Ansatz zur Definition einer Mustersprache für Lehr-/Lernsysteme

Andreas Harrer[+], Alke Martens[*]
[+]Institut für Informatik und interaktive Systeme, Universität Duisburg-Essen
[*]Institut für Informatik, Universität Rostock
harrer@collide.info, martens@informatik.uni-rostock.de

Abstract: Trotz der verhältnismäßig langen Geschichte, auf die e-Learning zurückblicken kann, ist die Verwendung von Softwaretechniken zur Entwicklung dieser Programme noch lange nicht durchgesetzt. Es gibt verschiedene Ansätze, Methoden der Softwareentwicklung in den Bereich des e-Learning zu etablieren. Die Bandbreite reicht von Beschreibung von Projektentwicklungsmethoden für e-Learning bis hin zu Pattern Mining in bestehenden Lehr-/Lernsystemen. Eine Vereinheitlichung verschiedener Arbeiten, beispielsweise in Form einer Mustersprache oder eines Musterkataloges fehlt bislang. Ein Ansatz in Richtung einer generativen Mustersprache wird in dem vorliegenden Papier beschrieben.

1 Einführung

Unter dem Oberbegriff e-Learning verbergen sich eine Vielzahl von Systemtypen. Die in diesem Papier dargestellten Inhalte beziehen sich hauptsächlich auf den besonderen Typ der intelligenten Lehr-/Lernsysteme (ILLS). Im Gegensatz zu anderen e-Learning Systemen besitzen intelligente Lehr-/Lernsysteme ein komplexes Wissensmodell der Anwendungsdomäne, das Expertenwissensmodell. Dieses Expertenwissen, das in herkömmlichen Systemen in Form eines Expertensystems in das Lehr-/Lernsystem integriert war [Cla86], stellt die eigentliche „Intelligenz“ des Systems dar. Basierend auf den im Expertenwissen gehaltenen Fakten und Regeln ist das ILLS in der Lage, selbständig auf Fehler des Lernenden zu reagieren, diese zu korrigieren, sowie Tipps und Hinweise zu geben. Neben dem Expertenwissen ist ein weiterer Bestandteil eines ILLS das pädagogische Wissensmodell. Abhängig von der Realisierung des Systems ist dieses Modell als Bestandteil des Expertenwissens oder in Form eines extra Moduls realisiert. Es enthält wahlweise didaktisch aufbereitete Lehrinhalte oder didaktische Strategien zur Inhaltsvermittlung. Des weiteren besteht ein klassisches ILLS aus dem Lernermodell und dem Modell der grafischen Benutzungsschnittstelle. Die Bezeichnung als Modell kennzeichnet bei den Bestandteilen der Architektur zunächst nur die theoretische Beschreibung der jeweiligen Komponente und muss entsprechend der Beschreibungsebene geeignet ersetzt werden.

Die ILLS Architektur, die bereits in den 1980er Jahre von Clancey beschrieben wurde [Cla84], kann bereits im weiteren Sinne als ein Muster zur Entwicklung von ILLS ange-

sehen werden [DH02]. Im Kontrast zu der sehr einheitlichen Verwendung der Benennung der Komponenten steht jedoch die Interpretation der Funktion der Komponenten in der Entwicklung von intelligenten Lehr-/Lernsystemen [Mar03]. Während einige Systementwickler beispielsweise das Expertenwissensmodell noch immer als separates Expertensystem realisieren, wird in anderen Systemen hier auf eine einfache Datenstruktur ohne eigene Ausführung zurückgegriffen [Mar03]. Die Freiheit, die Clanceys Beschreibung dem Systementwickler läßt, wirkt sich negativ in Form von homogenen Benennungen heterogener Funktionsweisen aus. Dies erschwert sowohl die Wiederverwendung bestehender Softwarekomponente, als auch den Vergleich bestehender Systeme.

In den letzten Jahren ließen sich Ansätze beobachten, den Nachteilen der informalen Beschreibung zu begegnen und mehr Einheitlichkeit in die Entwicklung von Lehr-/Lernsystemen zu bringen. Dies sind einerseits die Entwicklung verschiedener Standards und andererseits die Etablierung von Methoden der Softwareentwicklung. Aus dem Bereich der Standards liegen Ansätze für unterschiedliche Ebenen der Systementwicklung vor. Die Bandbreite reicht von Architekturstandards (z.B. Learning Technology System Architecture LTSA [FT01]), über Arbeiten zur Projektbeschreibung (z.B. Essener Lern Modell [Paw00]), bis hin zu XML Beschreibungen für die Lehrmaterialien (z.B. Dublin Core Metadata Initiative [DCM02]). Standards haben den Vorteil der einheitlichen Verwendung und Benennung von Bestandteilen des Lernsystems und unterstützen damit die Austauschbarkeit auf der Ebene, auf der sie diese Bestandteile beschreiben. Sie haben allerdings oft den Nachteil, dass sie für die konkrete Softwareentwicklung zu unspezifisch sind. Die Verwendung von Standards verhindert die uneinheitliche Interpretation der Arbeitsweise von Komponenten nicht unbedingt. Beispielsweise wurde die LTSA ausdrücklich mit dem Ziel entwickelt, auf viele unterschiedliche Lernsysteme anwendbar zu sein. Dieses Ziel wurde im wesentlichen erreicht - auf Kosten eines Mangels an Detail für die konkreten Systemtypen. Um diesen Nachteil auszugleichen, wird an Erweiterungen der LTSA gearbeitet.

