Eine Softwarearchitektur für serviceorientierte Fragetypen in E-Learning-Systemen

Benjamin Saul
Martin-Luther-Universität Halle-Wittenberg,
Institut für Informatik
Email: saul@informatik.uni-halle.de

Wolf Zimmermann
Martin-Luther-Universität Halle-Wittenberg,
Institut für Informatik
Email: zimmer@informatik.uni-halle.de

***Abstract*—Electronic assessments in E-Learning-Systems (e. g. ILIAS) can be used in exercises to reduce personal costs. Question types in computer science or mathematics often require formal inputs like terms or proofs. A string based comparison between the users submission and the solution is not sufficient to evaluate the correctness of the submission. Plugin systems, as used in ILIAS, can be used to implement new question types but this method is complex and error-prone. We show a service-oriented approach to implement new question types in ILIAS based on existing services. A compiler is used to translate user input language in service language which improves users readability and allows syntax checking before invoking a service call.**

***Zusammenfassung*—Lernplattformen wie ILIAS bieten die Möglichkeit, Test- und Übungsaufgaben zu stellen und automatisch auswerten zu lassen. Aufgabenstellungen der Informatik und Mathematik fordern insbesondere die Eingabe formaler Ausdrücke, z. B. Terme, Beweise und Programmcode. Hier reicht ein einfacher Textvergleich mit der Musterlösung nicht mehr aus, um alle Variationen gültiger Lösungen abzudecken. Die Erweiterungsmöglichkeiten von ILIAS durch Plugins sind für das Hinzufügen spezieller Fragetypen zu komplex und fehlerträchtig. Wir stellen einen serviceorientierten Ansatz vor, mit dem neue Fragetypen durch Anbinden und Kombinieren bereits vorhandener Werkzeuge in ILIAS integriert werden können. Durch den Einsatz von Übersetzern in den Adaptern lässt sich die Eingabesprache anpassen und auf syntaktische und statisch semantische Fehler überprüfen, noch bevor externe Services aufgerufen werden.**

***Index Terms*—E-Learning, E-Assessment, Übersetzerbau, serviceorientiert**

I. Übersicht

Eine wichtige Funktion von E-Learning-Systemen ist die Durchführung von Tests bzw. das Stellen von Aufgaben, deren Antworten automatisch ausgewertet werden. Diese werden von Dozenten erstellt und von Studenten beantwortet. Häufig auftretende Fragetypen sind Multiple-Choice Aufgaben und Lückentexte. Der Student wählt dabei eine der vorgegebenen Antworten bzw. ein Bild aus oder gibt einen Begriff ein. Das System prüft anschließend, ob die gegebene Antwort mit einer Musterlösung übereinstimmt und generiert die entsprechende Rückmeldung. Diese Rückmeldung ist i. d. R. eine Punktzahl, kann aber auch aus Erläuterungen zur Antwort bestehen. Derartige Tests sind gut zum Abfragen von Fachwissen geeignet, z. B. in der Biologie oder der Medizin.

Übungsaufgaben aus der Informatik bzw. Mathematik fordern häufig die Eingabe formaler Ausdrücke, z. B. Terme, Beweise und Programmausschnitte [1]. Bei diesen ist eine Prüfung auf Identität bzw. auf Ähnlichkeit zur Musterlösung oft nicht ausreichend. Die Eingaben variieren durch gesetzte Leerzeichen, Vertauschungen und alternative Formulierungen. Ein Beispiel für einen solchen Fragetyp aus der Mengenlehre ist in Beispiel 1 formuliert.

Beispiel 1 (Fragetyp: Mengenlehre - Operatoren)**.**
Aufgabe: *Bestimmen Sie* $\{a, b, c\} \cup \{c, d\}$
Lösung: $\{a, b, c, d\}$ *oder* $\{d, b, c, a\}$ *oder* $\{b, a, c, d\}$ ...

Hierbei wird deutlich, dass durch Permutation sehr viele korrekte Lösungen existieren. Diese vollständig für einen textbasierten Vergleich aufzulisten ist nicht praktikabel. Darüber hinaus treten weitere Probleme auf, die beim bisherigen Übungsbetrieb bemerkt wurden sind. Die Eingaben enthalten zusätzliche Leerzeichen, die nicht automatisch entfernt werden können. Derartige Eingaben wurden fälschlicherweise als Fehler bewertet und mussten manuell nachkorrigiert werden. Durch diese Fälle ist eine vollständige Auflistung aller Lösungen unmöglich.

