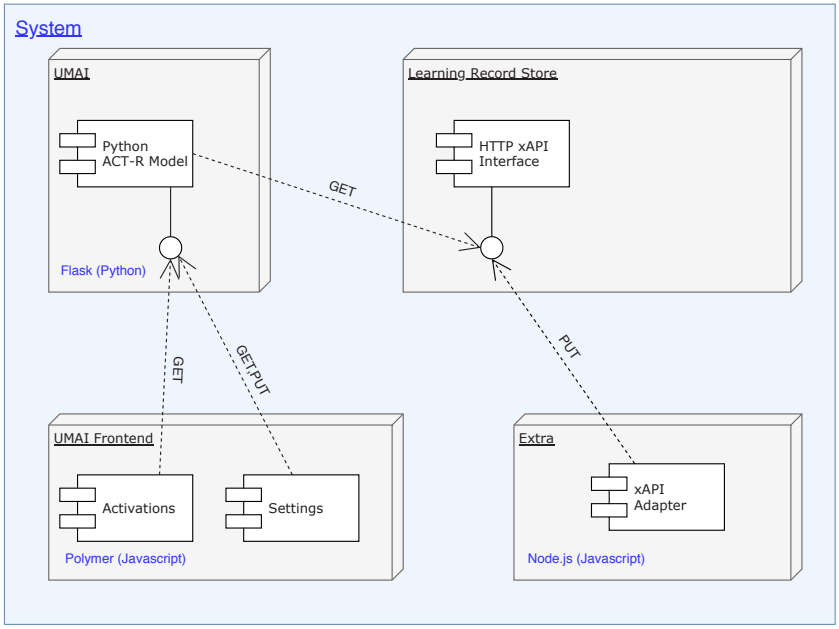

Abbildung 7.5: Systemarchitektur

Bei jeder von John ausgeführten Simulation der Aktivierungen zu den Aktionen wird eine asynchrone Ajax[1] Anfrage an das UMAI-Backend gesandt. Anschließend werden die Statements von UMAI über eine REST Anfrage vom LRS bezogen. Die genauere Funktionsweise, wie nur die Statements der aktuellen Session angefragt werden, ist in Abschnitt 6.3 beschrieben. Der nun bezogene Activity-Stream wird darauf auf einen Action-Stream reduziert, welcher nur eine Kurzbeschreibung der ausgeführten Aktionen von Alan binhaltet. Anschließend wird auf Basis dessen die Modellierung in Python ACT-R generiert und die dabei produzierten Aktivierungen zurückgegeben. Dass John die Simulierung sogar während der Spiel-Session von Alan aktualisieren kann, liegt daran, dass UMAI bei jeder Anfrage alle Statements seit der Initialisierung des Spiels bezieht. Wie von John richtig vermutet, wurde in der zweiten Simulation eine Aktion vergessen. Grund hierfür war, dass die Aktivierung der Aktion unter der initial gesetzten Threshold-Grenze von 0,5 lag. So wurde die unter Abschnitt 6.1 genauer beschriebene `forgot` Produktion aufgerufen, da der Zugriff auf den zu der Aktion modellierten Chunk verweigert wurde. In der letzten Simulation hingegen wurde der Threshold von John so gewählt, dass die Aktivierung der Aktion darüber lag. Somit spiegelte sich dies in der Visualisierung in Form einer Reaktivierung wieder.