Die Verwendung von Methoden der Softwareentwicklung im Bereich der Lehr-/Lernsysteme basiert auf der Idee, dass es sich bei der Entwicklung von Software um eine ingenieurmäßige Tätigkeit handelt - und nicht, wie vielfach praktiziert, um eine „Kunst“. Diese Sichtweise impliziert, dass es sich bei der Entwicklung von Software um ein Anwenden von Techniken handelt, die klare und eindeutige Nachvollziehbarkeit verschiedener Aspekte des Entwicklungsprozesses und des Ergebnisses, also der Software, erfordert.

In dem vorliegenden Papier verfolgen wir den Ansatz, ausgehend von einem zentralen Architekturmuster und dessen Ausbau, eine generative Mustersprache für ILLS vorzubereiten. Im nachfolgenden Kapitel wird anhand einer Vorstellung der Methoden des Software Engineering im e-Learning eine Abgrenzung zu anderen Arbeiten vorgenommen. Das nächste Kapitel dient der Schaffung einer gemeinsamen Terminologie für die Beschreibung der generativen Mustersprache. Ein Ausschnitt der generativen Mustersprache selbst wird im Anschluss daran beschrieben. Das Papier endet mit einer Zusammenfassung des Ansatzes und einer Diskussion.

2 Methoden des Software Engineering im e-Learning

Betrachtet man die Vorgehensweisen bei Entwurf und Implementierung von Lehr-/Lernsystemen genauer, so kann folgendes festgestellt werden: Einfache lernunterstützende Systeme, wie Web-Portale [Pos05] und strukturierte Diskussionsforen (z.B. FLE3 [MHL99]), sind häufig nach softwaretechnischen Grundsätzen wie Modularität und Erweiterbarkeit gestaltet sind, was sich durch funktionale Erweiterungen durch andere Entwickler ausdrückt [PHLV05, DCM03]. Dahingegen werden komplexe Lehr-/Lernsysteme, wie ILLS oder andere Systeme, die auf Anwendungen künstlicher Intelligenz für die Lehre basieren (AIED-Systeme), häufig entsprechend der Fachdomäne oder orientiert an einem kognitiven bzw. pädagogischen Ansatz konzipiert, ohne softwaretechnischen Anforderungen Rechnung zu tragen. Die Offenlegung von Architekturdetails oder Systemschnittstellen, sowie Erweiterbarkeit der Systeme ist hier kaum ein Thema. Zum Teil werden die Teilbereiche sogar als konkurrierend betrachtet, was sich in Aussagen wie „AI kept fighting and losing against Software Engineering“ [Roh04] widerspiegelt.

Da die vollständige Entwicklung von Lehr-/Lernsystemen aufgrund der Anforderungen im programmiertechnischen Bereich, aber auch was den pädagogischen und kognitionswissenschaftlichen Hintergrund angeht, sehr aufwendig ist, sehen wir es als notwendig an, auch Lehr-/Lernsysteme als Softwaresysteme zu betrachten, die mit Software Engineering Vorgehensweisen konzipiert und realisiert werden sollten. Dies bringt eine Verringerung der Komplexität bei der Erstellung der Teilkomponenten eines Softwaresystems mit sich. Insbesondere ist es bei geeignetem Design der Systeme möglich, Fachgebietsexperten (Didaktiker, Fachexperten) einzubinden, ohne dass diese Programmierkenntnisse entwicklen müssen. Durch Trennung der fachbegründeten Spezifikation (beispielsweise von Lehrprozessen und Domänenwissen) von programmiertechnischen Aspekten kann dies erreicht werden. Als softwaretechnische Prinzipien, die die Entwicklung von Lehr-/Lernsystemen unterstützen können, sehen wir folgende:

Referenzarchitekturen und Architekturmuster dienen dazu, die grundsätzliche Struktur und Beziehungen der wesentlichen Komponenten eines Softwaresystems festzulegen. Im Bereich der Lehr-/Lernsysteme gibt es einige Vorschläge für Referenzarchitekturen, wie die LTSA-Referenzarchitektur (IEEE Learning Technology Systems Architecture) [FT01] oder die studentenmodell-zentrierte Architektur [Bru95]. Architekturmuster, die die Struktur von Softwaresystemen auf abstraktem Niveau beschreiben [BMR+96], wurden im Bereich der ILLS erstmals in [Dev01] behandelt und dienten dort zur Stukturierung von Tutorsystemen nach dem Schichtenprinzip (z.B. in [IMSW01]).
Frameworks definieren bereits auf konkreter Programmebene die Struktur eines Softwaresystems. Einige Teile eines Frameworks können direkt und in verschiedenen Systemen verwendet werden, andere Teile sind den spezifischen Gegebenheiten des konkreten Systems anzupassen. Frameworks, die im objektorientierten Programmierstil definiert sind, werden durch Instantiierung von bereits vorhandenen Bausteinen, durch Implementierung von vordefinierten Schnittstellen oder Spezialisierung von Klassen benutzt. FITS [IM94] und das Framework für offene verteilte Lernumgebungen [MTH98] sind Beispiele für die Frameworkidee in Lehr-/Lernsystemen.