Am Beispiel des an der Universität Halle-Wittenberg eingesetzten E-Learning-Systems ILIAS [2] soll untersucht werden, wie neue Fragetypen effizient umgesetzt werden können. ILIAS selbst unterstützt bereits eine Vielzahl an Fragetypen, darunter Multiple-Choice Aufgaben und Lückentexte. Letztere ermöglichen das Abfragen von frei eingebbaren Texten, die dann allerdings nur auf Identität mit der Musterlösung getestet werden. Die Mengenaufgabe aus Beispiel 1 wurde mit diesem Fragetyp erstellt.

ILIAS ist modul- bzw. serviceorientiert aufgebaut [2]. Einzelne Module sind voneinander unabhängig und verwenden gemeinsame Services (Abb. 1).

ILIAS wird basierend auf Feature-Requests entwickelt. Das heißt, neue Fragetypen für automatische Tests werden jedes mal als neues Plugin implementiert. Dadurch entsteht eine Vielzahl an Modulen für Fragetypen, die alle mit einem ähnlichen Grundmuster aufgebaut sind. Funktionen wie Anlegen, Kopieren und Löschen von Tests müssen implementiert werden, obwohl sich diese nur mi-

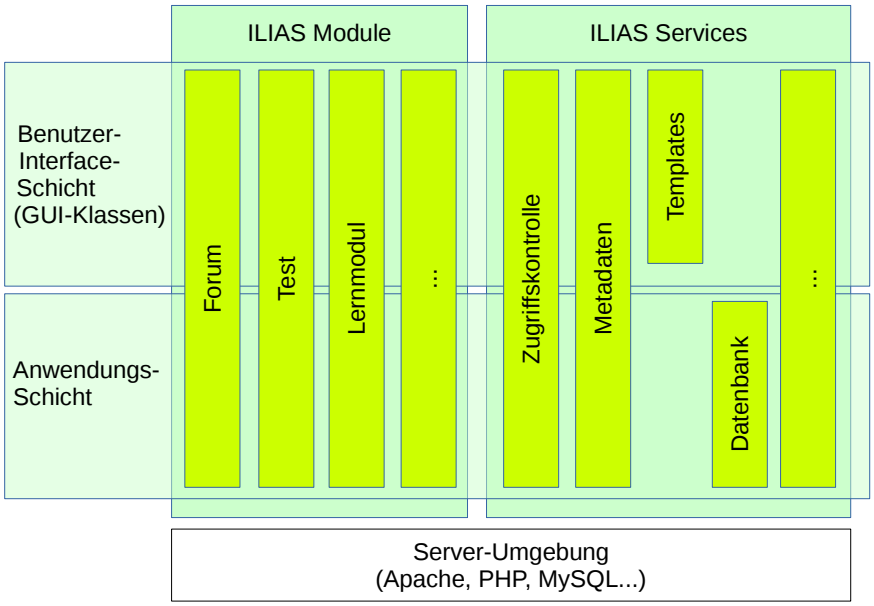

Abbildung 1. Der modulare Aufbau der E-Learning-Plattform ILIAS [2]

nimal bis gar nicht unterscheiden. Zusätzlich treten Alterungserscheinungen auf. Einzelne Fragetypen existieren in zwei oder mehr Varianten. Beispielsweise sei hier die Dateieinsendung erwähnt, die sowohl in Übungen als auch in Tests implementiert ist. Beide modellieren den gleichen Anwendungsfall, sind aber doppelt implementiert.

ILIAS unterstützt ein Feedback über richtige und falsche Antworten. Dieses wird manuell für jede erwartete Antwort eingetragen. Für Entscheidungsfragen wie Multiple-Choice ist dies völlig ausreichend. Bei Textfeldern benötigt man allerdings ein angepasstes Feedback, das sich auf individuelle Lösungen bezieht. Hier ist es nicht mehr möglich, alle gültigen Lösungen manuell zu erfassen.

Wird ein Fremdsystem zur Auswertung verwendet, z. B. das Computer-Algebra-System Maxima [3], so können ungewollte Nebeneffekte auftreten. Wird nämlich Eingabe der Studenten direkt als Maxima-Ausdruck ausgewertet, so muss man sicherstellen, dass die Studenten nicht betrügen und Funktionen wie den Mengenschnitt als Funktionsaufruf in Maxima explizit in die Antwort eingeben und sich so das Ergebnis berechnen lassen. Weitergehende Fragetypen, wie das Schreiben von Beweisen, erfordern außerdem komplexe Formulierungen, die unabhängig von den dahinter liegenden Werkzeugen sein sollten.