7.3 Diskussion

Zu Beginn der Recherche dieser Arbeit stellten sich zwei Varianten der Umsetzung von personalisierter Adaptivität in Lernsystemen heraus, die im Kapitel 2 vorgestellt wurden. Der grundlegende Unterschied liegt im Einsatz und der Art der künstlichen Intelligenz, beide Varianten setzen jedoch eine Modellierung des Wissens der Domäne voraus. In der ersten Variante wird diese Modellie-

1 `https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started`

rung mithilfe kognitiver Architekturen bewerkstelligt, während die Zweite auf der Formulierung von Constraints basiert. Diese Modellierung des Wissens ist der zentrale Ausganspunkt der in Abschnitt 3.4 thematisierten intelligenten Tutoringsysteme, welche das in dieser Arbeit angestrebte Ziel eines intelligenten personalisierten Benutzermodells umsetzten. So tritt, außer im Bereich der Lernanalyse, das Student-Model immer in Kombination mit einem Domain-Model auf. Der Nachteil einer kognitiven Modellierung ist die Komplexität der Modellierung in Produktionsregeln und die benötigte Erfahrung auf dem Gebiet [Sal03]. Aus diesem Grund wurden einfachere Frameworks wie Simple-ACT-R entworfen, die automatisiert Modelle generieren. Die in den jeweiligen Publikationen angeführten Internetseiten sind jedoch nicht mehr verfügbar.

Für das in Abschnitt 2.1 vorgestellte MOUT Serious Game [Bes02] wurde ein Agent in ACT-R modelliert, der automatisiert gegen menschliche Gegner spielen kann. Entwickler eines auf kognitiven Architekturen basierenden intelligenten Tutoringsystems müssen jedoch, anders als in MOUT, nicht nur Reaktionen auf bestimmte Szenarien modellieren, sondern alle möglichen Szenarien und Lösungswege der Domäne. So ist die Generierung eines autonomen Agenten wie in MOUT nicht so komplex, wie die Modellierung einer gesamten Domäne in intelligenten Tutoringsystemen. Der Grund hierfür ist, dass alle Szenarien abgedeckt werden müssen, was auch die falschen Lösungswege einschließt. Wird in dem Produktionssatz eine spezifische Kombination der Aktionen nicht modelliert, so kann dies zu fehlerhafter Bewertung der Interaktion führen. Wendet der Schüler eine richtige Lösungsstrategie an, die in dem Produktionssatz jedoch nicht vollumfänglich enthalten ist, so wird die Strategie in kognitiven Tutoren trotzdem als falsch angesehen [Ant03]. Hierbei würde das Fehlen auch nur einer Produktion das fehlerhafte Verhalten des Tutors auslösen. Im Beispiel von EXTRA ist eine Modellierung aller Lösungswege nicht möglich, da durch die Reihenfolge der Interaktion eine große Anzahl an Möglichkeiten des Durchlaufens der Levels entsteht.

In der zweiten Variante wird dieses Problem behandelt, indem nur die richtigen Lösungswege der Domäne in Form von Constraints dargestellt werden [Ohl94]. So konzentriert sich der Einsatz künstlicher Intelligenz auf das Student-Model. Hierbei werden studentische Interaktionsdaten gesammelt und mithilfe maschineller Lerntechniken wie unüberwachten Lernen analysiert. Diese Systeme können ebenso die Lernperformanz von Schülern verbessern, ohne das in Abschnitt 3.4 vorgestellte Ideal-Student-Model umzusetzten [Ohl94] [Ant03] [Sur02] [Mit01] [Mil07].

Nach der Aufgabenstellung und Zielsetzung richtete sich der Fokus jedoch auf den Einsatz kognitiver Architekturen. Das Problem der Komplexität der Modellierung konnte mit der Idee der dynamischen Generierung von Produktionsregeln aus dem xAPI Action-Stream umgangen werden. Hierbei wurde also nicht das Domain-Model entworfen, sondern direkt ACT-R im Student-Model eingesetzt. Somit ergibt sich noch kein Tutoringsystem und die resultierende Funktionalität ist in das Gebiet der Lernanalyse einzuornden. Wie bereits erwähnt wäre die Modellierung des Domänenwissens notwendig, um Hilfestellung zu bewerkstelligen, da das angewandte Wissen des Schülers im System nachvollzogen werden muss. Der Mehrwert der in der Arbeit entstandenen UMAI liegt in der Aufzeichnung von Aktivitätsdaten zu den Aktionen eines Benutzers, die nach zeitlichem Vorkommen und Frequenz bestimmt werden. Im Groben erhalten Aktionen, die des Öfteren vorkamen oder zeitnah benutzt wurden, eine erhöhte Aktivität. Die Berechnung dieser Aktivitätsraten baut auf der Theorie “Power Law of forgetting” auf, nach welcher Anwendung von Wissen und Performanz nach einer Potenzfunktion degeneriert [Rub96].