Entwurfsmuster beschreiben typische Lösungen für wiederkehrende Entwurfsprobleme im kleineren Maßstab von Systemen [GHJV95]. Sie helfen dabei, die System-Grundstrukturen zu verfeinern und mit eleganten Lösungen auszuformen. Entwurfsmuster wurden in [Dev99] in den Bereich der ILLS eingeführt.
Prozessmuster und Learning Design sind ein Ansatz, prozessorientierte Aspekte des Lernens explizit darzustellen und wiederverwendbar zu machen. Durch diese Explizierung werden die häufig nur implizit vorhandenen Prinzipien der Didaktik oder des Arrangements von kollaborativen Szenarien offengelegt und damit auch für die jeweiligen Experten nachvollziehbar. Weiterhin sind diese Beschreibungen im Allgemeinen auch für andere Systeme nutzbar. Erste Ansätze dazu fanden sich im Bereich der tutoriellen Lehrsysteme in Form von austauschbaren Katalogen von Tutorregeln [Har97], die die Reaktionen des Computertutors auf Benutzereingaben definieren. Insbesondere in den Bereichen des sog. Learning Designs [Con03] zur Beschreibung von Phasierung und Werkzeugunterstützung von computerbasierten Lernszenarien und im Bereich des kollaborativen Lernens durch sog. Kollaborationsskripte [Dil02] wird diese Thematik seit kurzem intensiv verfolgt.
Komponentenbasierter Entwurf bringt die Definition von Bausteinen mit festgelegten Schnittstellen und Austauschformaten mit sich, mit der Zielstellung, verschiedene Komponenten flexibel miteinander kombinieren zu können. Neben einigen speziellen Lernanwendungen, z.B. zur graph-basierten Modellierung mit verschiedensten austauschbaren und erweiterbaren visuellen Sprachen [Pin03], stellt die SAIL-Initiative (Scalable Architecture for Interactive Learning) [SAI05] einen Ansatz dar, eine Vielzahl von Lernkomponenten verschiedener Systeme, wie Simulationskomponenten, Concept-Mapping Werkzeuge, Unterstützungswerkeuge zur Hypothesenbildung oder Annotationswerkzeuge, flexibel in eine Laufzeitumgebung einzubinden und damit die Stärken verschiedener Lehr-/Lernsysteme miteinander zu verbinden.
Ontologien bestimmen wesentliche Konzepte einer Domäne und deren Beziehungen miteinander. Bezogen auf Softwaresysteme können sie dazu eingesetzt werden, das Vokabular festzulegen, das Komponenten zur Kommunikation untereinander verwenden können. Da dieses repräsentierte Wissen für die Wiederverwendung und Interoperabilität der Komponenten notwendig ist, findet dieses Thema auch im Bereich der Lehr-/Lernsysteme seit einigen Jahren seine Berücksichtigung [MIS97].
Refactoring stellt eine Technik zur Weiterentwicklung existierender Softwaresysteme dar, die es ermöglicht die Systemstruktur zu verbessern, z.B. modularer, flexibler und erweiterbar zu machen, ohne die Systemfunktionalität zu berühren [FBBR99]. Insbesondere in Kombination mit dem komponentenbasierten Ansatz ist diese Technik einsetzbar, um die bestehenden Schnittstellen eines Systems so zu transformieren, dass sie den Standards der Komponentenarchitektur entsprechen und somit in das System integriert werden können. Diese Technik wurde im Bereich von Lehr-/Lernsystemen in [Har03] diskutiert und wird im Rahmen von OpenSource Lernsystemen (wie der oben erwähnten SAIL-Initiative) sicherlich an Bedeutung gewinnen, da dort die Modifikationen von Komponenten und Schnittstellen wesentlich besser sichtbar sind, als in nicht offengelegten Forschungsprototypen oder kommerziellen Lehrsystemen.

Wir beschränken uns in diesem Artikel auf die konzeptuelle Integration einiger der genannten Ansätze. Dazu wählen wir uns die verschiedenen muster-orientierten Ansätze, wie

Architekturen, Entwurfsmuster und Prozessmuster aus und diskutieren im folgenden die Integration dieser Ansätze im Rahmen einer sog. Mustersprache für Lehr-/Lernsysteme. Diese soll als Orientierungshilfe bei der Entwurf und Entwicklung von Lehr-/Lernsystemen dienen.

3 Mustersprache - Begriffsklärung

Die Kenntnis einzelner Muster ist zur Lösung typischer Probleme hilfreich. Allerdings ergibt sich durch die Verknüpfung und kombinierte Anwendung verschiedener Muster der Mehrwert, ganze System und deren Subsysteme miteinander abgestimmt entwerfen zu können. Dazu ist es erforderlich, Beziehungen zwischen den Mustern herzustellen und diese differenziert zu betrachten. Typische Fragen dabei sind zum Beispiel:
Welche Muster können verwendet werden, um ein anderes Muster zu realisieren?
Welche Muster ergänzen sich und sind zusammen einsetzbar?
Welche Muster können als Alternativen betrachtet werden?
Welche Gemeinsamkeiten bzw. Unterschiede bestehen mit anderen Mustern?
Eine Sammlung von Mustern, in der solche Beziehungen explizit repräsentiert werden, wird als Mustersprache bezeichnet [SSRB00]. Eine muster-orientierte Software Entwicklung kann mit Hilfe einer Mustersprache wie folgt realisiert werden: Basierend auf den Anforderungen an ein System wird ein sog. *Einstiegsmuster* gewählt, das im Normalfall ein Architekturmuster sein wird. Aufgrund der noch nicht realisierten Anforderungen (und den Gegebenheiten des Einstiegsmusters) wird mit Hilfe der Mustersprache ein *Fortsetzungsmuster* gesucht, das einen Teil der Anforderungen realisiert. Damit wird das System schrittweise verfeinert. Dieser Schritt wird solange wiederholt, bis alle Anforderungen erfüllt sind oder kein Fortsetzungsmuster mehr vorliegt.