In dieser Arbeit wird eine geplante Architektur gezeigt, die auf Herausforderungen bei der Erstellung neuer Fragetypen eingeht. Im Wesentlichen soll die Architektur dabei

- externe Werkzeuge effizient einbinden lassen,
- minimal in das bestehende System (ILIAS) eingreifen,
- umfangreiches Feedback dem Anwender bereitstellen sowie
- fehlerhafte Eingaben bzw. Eingaben mit ungewollten Nebeneffekten detektieren und anzeigen.

Kapitel II zeigt verwandte Arbeiten, die sich bereits mit automatischer Testauswertung und den damit verbundenen Architekturen auseinandersetzen. Kapitel III stellt unseren Architekturansatz vor. In Kapitel IV werden die verschiedenen Herausforderungen bei der bisherigen Umsetzung bzw. bei der Arbeit mit ILIAS diskutiert. In Kapitel V werden die Ergebnisse zusammengefasst und ein Ausblick auf die geplanten Arbeiten gegeben.

II. Verwandte Arbeiten

Im Umfeld von E-Learning und im speziellen der automatisierten Auswertung von Fragestellungen im Bereich von Übungen und Tests gibt es viele Arbeiten. Schon in [4] wurden einige dieser Projekte vorgestellt. Hier liegt der Schwerpunkt auf der Generierung, Auswahl und Präsentation von Testaufgaben, um einen möglichst hohen Lernerfolg bei den Studenten zu erzielen. In [5] wird eine entsprechende Architektur präsentiert. Die Aufgaben selbst werden mithilfe einer Wissensbasis ausgewertet. Der Student erhält ein auf sein Lernverhalten angepasstes Feedback. Diese eigenständig laufenden Systeme sind gut erweiterbar, aber noch nicht in ILIAS integriert worden.

Serviceorientierte Implementierungen automatisierter Tests, hier im speziellen die Auswertung von Programmcode, finden sich in [6] und [7]. Während ersteres sich auf eine Architektur konzentriert, welche leicht um weitere Programmiersprachen erweitert werden kann, liegt bei der zweiten Arbeit der Fokus auf der Bereitstellung einer virtuellen Programmierumgebung. Eine Umsetzung für eine Programmierumgebung innerhalb von ILIAS gibt es in [8]. Hier wurde ein Plugin entwickelt, welches Code verschiedener Programmiersprachen automatisch auswerten kann, z. B. durch Unittests für die Sprache Java. Diese

Beitrag	Erweiterbarkeit um neue Fragetypen	Integrierbarkeit in ILIAS	Feedbackmöglichkeiten	Eingabeprüfung
[5]	+	0	+	0
[6]	+	0	0	0
[7]	0	+	+	+
[8]	0	+	+	+
[9]	+	0	+	0
[11]	+	-	0	0
[12]	-	+	+	+

Tabelle I
Übersicht über verwandte Arbeiten im Bereich der elektronischen Tests. Es wird unterschieden zwischen guter Unterstützung bzw. bereits in ILIAS umgesetzt (+), keiner genauen Angabe bzw. eigenständiges System (0) sowie keiner Unterstützung (-)

Arbeiten sind auf Programmierübungen spezialisiert. Eine Erweiterung um beliebige Eingabeformate wurde nicht diskutiert.

Das automatische Übungs- und Prüfungssystem Jack [9] bietet ein server-basiertes System, welches in Lernplattformen eingebunden werden kann. Hier wurden diverse Fragetypen implementiert, darunter Multiple-Choice Aufgaben sowie mathematische Fragestellungen. Die Auswertung der Ausdrücke erfolgt unter anderem durch Computer-Algebra-Systeme. Das Übungssystem bietet gute Möglichkeiten für die Erweiterung um weitere Fragetypen, wurde allerdings noch nicht in ILIAS integriert.

Eine serviceorientierte Architektur, im Speziellen für Freitext-Fragen, findet sich in [10] und [11]. Die flexible Integration von Fremdsystemen und Werkzeugen mithilfe des Software-as-a-Service Ansatzes wird hier vorgestellt. Die Lernplattform selbst wird als Basis für diverse Services und Werkzeuge implementiert. Voraussetzung dafür sind Standards für die diversen Schnittstellen. Es findet also keine Integration in bestehende Systeme in diesem Sinne statt, sondern die Entwicklung eines alternativen E-Learning-Systems, welches dann flexibler in der Zusammenstellung sein soll.