8

Zusammenfassung und Ausblick

In dieser Arbeit wurde aufbauend auf dem schon bestehendem Serious Game EXTRA ein kognitives intelligentes Benutzermodell entwickelt, das individuell auf die Aktionen eines Spielers eingeht. Hierbei wurde das Spiel durch einen Adapter erweitert, welcher Benutzerinteraktionen aufzeichnet und an einen Learning Record Store (LRS) mittels xAPI-Statements schickt. Zur Analyse dieser Daten wurde eine User Model Artificial Intelligence (UMAI) aufbauend auf der kognitiven Architektur ACT-R in Form eines Microservice entwickelt. Die kognitive Architektur wurde zuvor in einer subjektiven Nutzwertanalyse ausgewählt. UMAI greift den von einem Spieler in EXTRA generierten Activity-Stream vom LRS ab und modelliert dynamisch zu jeder Interaktion einen deklarativen Gedächtnisspeicher in ACT-R. Diese Speicher sind mit Aktivitätsmerkmalen versehen, welche über die Zeit verfallen, bis ein Zugriff auf den Speicher nicht mehr möglich ist. Auf diese Art und Weise kann das Vergessen von bestimmten Aktionen simuliert werden. UMAI stellt eine API zur Verfügung, welche es ermöglicht auf die während der Simulation generierten Aktivitätswerte zuzugreifen und verschiedene Grundeinstellungen vorzunehmen, die den Verfall der Aktivitätsmerkmale beeinflussen.

Dazu wurde ein Front-End entwickelt, das auf die API-Schnittstellen zugreift und den Verlauf der Aktivitätswerte visualisiert. Das entstandene System wurde domänenunabhängig konzeptioniert und kann als Ausgangspunkt dienen, ein ITS im Hinblick auf kognitiven Architekturen weiterzuentwickeln. Mit Hilfe des Front-Ends kann Entwicklern von UMAI geholfen werden die Theorie des deklarativen Gedächtnisses in ACT-R interaktiv zu verstehen. Wie in dem in Abschnitt 7 thematisierten Szenario gezeigt wurde, kann das resultierende System Tutoren in der Verbesserung ihrer Lehre unterstützen. Hierbei kann dem Tutor aufgezeigt werden, dass die für ein System entworfenen Lernszenarien bestimmte Inhalte nicht ausreichend intensivieren.

Des Weiteren wurde gezeigt, dass xAPI in webbasierten heterogenen verteilten Systemen als zentrales Kommunikationsinstrument dienen kann. Dies gilt auch für die Entwicklung komplexer ITS-Systeme, die von der auf xAPI aufbauenden statistischen Lernanalyse profitieren können.

Die Orchestrierung der verschiedenen Systeme UMAI, EXTRA und Learning Locker zeigt ein Grundgerüst eines webbasierten ITS auf Grundlage von aktuellen Interoperabilitätsstandards auf. Somit kann das System als Ausgangspunkt der Entwicklung eines webbasierten ITS dienen und damit den Entwurfsprozess und die Einarbeitung für Leihen auf dem Gebiet beschleunigen.

Der Einsatz künstlicher Intelligenz in adaptiven Lernsystemen, die Personalisierung auf den Benutzer ermöglichen, teilt sich in zwei Varianten auf, welche die zu Grunde liegende Domäne zum Einen mit Constraints modellieren und zum Anderen mithilfe kognitiver Architekturen.