Eine Mustersprache, die nach diesem Vorgehen für jede Anwendung ein vollständiges System erbringt, wird als *generative Mustersprache* oder als *vollständig* bezeichnet. Mustersprachen dieser Art sind normalerweise jedoch nur für sehr eingeschränkte Anwendungsklassen formulierbar. Für den Bereich der ILLS sind vereinzelte Ansätze für Mustersprachen vorhanden, wie z.B. eine Mustersprache für Tutorsysteme, die nach dem Schichtenprinzip aufgebaut sind [Dev01] oder eine Mustersprache für blended learning Prozessmuster [DMP04]. Allerdings betrachten diese Sprachen jeweils nur einen bestimmten Aspekt von Mustern. Wir wollen in dieser Arbeit einen integrativen Ansatz vorstellen, der Architekturmuster, Entwurfsmuster und Prozessmuster in Beziehung setzt, wobei insbesondere verschiedene Granularitätsstufen berücksichtigt werden sollen. Beispielsweise sollen Mikro-Lehrprozesse, wie sie bei tutoriellen Systemen üblich sind, mit Makro-Lehrprozessen im Sinne des IMS Learning Design [Con03] in Verbindung gesetzt werden.

4 Ansatz zur Definition einer Mustersprache für Lehr-/Lernsysteme

In diesem Abschnitt wird unser Ansatz zur Definition einer Mustersprache für ILLS vorgestellt. Die Überlegungen zur Entwicklung eines systematischen, auf Software Engineering basierenden Ansatzes zur Beschreibung von ILLS stützt sich zunächst als Einstiegsmuster auf die klassische Architektur intelligenter Lehr-/Lernsysteme. Diese Architektur wurde in verschiedenen Quellen beschrieben [Cla86, Mar03]. Die Bestandteile der Architektur werden unterschiedlich bezeichnet und wahlweise als Modell, als Modul oder als Komponente charakterisiert (zur Diskussion dieser Bezeichnungen siehe beispielsweise [Mar03]). In der vorliegenden Darstellung wurden die Bestandteiles eines ILLS als Module bezeichnet. Hiermit wird die Interpretation nahelegt, dass es sich nicht um ein ausformuliertes oder klar beschriebenes Modell handelt, sondern um eine aus mehreren Elementen zusammengesetzte Einheit innerhalb eines Gesamtsystems. Entsprechend besteht ein ILLS aus den in Abbildung 1 aufgezeigten Modulen, die als Fortsetzungsmuster betrachtet werden können: dem Expertenmodul, dem Lernermodul, dem User Interface Modul und dem Prozesssteuerungsmodul (oder Tutormodul). Die Grafik verwendet Aggregationsangaben, aber weist keine Notierung über die Menge der verbundenen Module auf. Annahme ist zunächst, dass es von jedem der beteiligten Module genau eins in einem ILLS gibt. Entsprechend muss eine weitere Aufteilung erfolgen - beispielsweise kann das Expertenwissensmodul als zwei getrennte Modelle (Expertenwissensmodell und pädagogisches Wissensmodell) realisiert werden.

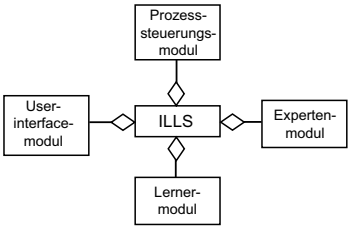

Abbildung 1: Module eines ILLS

In der nächsten Ausbaustufe, zu sehen in Abbildung 2, wird entsprechend dieser Überlegung eine Erweiterung vom Expertenmodul und vom Lernermodul beschrieben. Diese Darstellung ist nicht erschöpfend, sondern lediglich eine erste Skizze, um den Zusammenhang zu verdeutlichen. Vorgeschlagen wird über die Spezialisierungsbeziehung die Umsetzung des Expertenmoduls als pädagogischer Agent [JRL00], als fallbasiertes Expertenmodul [Mar03] oder als Expertenmodul basierend auf Produktionsregeln [ABCL90]. Das Lernermodell wird ebenfalls spezialisiert. Hier sind beispielsweise die folgenden Spezialisierungen denkbar: CSCL (Computer Supported Collaborative Learning) Lernermodul [Har00]; ein Kognitiver Tutor, der in der Lage ist, mittels Modeltracing eine Bewertung des Lerners vorzunehmen [ABCL90]; das einfache Lernerprofil, das keine weitere Information zur Korrektur und Bewertung des Lerners enthält, aber trotzdem bei der Adaption des Lehrszenarios Verwendung findet [Mar03]; die klassische Fehlerbibliothek [BB78],

die Methoden zum Vergleich der durch einen Lerner gemachten Fehler mit bekannten Fehlern im Anwendungsgebiet bereitstellt; das Overlay Modul [Gol77], das über einen Vergleich des Lernerwissens mit dem Expertenwissen des Anwendungsbereiches Aussagen über den Wissensstand des Lerners erlaubt.