Eine Umsetzung für mathematische Fragestellungen als Plugin für ILIAS ist das Stack-Plugin [12], [2]. Dieses sowohl in Funktionalität als auch im Code umfangreiche Plugin bietet eine Schnittstelle für das Auswertungssystem STACK, welches wiederum auf dem Computer-Algebra System Maxima basiert. Terme können auf diverse Arten miteinander verglichen werden und es gibt eine Auswertestrategie für Folgefehler berücksichtigende Punktevergabe. Eine Anbindung weiterer Werkzeuge und Services bzw. die Erweiterung des Plugins ist nicht geplant. Es ist möglich, die Mengenaufgabe aus Beispiel 1 mit diesem Plugin umzusetzen. Allerdings müssen die Studenten die Eingabesprache von Stack bzw. Maxima benutzen.

Die hier vorgestellten Arbeiten beschreiben eigenständige Systeme und Erweiterungen bestehender Systeme. Der Fokus liegt dabei entweder auf der Bereitstellung von Feedback sowie der Integration in ein bestehendes Lernsystem oder auf einer erweiterbaren Architektur. Eine Arbeit, die sowohl in ein bestehendes E-Learning-System integriert ist, als auch um neue Fragetypen erweiterbar ist und den Studenten Hilfestellungen durch Feedback und Eingabeprüfungen leistet, gibt es unserer Kenntnis nach noch nicht. Diese Punkte versuchen wir in unserer Arbeit zu erfüllen. Tabelle I fasst noch einmal die hier betrachteten Merkmale der verwandten Arbeiten zusammen.

III. Eine Architektur für modularisierte Fragetypen

Eine typische Frage besteht aus Aufgabenstellung, Musterlösung sowie einem Algorithmus zur Berechnung einer Punktzahl bzw. Rückmeldung zur gegebenen Antwort. Lösungen und Antworten sind häufig textuell und werden in Eingabefelder eingetragen. Es gibt aber auch Fragetypen, die grafische Objekte wie geometrische Zeichnungen oder Diagramme abfragen. Die Algorithmen zur Evaluierung der Eingaben sind für viele Aufgabenstellungen sehr komplex. Daher sollten diese nicht in dem E-Learning-System implementiert werden, sondern bestehende Services wie Computer-Algebra-Systeme eingebunden werden.

Derzeit werden solche Schnittstellen explizit im entsprechenden Plugin in ILIAS implementiert. Die Implementierung umfasst außerdem zusätzliche Funktionen, z. B. für Datenbankzugriffe und Kopierfunktionen. Objektorientierte Programmierung schränkt den Implementierungsaufwand nur zum Teil ein. Neue Plugins sind nicht aus Bestandteilen anderer Plugins erzeugbar. Eine gemeinsame Middleware, welche die Verwaltungsaufgaben übernimmt, ist daher erstrebenswert. Abbildung 2 zeigt, wie verschiedene Komponenten über eine gemeinsame Middleware angebunden werden.

Durch die Auslagerung der Auswertungsfunktionen lassen sich die bisherigen umfangreichen Plugins für Fragetypen in mehrere zusammenstellbare Module aufteilen. Die Aufteilung erfolgt dabei anhand der verschiedenen Aspekte des Fragetyps. Diese werden in Abbildung 3 gezeigt.

Nach der Eingabe einer Antwort in eine entsprechende Eingabemaske erfolgt zunächst eine Überprüfung auf syntaktische bzw. statisch semantische Fehler durch einen geeigneten Übersetzer oder eine vergleichbare Methode. Die Auswertung der Lösung geschieht dann durch einen Service, welcher eventuell über Adapter aufgerufen werden muss. Da die Eingabesprache der externen Services