Literaturverzeichnis

[And91] ANDERSON, John R. und SCHOOLER, Leal J.: Reflections of the environment in memory (1991)

[And00] ANDERSON, John R und GLUCK, Kevin A: What role do cognitive architectures play in intelligent tutoring systems? (2000)

[And06] ANDERSON, John; CHRISTIAN, Lebiere und TAATGEN, Niels: Modeling Paradigms in ACT-R, in: *Cognition and Multi-Agent Interaction* (2006), S. 29–52

[Ant03] ANTONIJA, Mitrovic; KENNETH, R. Koedinger und BRENT, Martin: A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling, Techn. Ber. (2003), URL https://link.springer.com/content/pdf/10.1007%2F3-540-44963-9.pdf

[Ass15] ASSELMAN, Amal; AAMMOU, Souhaib und NASSEH, Az-Eddine: Comparative study of cognitive architectures. *International Research Journal of Computer Science (IRJCS) Issue* (2015), Bd. 9(2)

[Bes02] BEST, Bradley J.; LEBIERE K., Christian und SCARPINATTO, Christine: Modeling synthetic opponents in MOUT training simulations using the ACT-R cognitive architecture, Techn. Ber. (2002), URL https://www.researchgate.net/publication/228910160

[Bev09] BEVERLY PARK WOLF: *Building intelligent interactive tutors* (2009)

[Bot04] BOTHELL, Dan: ACT-R 6.0 Reference Manual (2004)

[Bra14] BRASOVEANU, Adrian: Intro to Python ACT-R LaLoCo Lab (2014), URL https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxweXRob25hY3RyfGd4OjZhZDc0NjM2MTBhYzgyOTU

[Bra15] BRASOVEANU, Adrian: Introduction to (Python) ACT-R, Techn. Ber. (2015), URL https://sites.google.com/site/pythonactr/,

[Car] CARPENTER, Gail A und GROSSBERG, Stephen: ADAPTIVE RESONANCE THEORY. URL http://techlab.bu.edu/members/gail/articles/141_HBTNN2e_ART_2003_.pdf

[Car08] CARNIEGE LEARNING: PAT Linear Algebra Tutor (2008)

[Cho07] CHONG, Hui-Qing; TAN, Ah-Hwee; NG, Gee-Wah; CHONG, H.-Q; TAN, A.-H und NG, G.-W: Integrated cognitive architectures: a survey. *Artif Intell Rev* (2007), Bd. 28: S. 103–130, URL https://link.springer.com/content/pdf/10.1007%2Fs10462-009-9094-9.pdf

[Cor95] CORBETR, Albert T und ANDERSON, John R: Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge. *User Modeling and User-Adapted Interaction* (1995), Bd. 4(253): S. 253–278, URL https://link.springer.com/content/pdf/10.1007%2FBF01099821.pdf

[Cor97] CORBETT, Albert T; KOEDINGER, Kenneth R und ANDERSON, John R: Intelligent Tutoring Systems (1997), URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.5216&rep=rep1&type=pdf

[Fod88] FODOR, Jerry A; PYLYSHYN, Zenon W; NOAM CHOMSKY, Professors; DEMOPOULOS, William; GLEITMAN, Lila; GREINER, Russ; HORNSTEIN, Norbert; HUMPHREY, Keith; PENTLAND, Sandy; PINKER, Steven; ROSENTHAL, David und STABLER, Edward: Connectionism and Cognitive Architecture Connectionism and Cognitive Architecture: 1 A Critical Analysis, Techn. Ber. (1988), URL https://pdfs.semanticscholar.org/d806/76034bfabfea59f35698af0f715a555fcf50.pdf

[Fra07] FRANKLIN, Stan; D 'MELLO, Sidney; DATLA, Vivek; RAMAMURTHY, Uma; D 'MELLO, Sidney K; MCCAULEY, Lee; NEGATU, Aregahegn und SILVA, Rodrigo: LIDA: A computational model of global workspace theory (2007)