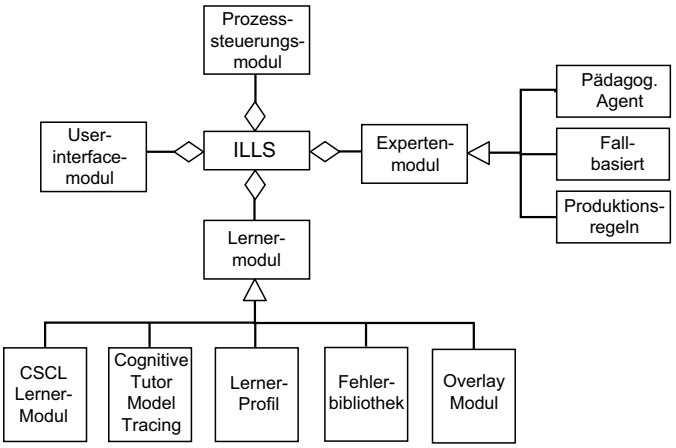

Abbildung 2: Erweiterung der Module eines ILLS

Im Gegensatz zu den Spezialisierungsbeziehungen, die beim Expertenmodul und beim Lernermodul vorstellbar sind, wird hinsichtlich des Ausbaus des Prozesssteuerungsmoduls eine andere Perspektive eingenommen: Konzeptionell kann das Prozesssteuerungsmodul als aus einem Engine-Teil und einem Spezifikationsteil zusammengesetzt interpretiert werden. Dies hat den Vorteil, dass die Spezifikation des Prozesses von der für alle Spezifikationen gleichartigen Ausführungssemantik abgetrennt werden kann. Im Engine-Teil wird die Prozesssteuerungskomponente, also die Semantik, beschrieben. Hier findet die tatsächliche Realisierung und Ausführung der Prozesssteuerung statt. Die Prozesssteuerungskomponente kann beispielsweise in Form einer Learning Design (LD) Engine (wie beispielsweise Coppercore, siehe www.coppercore.org), eines Intelligent Distributed Learning Environment (IDLE) Agenten [Har00] oder auch eines Tutoring Prozess Modell (TPM) [Mar03] Interpreters spezialisiert werden. Der Spezifikationsanteil der Prozesssteuerungskomponente wird durch Prozessbeschreibungen repräsentiert. Hier findet die formale Spezifikation der durchzuführenden Lehr-/Lernprozesse statt: So beschreiben LD-Dokumente [Con03] die Lernszenarien mitsamt Aktivitäten, Diensten und Lernresourcen, die in einer LD-Engine umgesetzt werden; dem IDLE Agenten ist die entsprechende Verhaltensbeschreibung der Agentenrolle in der Lerngemeinschaft zugeordnet; dem TPM Interpreter ist ein Tutoring Prozess Modell (TPM) assoziiert. Das Tutoring Prozess Modell kann in weiteren Spezialisierungen vorliegen, beispielsweise als Timed TPM mit Berücksichtigung von temporalen Aspekten, oder als adaptives TPM (vergl. [Mar03]). Diese Struktur des Prozesssteuerungsmoduls wird in Abbildung 3 dargestellt. Damit können vorliegende Mustersammlungen, z.B. Bibliotheken von IMS/LD-Dokumenten oder verschiedene TPM-Beschreibungen, in das System integriert werden,

wobei durch die Trennung von Intepreter und Spezifikation jeweils nur die Spezifikation eingebunden werden muss.

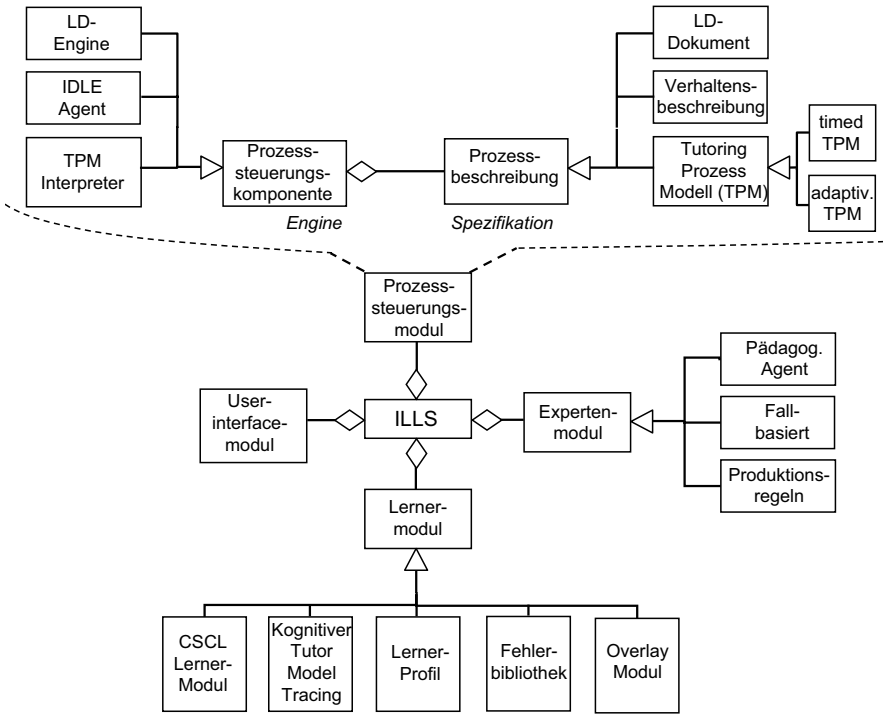

Abbildung 3: Erweiterung des Prozesssteuerungsmoduls - Trennung in Engine und Spezifikation