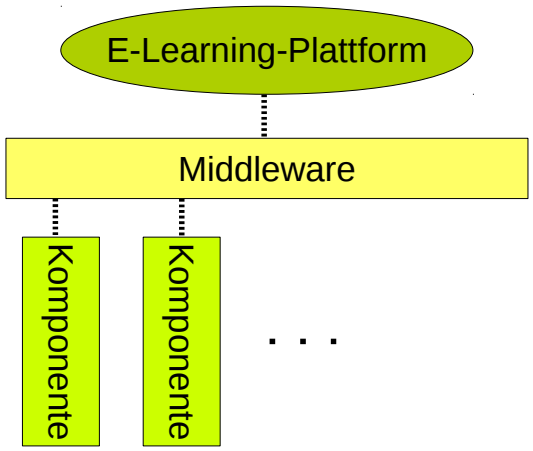

Abbildung 2. Anbindung von Komponenten für Vergleichsfunktionen

nicht für die Eingabe von bestimmten Fragestellungen konzipiert ist, sollte ein lesbares Eingabeformat durch den Übersetzer bereitgestellt werden. Dadurch kann die Eingabesprache auf das abgefragte Themengebiet beschränkt werden, was Vorteile für die Studenten bringt.

Die Middleware aus Abb. 3 soll als Fragetypen-Plugin in ILIAS umgesetzt werden. Geplant ist ein einziger Fragetyp, bei dem die Textfelder durch verschiedene Services ausgewertet werden können, welche zuvor im Plugin registriert werden können. Dies hat den weiteren Effekt, dass die Anzahl der Fragetypen nicht mit jedem Service ansteigt, was die Übersichtlichkeit erhöht. Ein neuer Fragetyp benötigt Komponenten für

- Syntaxprüfung und einfache Konsistenzprüfungen,
- Übersetzer bzw. Formatierer zwischen Eingabesprache und Fremdsystemsprache,
- Schnittstellen aufzurufender Services,
- Feedbackfunktionen, die Antworten der Services auf Punkte und sonstige Meldungen abbilden sowie
- Eingabemaske.

Da alle Eingaben inklusive der Musterlösung textuell erfolgen sollen, entfallen zusätzliche Datenbankeinträge. Sicherheitskritische Aspekte, wie der uneingeschränkte Zugriff durch Plugins, werden dadurch entschärft. Wird der gleiche Service verwendet, können die Anbindungen und der Übersetzer zu großen Teilen erneut verwendet bzw. ergänzt werden. Eingabemasken für grafische Elemente, z. B. Diagramme oder geometrische Skizzen, müssen im Hintergrund ein textuelles Format wie XML zur Verarbeitung verwenden. Eine Anbindung von Systemen zur grafischen Eingabe wie WIRIS ([13]) erfolgt dann in den Komponenten der Eingabemaske. In den weiteren Komponenten wird nur die textuelle Form der grafischen Objekte weiter verarbeitet.

Beispiel 2 (Mengenlehre - Fortsetzung)**.** *Das erste hier betrachtete Beispiel ist die Umsetzung eines Fragetyps für Mengenaufgaben. Der Student trägt dazu die Ergebnismenge einer Aufgabe in ein Textfeld ein, und die eingegebene Menge wird anschließend mit einer Ergebnismenge verglichen.*

Für die Umsetzung des Fragetyps ist es notwendig, die Eingabe zunächst geeignet zu formatieren und anschließend einen Vergleich zwischen zwei Mengen durchzuführen, beispielsweise mit Maxima. Schreibfehler, wie vergessene Kommata oder Klammern, können zu Fehlern führen, sodass der Term nicht ausgewertet werden kann. Diese sollten dem Studenten also schon vor dem Berechnen der Punktzahl angezeigt werden. Die dabei benötigten Funktionalitäten sollten soweit wie möglich nicht innerhalb des Moduls stehen, sondern als Service angeboten werden.

Im Falle der Mengenaufgabe könnte man das Maxima-Werkzeug verwenden, um die Menge aus der Eingabe mit der Menge der Musterlösung zu vergleichen. Eine typische Fragestellung wurde in Beispiel 1 gezeigt.

Als Musterlösung kann direkt der Term der Aufgabenstellung verwendet werden. Die Eingabe für Maxima ergibt sich für die Beispieleingabe $\{a, b, d\}$ daraus wie folgt.

```
input:  set(a,b,d);
muster: adjoin(set(a,b,c),set(c,d));
result: isEqual(x,muster);
```

Dabei fällt auf, dass die Schreibweise der Aufgabenstellung kürzer ist und der schriftlichen Notation angepasst ist. Der Student muss sich keine neue Syntax aneignen sondern kann die mathematische Notation verwenden.

Weitere Rückgabewerte von Maxima können hierbei auch die Elemente sein, die in der eingesandten Lösung fehlen oder fälschlicherweise in die Menge aufgenommen wurden. Als Feedback lassen sich diese Elemente dann dem Studenten anzeigen. Auch ist eine differenzierte Punktevergabe möglich, indem Punktabzüge entsprechend den falschen Elementen stattfinden. Dies ist in dieser Art mit einem Identitätsvergleich zwischen Texten nicht möglich.