[Fra11] FRANKLIN, Stan; SNAIDER, Javier und MCCALL, Ryan: The LIDA Framework as a General Tool for AGI (2011), URL https://www.researchgate.net/publication/221328964

[Gun16] GUNDERMANN, Alexander; ADVISOR, Main; BEYERER, Jürgen und STREICHER, Co-Advisor Alexander: A Web-Based Serious Game for Joint Training (2016), URL https://www.iosb.fraunhofer.de/servlet/is/63927/Thesis_Gundermann_2016.pdf?command=downloadContent&filename=Thesis_Gundermann_2016.pdf

[Hau] HAUPT, Florian; LEYMANN, Frank; SCHERER, Anton und VUKOJEVIC-HAUPT, Karolina: A Framework for the Structural Analysis of REST APIs. *ICSA*, Bd. 2017, URL http://www.iaas.uni-stuttgart.de/RUS-data/INPROC-2017-08%20-%20A%20Framework%20for%20the%20Structural%20Analysis%20of%20REST%20APIs.pdf

[Hau14] HAUSTANT, Axel: Flask-RESTPlus 0.11.0 documentation (2014), URL http://flask-restplus.readthedocs.io/en/stable/

[JA13] JYOTHI AHUJA, Neelu; SILLE, Roohi und PROFESSOR, Assistant: A Critical Review of Development of Intelligent Tutoring Systems: Retrospect, Present and Prospect, Techn. Ber. (2013), URL www.IJCSI.org

[Kot18] KOTSERUBA, Iuliia und TSOTSOS, John K: A Review of 40 Years in Cognitive Architecture Research Core Cognitive Abilities and Practical Applications (2018)

[Kul16] KULIK, James A und FLETCHER, J D: Effectiveness of Intelligent Tutoring Systems: A Meta-Analytic Review. *Review of Educational Research* (2016), Bd. 86(1): S. 42–78, URL http://journals.sagepub.com/doi/pdf/10.3102/0034654315581420

[Lai87] LAIRD, John E; NEWELL, Allen; ROSENBLOOM, Paul S; NILSSON, Nils; BOBROW, Daniel G; LAIRD, J E und AL, Et: SOAR: An Architecture for General Intelligence*. *Arttfictal Intelhgence* (1987), Bd. 33: S. 1–64, URL https:

//ac.els-cdn.com/0004370287900506/1-s2.0-0004370287900506-main.pdf?_tid=dace06bf-9504-48d5-b306-1fc8f2b8dd31&acdnat=1529673260_61056ad7a1fe3121b171fa5ad16c6632

[Lan91] LANGLEY, Pat; MCKUSICK, Kathleen B; ALLEN, John A; IBA, Wayne F und THOMPSON, Kevin: A Design for the Icarus Architecture Goals of the Research, Techn. Ber. (1991), URL http://www.isle.org/~langley/papers/icarus.sss91.pdf

[Lan08] LANGLEY, Pat; LAIRD, John E und ROGERS, Seth: Cognitive architectures: Research issues and challenges. *Cognitive Systems Research* (2008), Bd. 10: S. 141–160, URL https://ac.els-cdn.com/S1389041708000557/1-s2.0-S1389041708000557-main.pdf?_tid=1eec78f2-e3cb-4e19-8d5a-e8f909e1edc3&acdnat=1529672989_2e6484707368e3e86f4ed768d2c79c7d

[Lew99] LEWIS, R.L.: Cognitive modeling, symbolic, Techn. Ber. (1999), URL https://vision.unipv.it/IA/mitecs-article.pdf

[Ma14] MA, Wenting; ADESOPE, Olusola O; NESBIT, John C und LIU, Qing: Intelligent Tutoring Systems and Learning Outcomes: A Meta-Analysis (2014), URL http://dx.doi.org/10.1037/a0037123.supp