In der nächsten Ausbaustufe in Richtung einer Mustersprache wird die Hinzunahme von allgemeinen Softwaremustern motiviert. In Abbildung 4 sind die Generellen Softwaremuster, die sich für die Realisierung der User Interface (UI) Komponente anbieten, exemplarisch dargestellt. Beispiele für klassische Muster sind für die Realisierung des UI das Presentation Abstraction Control (PAC), und das Model View Controller (MVC) Pattern (beide in [BMR+96]). PAC verwendet in diesem Zusammenhang z.B. die Pattern Chain of Responsibility und Publish Subscribe (beide in [GHJV95]) zur Kommunikationsregelung zwischen den Agenten. Letzteres wird typischerweise auch in der alternativen MVC-Realisierung angewendet. MVC würde z.B. die Verwendung des Singleton und des Factory Patterns motivieren (beide in [GHJV95]). Das Factory Pattern bietet sich darüber hinaus für die Entwicklung des CSCL Lernermoduls an. Für die Entwicklung einer Prozesssteuerungskomponente bietet sich u.a. das State Pattern [GHJV95] an, um Zustandsabhänigkeiten der Lehrprozessführung zu implementieren.

Diese weitgehend konzeptuelle Darstellung von Mustern und den Zusammenhängen zwischen ihnen (ausgedrückt durch Standardbeziehungen wie Komposition, Spezialisierung und Benutzung) kann durch Festlegung von Schnittstellen, Protokollen und Ontologien für

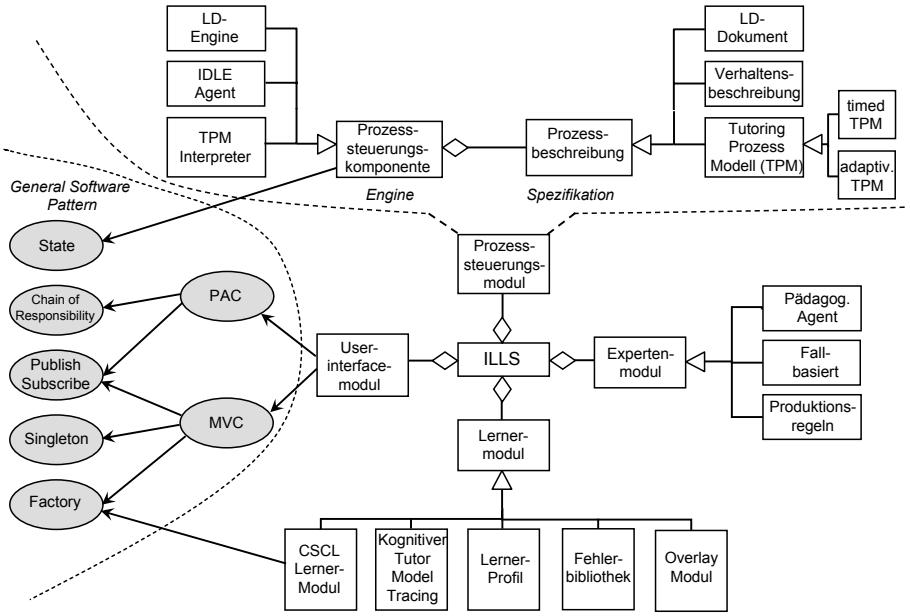

Abbildung 4: Erweiterung um generelle Softwaremuster

das erlaubte Vokabular auch im Hinblick auf technische Integration und Interoperabilität ausgerichtet werden. Durch Definition von Standardschnittstellen, Datentypen und Transformationsvorschriften zwischen den Realisierungen von Mustern, kann ein entscheidender Schritt hin zu einer komponenten-basierten muster-orientierten Entwicklung von Lehr-/Lernsystemen gemacht werden. Dabei sind sowohl Standards für Inhalte (z.B. Dublin Core und LOM) als auch für Komponentendefinitionen (z.B. Properties bei Java Beans) ein Hilfsmittel, um nach den Prinzipien von Mustern entwickelte Software-Komponenten interoperabel zu machen. Sind bereits Realisierungen von Komponenten vorhanden, die von den Schnittstellen her nicht aufeinander abgestimmt sind bzw. Standards folgen, so kann durch Einsatz des Adapter-Entwurfsmusters [GHJV95] Interoperabilität erreicht werden.