Später können einige der Komponenten wieder verwertet werden, z. B. bei Fragetypen zu Vielfachmengen. Hier ist die Anbindung an Maxima sowie die Feedback-Auswertung die selbe.

Beispiel 3 (Fragetyp: Abstrakte Datentypen)**.** *Das zweite Beispiel, welches derzeit geplant ist, ist ein Fragetyp im Bereich abstrakter Datentypen. Die typische Fragestellung hierbei ist es, Terme eines gegebenen abstrakten Datentyps auf Gleichheit zu untersuchen. Durch Anwendung der Axiome bzw. Regeln auf bestimmte Stellen von Termen ist es möglich, einen Term in einen anderen, im abstrakten Datentypen äquivalenten, Term umzuwandeln. Eventuell müssen dazu auch Unterterme substituiert werden. Eine beispielhafte Aufgabenstellung nebst Lösung lautet wie folgt.*

Aufgabe: *Gegeben Sei der abstrakte Datentyp* `Mystery` *mit*

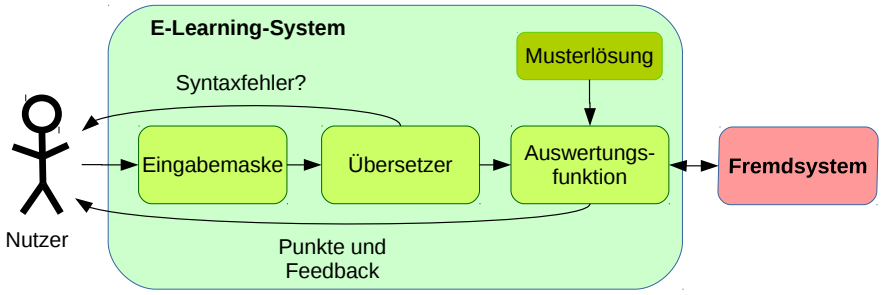

Abbildung 3. Übersicht über die Architektur für die automatisch Auswertung von Aufgaben

spec Mystery is
sorts A, U
operations
feuer : −> A Konstruktor
erde : A −> A Konstruktor
wasser : A x A −> A
apfel : −> U Konstruktor
orange : A x U −> U Konstruktor
banane : U x U −> U
kiwi : U −> A
vars
n,m : A
x,y : U
axioms
B1 : wasser (feuer, n) = n
B2 : wasser (erde(n), m)
= erde(wasser (n, m))
C1 : banane(apfel, y) = y
C2 : banane(orange(n, x), y)
= orange(n, banane(x, y))
C3 : kiwi(apfel) = feuer
C4 : kiwi(orange(n, x))
= wasser (n, kiwi(x))

Zeigen Sie, dass

`wasser(kiwi(apfel), kiwi) = kiwi`

unter Angabe der angewendeten Regeln und Substitutionen. Markieren Sie die betreffenden Unterterme mit eckigen Klammern!

wasser([kiwi(apfel)], kiwi)
= [wasser(feuer, kiwi)] mit C3
= kiwi mit B1 und S=[kiwi/n]

Die Schritte müssen dann in ein entsprechendes Format übersetzt werden und mittels geeigneter Prüfer, wie z. B. PVS [14], ausgewertet werden. Zuvor können die Eingaben auf statische Bedingungen, wie Typprüfungen der Terme, Konsistenz der Substitutionen und Vorhandensein von Variablen, geprüft werden. Diese Übersetzung ist für zukünftige Arbeiten geplant. Auch die Problematik mit Folgefehlern bei dieser Aufgabenart ist noch nicht gelöst. Verfahren mit Entscheidungsbäumen wie sie in [12] erfolgen, wurden noch nicht weiter betrachtet.

Dem Studenten kann nach der Auswertung angezeigt werden, welche Umformungsschritte falsch oder unvollständig waren. Dazu wird die Aussage des externen Prüfers über Widersprüche bzw. offene Beweisverpflichtungen in den Schritten ausgewertet. Beides stellt ein für Studenten nützliches Feedback dar.

Die vorgestellte Architektur soll in ILIAS integriert werden, mit der neue Fragetypen implementiert werden können. Die beiden Beispiele veranschaulichen auch, dass externe Werkzeuge bzw. Services notwendig sind, um Aufgaben korrekt bewerten zu können. Zusätzlich zu der Integration in bestehende bzw. genutzte Systeme ist es wichtig, dass den Studenten ein umfangreiches Feedback gegeben wird. Einfache formale Fehler können durch Übersetzer erkannt werden, was Studenten von Flüchtigkeitsfehlern entlastet.

IV. Diskussion

Der hier vorgestellte Ansatz soll die in Kapitel I gezeigten Anforderungen erfüllen. Insbesondere soll eine Möglichkeit für die Erweiterung von ILIAS geboten werden, die ohne Neuentwicklung von Plugins für jeden einzelnen Fragetyp auskommt. Plugins in ILIAS haben den vollen Zugriff auf sämtliche Bereiche der Lernplattform und sind somit zum einen schwerer wartbar und bergen zum anderen auch Sicherheitsrisiken.

Die Entkopplung der einzelnen Komponenten eines Fragetyps stellt dabei ein wesentliches Argument dar. Möchte man beispielsweise einen bestehenden Aufgabentyp nutzen und nur die Eingabemethode verändern, so tauscht man lediglich die dafür genutzten Komponenten aus. Voraussetzung dafür ist, dass die Eingabe am Ende ein textuelles Format liefert, welches von den Schnittstellen übertragen werden kann. Diese Einschränkung ist für formale Eingaben vertretbar.

Die Umsetzung der Architektur als Plugin in ILIAS ist noch nicht abgeschlossen. Erste Implementierungen für den Mengenaufgabentyp aus Beispiel 1 sind erfolgt. Die Umsetzung eines ähnlichen Fragetyps für Vielfachmengen zeigte, dass große Teile der Implementierung erneut verwendet werden konnten. Nur die Funktion für den

Vergleich der Menge wurde angepasst, damit mehrfach auftretende Elemente auch erfasst werden.

Auf Grundlage der für Plugins genutzten Schnittstellen von ILIAS soll die Middleware umgesetzt werden. In den entsprechenden Funktionsaufrufen wird dann auf die Komponenten, wie Übersetzer und Eingabemasken, verwiesen.

Die Erweiterung um weitere Fragetypen erfolgt anschließend in speziellen Modulen. Da diese nur sehr wenige Schnittstellen bereitstellen müssen, sollte die Implementierung weniger aufwendig sein als die Implementierung eines Plugins. Der Vorteil gegenüber eines objektorientierten Ansatzes liegt dabei bei der anschließenden Kombination der Module. Wie im Beispiel 1 vorgestellt, können Komponenten wie die Maxima-Anbindung erneut genutzt werden.

Die Integration von in Kapitel II vorgestellten Systemen wurde in dieser Arbeit nicht näher betrachtet, da diese nicht alle Anforderungen an Erweiterbarkeit, Feedback und Eingabeprüfung unterstützen. Die Verwendung vorhandener Plugins, wie dem Stack-Fragetyps, lässt die Erweiterung um weitere formale Sprachen nicht zu.

Bei der Umsetzung der Schnittstellen der Middleware sind noch Fragen darüber offen, wie die Kommunikation mit dem Fremdsystem zu erfolgen hat, wenn dieses über einen längeren Zeitraum das Ergebnis berechnet. Weiterhin wird die Bereitstellung von Feedback durch die Architektur von ILIAS erschwert. Eine Rückmeldung für Studenten muss mit seiner Eingabe verknüpft in der Datenbank gesichert werden, bevor sie dem Studenten angezeigt werden kann. Dieses Prinzip erscheint unveränderlich und könnte zu Problemen beim Generieren bzw. Auswerten der Rückmeldungen führen.

Bei einigen Versionsänderungen von ILIAS kam es zu Veränderungen in den Schnittstellen für Plugins. Inwieweit die Wartbarkeit einer als Plugin entwickelten Middleware gegeben ist, kann nicht abgeschätzt werden. Kleinere Anpassungen sind ohne weiteres umsetzbar, aber durch die auf Feature-Requests basierte Entwicklung sind die zukünftigen Entwicklungen von ILIAS nicht absehbar.

V. Zusammenfassung

Die hier vorgestellte Architektur dient als Vorschlag für eine effiziente Integration neuer textbasierter Fragetypen in E-Learning-Systemen. Die Auswertung der Eingaben erfolgt dabei über externe Services. Ein zwischen Nutzereingabe und Komponente zum Auswertealgorithmus geschalteter Übersetzer ermöglicht die Formulierung und die syntaktische und semantische Überprüfung einer auf den Fragetypen angepassten domänenspezifischen Sprache. Der Sprachentwurf und der Übersetzer können im einfachsten Fall auch reguläre Ausdrücke sein. Die syntaktische und semantische Überprüfung ermöglicht das frühzeitige Erkennen von Eingabefehlern und senkt den Aufwand für Nachkorrekturen durch Autoren.

Die Entwicklung und Nutzung derartiger Frameworks stärkt die Anwendbarkeit von E-Learning Systemen und erhält gleichzeitig deren Software-Architektur. Neue Fragetypen können im Idealfall aus bereits bestehenden Komponenten zusammengesetzt werden. Dies stärkt die Wiederverwendbarkeit und beugt der Explosion der Anzahl von Fragetypen vor. Der Aufwand der Entwicklung des notwendigen Übersetzers ist im Rahmen des Aufwandes für die entsprechenden Adapter.

Eine modulare Zusammensetzung neuer Fragetypen aus bestehenden Vergleichsoperatoren, Fremdsystemen und im Idealfall auch GUI-Elementen beschleunigt die Entwicklung angepasster Fragestellungen. Werkzeuge, die die Eingabe unterstützen, wie z. B. WIRIS [13], können in neue oder bestehende Fragetypen integriert werden und es muss nur die entsprechende Verarbeitungssprache angepasst werden.

Die Architektur soll in ILIAS mithilfe eines Plugin integriert werden. Damit sollen insbesondere auch komplexere Aufgaben wie z. B. Induktionsbeweise, Konstruktion von Grammatiken und Automaten oder Herleitungen im Logikkalkül implementiert werden.

Literatur

[1] M. Daniel, N. Köcher, and R. Küstermann, "E-Klausuren in der angewandten Mathematik – Herausforderungen & Lösungen." in *DeLFI*, 2014, pp. 265–270.

[2] I. open source e Learning e.V. (2017) ILIAS E-Learning. [Online]. Available: https://www.ilias.de/

[3] (2017) Maxima, a computer algebra system. [Online]. Available: http://maxima.sourceforge.net/

[4] J. W. Rickel, "Intelligent computer-aided instruction: A survey organized around system components," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 1, pp. 40–57, 1989.

[5] J. Gonzalez-Sanchez, M. E. Chavez-Echeagaray, K. VanLehn, W. Burleson, S. Girard, Y. Hidalgo-Pontet, and L. Zhang, "A system architecture for affective meta intelligent tutoring systems," in *International Conference on Intelligent Tutoring Systems*. Springer, 2014, pp. 529–534.

[6] M. Amelung, K. Krieger, and D. Rösner, "E-assessment as a service," *IEEE Transactions on Learning Technologies*, vol. 4, no. 2, pp. 162–174, 2011.

[7] T. Richter, S. Rudlof, B. Adjibadji, H. Bernlöhr, C. Grüninger, C.-D. Munz, A. Stock, C. Rohde, and R. Helmig, "Viplab: a virtual programming laboratory for mathematics and engineering," *Interactive Technology and Smart Education*, vol. 9, no. 4, pp. 246–262, 2012.

[8] M. M. Madrid and M. Lohmann, "Ein Fragetyp für Programmieraufgaben als Erweiterung des Learning-Management-Systems ILIAS." in *DeLFI*, 2015, pp. 341–343.

[9] M. Goedicke and M. Striewe, "10 Jahre automatische Bewertung von Programmieraufgaben mit Jack-Rückblick und Ausblick," *INFORMATIK 2017*, 2017.

[10] M. AL-Smadi and C. Guetl, "Service-oriented flexible and interoperable assessment: towards a standardised e-assessment system," *International Journal of Continuing Engineering Education and Life Long Learning*, vol. 21, no. 4, pp. 289–307, 2011.

[11] M. Al-Smadi and C. Gütl, "Soa-based architecture for a generic and flexible e-assessment system," in *Education Engineering (EDUCON), 2010 IEEE*. IEEE, 2010, pp. 493–500.

[12] C. Sangwin, *Computer aided assessment of mathematics*. OUP Oxford, 2013.

[13] D. Marquès, R. Eixarch, G. Casanellas, B. Martínez, and T. J. Smith, "WIRIS OM tools: a semantic formula editor," 2006.

[14] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," in *International Conference on Automated Deduction*. Springer, 1992, pp. 748–752.