[Mil07] MILLS, Chris; AUSTRALIA, Fujitsu und DALGARNO, Barney: A conceptual model for game based intelligent tutoring systems, Techn. Ber. (2007), URL http://www.ascilite.org/conferences/singapore07/procs/mills.pdf

[Mit01] MITROVIC, Antonija; MAYO, Michael; SURAWEERA, Pramuditha und MARTIN, Brent: Constraint-Based Tutors: a Success Story (2001), URL https://ir.canterbury.ac.nz/bitstream/handle/10092/334/42962_cbmtut.pdf?sequence=1

[New76] NEWELL, Allen und SIMON, Herbert A.: Computer science as empirical inquiry: symbols and search. *Communications of the ACM* (1976)

[Nwa90] NWANA, Hyacinth S: Intelligent Tutoring Systems: an overview. *Artificial Intelligence Review* (1990), Bd. 4: S. 251–277, URL https://link.springer.com/content/pdf/10.1007%2FBF00168958.pdf

[Ohl94] OHLSSON, Stellan: Constraint-Based Student Modeling, in: *Student Modelling: The Key to Individualized Knowledge-Based Instruction*, Springer Berlin Heidelberg, Berlin, Heidelberg (1994), S. 167–189, URL http://link.springer.com/10.1007/978-3-662-03037-0_7

[Ohl96] OHLSSON, Stellan: Learning from performance errors. *Psychological Review* (1996), Bd. 103(2): S. 241–262, URL http://doi.apa.org/getdoi.cfm?doi=10.1037/0033-295X.103.2.241

[O'R00] O'REILLY, Randall C. und MUNAKATA, Yuko.: *Computational explorations in cognitive neuroscience : understanding the mind by simulating the brain*, MIT Press (2000), URL https://dl.acm.org/citation.cfm?id=557205

[O'R12] O'REILLY, Randall C.; WYATTE, Dean R. und ROHRLICH, John: The Leabra Cognitive Architecture: How to Play 20 Principles with Nature and Win! (2012)

[Pol17] POLYMER PROJECT: Polymer Documentation (2017), URL https://www.polymer-project.org/2.0/docs/devguide/feature-overview

[RO90] REITMAN OLSON, Judith; OLSON, M und OLSON, Gary M: The Growth of Cognitive Modeling in Human-Computer Interaction Since GOMS, Techn. Ber. (1990), URL http://www.realtechsupport.org/UB/I2C/CogniveModellingHCI_1990.pdf

[Rob09] ROBINSON, Richard: Exploring the 'Global Workspace' of Consciousness. *PLoS Biology* (2009), Bd. 7(3), URL http://journals.plos.org/plosbiology/article/file?id=10.1371/journal.pbio.1000066&type=printable

[Rub96] RUBIN, David C; WENZEL, Amy E; ANDERSON, John; BONEAU, Alan; CERELLA, John; CROVITZ, Herb; HINTON, Sean; MACHADO, Armando; MURDOCK, Bennet; SERRA, Matt; SCHIFFMAN, Har-Old; STADDON, John; WICKELGREN, Wayne und WIXTED, John: Psychological Review One Hundred Years of ForgettingÄ Quantitative Description of Retention, Techn. Ber. 4 (1996)

[Rus18] RUSTICI SOFTWARE: Experience API Specification (2018), URL https://xapi.com/overview/

[SA05] ST AMANT, Robert; FREED, Andrew R und RITTER, Frank E: Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research* (2005), Bd. 6: S. 71–88, URL http://acs.ist.psu.edu/papers/stamantFR05.pdf

[Sal03] SALVUCCI, Dario D und LEE, Frank J: Simple Cognitive Modeling in a Complex Cognitive Architecture, Techn. Ber. (2003), URL https://www.cs.drexel.edu/~salvucci/publications/Salvucci-CHI03.pdf