5 Zusammenfassung

Verschiedene Untersuchungen von unterschiedlichen Lehr-/Lernsystemen, beispielsweise von intelligenten Lehr-/Lernsystemen, Adaptiven Hypermediasystemen, Lehr-/Lernsystemen mit CSCL, haben gezeigt, dass Techniken und Methoden des Software Engineering meist keine Anwendung finden. Unter anderem darauf ist es nach unserer Ansicht zurückzuführen, dass bislang Probleme in den drei Bereichen Wiederverwendung, Kommunikation und Vergleichbarkeit auftreten. Die Wiederverwendung von Systemen, Sys-

temkomponenten oder Systemmodellen scheint bislang kaum geglückt. Ein Problem bei der Entwicklung von Lehr-/Lernsystemen ist nach wie vor die Kommunikation und Kooperation sehr unterschiedlicher Projektbeteiligter, wie beispielsweise Software Entwickler und Experten des zu lehrenden Anwendungsgebietes. Ein Vergleich zwischen unterschiedlichen Lehr-/Lernsystemen findet meist durch eine Gegenüberstellung der Lernerzufriedenheit oder des Lernerfolgs statt. Angemessen wäre aber auch, insbesondere hinsichtlich unterschiedlicher Anwendungsdomänen der gegenübergestellten Systeme, ein Vergleich verschiedener Realisierungen auf der Systemebene. Dass ein Vergleich, sowohl didaktisch ähnlicher und domänengleicher, als auch heterogener Lehr-/Lernsysteme hinsichtlich (Wieder-)Verwendbarkeit, Performanz oder auch Realisierung vorgenommen wird, ist derzeit kaum möglich.

Ein Weg, der hinsichtlich aller drei Bereiche vielversprechend erscheint, ist die Zusammenführung verschiedener Bestrebungen aus dem Feld der Architekturbeschreibungen (z.B. Referenzarchitekturen, Architekturmuster und Frameworks), aus dem Gebiet des Softwareentwurfs (z.B. Pattern, komponentenbasierter Entwurf und auch Refactoring) und aus dem Feld der Lernprozessbeschreibungen. Gemeinsam können verschiedene dieser Ansätze in eine generative Mustersprache eingehen. Dies führen wir in dem vorliegenden Papier exemplarisch vor. Wir gehen von der klassischen ILLS Architektur, bestehend aus Expertenmodul, Lernermodul, Prozesssteuerungsmodul und User Interface Modul, aus. Diese Architektur verwenden wir als Einstiegsmuster. Aufbauend darauf beschreiben wir beispielhaft ein ILLS Fortsetzungsmuster. Auf diese Weise skizzieren wir die Erweiterung des Expertenmoduls und des Lernermoduls. Interessant ist für uns insbesondere die Verfeinerung des Prozesssteuerungsmoduls, für das wir eine Trennung von Engine und Spezifikation vorschlagen. Auf diese Weise adressieren wir die Möglichkeit zur Wiederverwendung allgemeiner Prozessausführungssemantiken, zur Spezifikation der Prozesse durch den Fachgebietsexperten und zur Transferierbarkeit von Beschreibungen in andere Kontexte. Zum Abschluss zeigen wir die Verküpfung von ILLS-spezifischen mit generellen Softwaremustern auf.

Literatur

[ABCL90] J. R. Anderson, C. F. Boyle, A. T. Corbett und M. W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42:7–49, 1990.

[BB78] J. S. Brown und R. R. Burton. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science*, 2:155–192, 1978.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, PeterSommerlad und Michael Stal. *A System of Patterns*. John Wiley & Sons, Chichester, 1996.

[Bru95] Peter Brusilovsky. Intelligent Learning Environments for Programming: The Case for Integration and Adaption. In J. Greer, Hrsg., *Proc. of the World Conference on Artificial Intelligence in Education AI–ED 95*, Seiten 1–7, Washington, 1995.

[Cla84] W. J. Clancey. Methodology for Building an Intelligent Tutoring System. In W. Kintsch, J. R. Miller und P. G. Polson, Hrsg., *Methods and Tactics in Cognitive Science*, Seiten 51–84. Lawrence Erlbaum Associates, Hillsdale, New Jersey, London, 1984.

[Cla86] W. J. Clancey. From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985. *The AI Magazine*, 7(3):40–61, August 1986.

[Con03] IMS Global Learning Consortium. IMS Learning Design Information Model. Bericht 1.0, IMS global learning consortium, 2003.

[DCM02] DCMI. Dublin Core Metadata Initiative. http://dublincore.org, 2002.

[DCM03] Jan Dolonen, Weiqin Chen und Anders Morch. Integrating Software Agents with FLE3. In Barbara Wasson, Sten Ludvigsen und Ulrich Hoppe, Hrsg., *Designing for Change in Networked Learning Environments - Proc. of CSCL 2003*, Jgg. 2 of *Computer-Supported Collaborative Learning*, Seiten 157–161. Kluwer Academic Publishers, 2003.

[Dev99] Vladan Devedzic. Using Design Patterns in ITS Development. In Susanne P. Lajoie und Martial Vivet, Hrsg., *Proc. of the World Conference on Artificial Intelligence in Education AI–ED 99*, Seiten 657–659, Amsterdam, 1999. IOS Press.

[Dev01] Vladan Devedzic. A Pattern Language for Architectures of Intelligent Tutors. In Johanna D. Moore, Carol Redfield und W. Lewis Johnson, Hrsg., *Proc. of the World Conference on Artificial Intelligence in Education AI–ED 2001*, Seiten 542–544, San Antonio, TX, 2001.

[DH02] Vladan Devedzic und Andreas Harrer. Architecural Patterns in Pedagogical Agents. In S. Cerri, G. Gouardéres und F. Paraguaçu, Hrsg., *Intelligent Tutoring Systems, 6th Internat. Conf.*, Lecture Notes in Computer Science 2363, Seiten 81–90, Biarritz, France and San Sebastian, Spain, June 2002. Springer.

[Dil02] Pierre Dillenbourg. Overscripting CSCL: The risks of blending collaborative learning with instructional design. In Paul Kirschner, Hrsg., *Three worlds of CSCL. Can we support CSCL?* Open Universiteit Nederland, Heerlen, 2002. Inaugural Adress.

[DMP04] Michael Derntl und Renate Motschnig-Pitrik. Patterns for blended, Person-Centered learning: strategy, concepts, experiences, and evaluation. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, Seiten 916–923, New York, NY, USA, 2004. ACM Press.

[FBBR99] Martin Fowler, Kent Beck, John Brant und William Opdykeand Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Reading, MA, 1999.

[FT01] Frank Farance und Joshua Tonkel. LTSA Specification - Learning Technology Systems Architecture, Draft 8. http://www.edutool.com/ltsa, 2001.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[Gol77] I. P. Goldstein. Overlays: A Theory of Modelling for Computer Aided Instruction. Bericht AI Memo 406, MIT, Cambridge, MA, Feb. 1977.

[Har97] Andreas Harrer. Both sides of the coin - blending cognitive and motivational aspects into a tutoring strategy. In T. Ottmann, Z. Halim und Z. Razak, Hrsg., *Proceedings of the International Conference on Computers in Education, ICCE97*, Seiten 188–195, Kuching, Malaysia, 1997.

[Har00] Andreas Harrer. *Unterstützung von Lerngemeinschaften in verteilten intelligenten Lehrsystemen*. Dissertation, Technische Universität München, Institut für Informatik, 2000.

[Har03] Andreas Harrer. Software Engineering Methods for re–use of Components and Design in Educational Systems. *International Journal of Computers & Applications*, Vol. 25, 2003.

[IM94] Mitsuru Ikeda und Riichiro Mizoguchi. FITS, A Framework for ITS – A computational model of tutoring. *Journal of Artificial Intelligence in Education*, 5(3):319–348, 1994.

[IMSW01] Torsten Illmann, Alke Martens, Alexander Seitz und Michael Weber. Structure of Training Cases in Web-Based Case-Oriented Training Systems. In *ICALT 2001 - IEEE International Conference on Advanced Learning Technologies*. IEEE, 2001.

[JRL00] William L. Johnson, Jeff Rickel und J. Lester. Animated Pedagogical Agents: Face-to-Face Interaction in Interactive Learning Environments. *International Journal of Artificial Intelligence in Education*, 11, 2000.

[Mar03] Alke Martens. *Ein Tutoring Prozess Modell fuer fallbasierte Intelligente Tutoring Systeme*. DISKI 281. AKA Verlag infix, 2003.

[MHL99] Hanni Muukkonen, Kai Hakkarainen und Minna Lakkala. Collaborative Technology for Facilitating Progressive Inquiry: Future Learning Environment Tools. In Chris Hoadly und Jeremy Roschelle, Hrsg., *Proc. of CSCL 1999*. Stanford University, 1999.

[MIS97] Riichiro Mizoguchi, Mitsuru Ikeda und Katherine Sinitsa. Roles of Shared Ontology in AI–ED Research. In B. du Boulay und R. Mizoguchi, Hrsg., *Proceedings of the World Conference on Artificial Intelligence in Education AI–ED 97*, Seiten 537–544, Kobe, 1997.

[MTH98] Martin Mühlenbrock, Frank Tewissen und H. Ulrich Hoppe. A Framework System for Intelligent Support in Open Distributed Learning Environments. *Journal of Artificial Intelligence in Education*, 9:256–274, 1998.

[Paw00] Jan M. Pawlowski. The Essen Learning Model - a Multi-Level Development Model. In *Proc. of the Int. Conf. on Educational Multimedia, Hypermedia & Telecommunications ED-MEDIA 00*, Montreal, Quebec, Canada, 2000.

[PHLV05] Niels Pinkwart, Andreas Harrer, Steffen Lohmann und Stefan Vetter. Integrating Portal Based Support Tools to Foster Learning Communities in University Courses. In Vladmiir Uskov, Hrsg., *Proceedings of the 4th In-ternational Conference on Web-Based Education WBE*, IFIP 18th World Computer Congress, Seiten pp. 201–206, Anaheim, CA., 2005. ACTA Press.

[Pin03] Niels Pinkwart. A Plug-In Architecture for Graph Based Collaborative Modeling. In Ulrich Hoppe, Felisa Verdejo und Judy Kay, Hrsg., *Shaping the Future of Learning through Intelligent Technologies. Proceedings of the 11th Conference on Artificial Intelligence in Education*, Seiten pp. 535–536, Amsterdam, 2003. IOS Press.

[Pos05] Postnuke. Postnuke project home page. http://www.post-nuke.net/, 2005.

[Roh04] Jean Rohmer. Yesterday, Today, and Tomorrow of AI Applications. Invited Talk, IFIP World Congress, 2004.

[SAI05] SAIL. SAIL project home page. http://docs.telscenter.org/display/SAIL/Home, 2005.

[SSRB00] Douglas Schmidt, Michael Stal, Hans Rohnert und Frank Buschmann. *Pattern–oriented Software Architecture – Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Chichester, 2000.