[Sho76] SHORTLIFFE, Edward Hance.: *Computer-based medical consultations, MYCIN*, Elsevier (1976), URL https://books.google.de/books?hl=en&lr=&id=i9QXugPQw6oC&oi=fnd&pg=PP1&dq=mycin+stanford&ots=I94mX7aBNI&sig=4PChL17eCakYQWm-T-AxcGI_F2Q#v=onepage&q=mycin%20stanford&f=false

[Sma18] SMARTBEAR SOFTWARE: Swagger Documentation (2018), URL https://swagger.io/docs/specification/about/

[Sow92] SOWA, John F: Semantic Networks, Techn. Ber. (1992), URL http://www.jfsowa.com/pubs/semnet.pdf

[Ste06] STEWART, Terrence C und WEST, Robert L: Deconstructing ACT-R, Techn. Ber. (2006)

[Str16] STREICHER, Alexander; SZENTES, Daniel und GUNDERMANN, Alexander: Game-Based Training for Complex Multi-Institutional Exercises of Joint Forces, Techn. Ber. (2016), URL https://pdfs.semanticscholar.org/3172/743094167b7aecdd04c79eba01cd56b1c116.pdf

[Str17] STREICHER, Alexander und ROLLER, Wolfgang: Interoperable adaptivity and learning analytics for serious games in image interpretation (2017), in: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*

[Sun00] SUN, Ron: Artificial Intelligence: Connectionist and Symbolic Approaches (2000)

[Sun01] SUN, Ron; MERRILL, Edward und PETERSON, Todd: From implicit skills to explicit knowledge: A bottom-up model of skill learning (2001)

[Sur02] SURAWEERA, Pramuditha und MITROVIC, Antonija: KERMIT: a Constraint-based Tutor for Database Modeling, Techn. Ber. (2002), URL https://ir.canterbury.ac.nz/bitstream/handle/10092/317/12584594_kermit.pdf?sequence=1

[Sze12] SZENTES, Daniel; BARGEL, Bela-Andreas und STREICHER, Alexander: ENHANCED VALUE BENEFIT ANALYSIS OF GAME FRAMEWORKS AS A TOOL FOR DIGITAL SERIOUS GAME DEVELOPMENT (2012), URL https://www.iosb.fraunhofer.de/servlet/is/63300/ICERI2012_ValueBenefitAnalysisSGFrameworks.pdf?command=downloadContent&filename=ICERI2012_ValueBenefitAnalysisSGFrameworks.pdf

[Taa04] TAATGEN, Niels A; VAN RIJN, Hedderik und RIJN@RUG, D H Van: ACT-R Tutorial. *Psychological Review* (2004), Bd. 111: S. 1036–29, URL http://act-r.psy.cmu.edu/wordpress/wp-content/themes/ACT-R/tutorials/ACT-R_intro_tutorial.ppt

[Tho15] THOMSON, Robert; LEBIERE, Christian; ANDERSON, John R und STASZEWSKI, James: A general instance-based learning framework for studying intuitive decision-making in a cognitive architecture. *Journal of Applied Research in Memory and Cognition* (2015), Bd. 4: S. 180–190, URL https://ac.els-cdn.com/S2211368114000539/1-s2.0-S2211368114000539-main.pdf?_tid=a0f0316e-e9a1-441c-be37-7f6c469d93bf&acdnat=1530716641_692916574fe4529cd1d7378ebf88df36

[Var15] VARANASI, Balaji und BELIDA, Sudha: Documenting REST Services, in: *Spring REST*, Apress, Berkeley, CA (2015), S. 91–104, URL http://link.springer.com/10.1007/978-1-4842-0823-6_6

[Vid15] VIDAL, Juan C.; RABELO, Thomas und LAMA, Manuel: Semantic description of the experience API specification (2015), in: *Proceedings - IEEE 15th International Conference on Advanced Learning Technologies: Advanced Technologies for Supporting Open Access to Formal and Informal Learning, ICALT 2015*

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungen