Ein benutzerfreundlicher, generischer, durch Plugins erweiterbarer ProFormA-Programmieraufgaben-Editor

Paul Reiser[1], Robert Garmann[2], Felix Heine[3]

Abstract: Programming assignments defined in the ProFormA format allow for an easy exchange between heterogeneous graders and learning management systems. The ProFormA format can be extended by foreign, grader-specific formats so that programming assignments can be evaluated across various grader systems. While editing ProFormA files with traditional tools such as generic (visual) XML editors is possible, it is neither intuitive nor especially user-friendly, and can be challenging for users that are not experienced in XML. This paper presents an editor that has been designed and created for editing programming assignments in the ProFormA format. Editing foreign formats is supported in a generic way and can be further extended by using special plug-ins that are specifically designed to improve the usability when editing these formats.

Abstract: Programmieraufgaben im ProFormA-Aufgabenformat können zwischen Gradern und Lernmanagementsystemen ausgetauscht werden. Um Grader-übergreifend nutzbar zu sein, erlaubt das ProFormA-Format die Integration von Fremdformaten für spezielle Grader-Anforderungen. Die an der Erstellung und Nutzung der Aufgabe beteiligten Personen können ProFormA-Dateien mit herkömmlichen ZIP- und XML-Werkzeugen bearbeiten. Bei komplexen Aufgaben ist diese Form der Bearbeitung jedoch nicht benutzerfreundlich. In diesem Beitrag stellen wir ein Werkzeug vor, mit dem man Programmieraufgaben im ProFormA-Format editieren kann. Fremdformate werden einerseits generisch unterstützt. Zudem erlaubt der Editor Erweiterungen durch Plugins, um die Benutzerfreundlichkeit für Fremdformate zu erhöhen.

Keywords: E-Assessment, Programmieraufgabe, generischer Editor, Plugin

1 Einleitung

In [St15] wurde das sog. ProFormA-Dateiformat eingeführt, mit dem Programmieraufgaben in einer standardisierten Form zwischen Lernmanagementsystemen (LMS) und Autobewertern (sog. Gradern) ausgetauscht werden können. Wesentliche Bestandteile einer Programmieraufgabe in diesem Format sind der Aufgabentext, Musterlösung(en), Bewertungshinweise sowie die Konfiguration verschiedener Prüfroutinen ggf. unter Nutzung von inkludierten Dateien oder externen Ressourcen. Technisch handelt es sich um eine XML-Datei, die entweder selbst alle benötigten Daten enthält oder die Teil eines ZIP-Archivs mit verschiedenen Anlagen ist. Das ProFormA-Format ist ein offenes For-

[1] Hochschule Hannover, ZLB – E-Learning-Center, Expo Plaza 12, 30539 Hannover, paul.reiser@stud.hs-hannover.de
[2] Hochschule Hannover, Fakultät IV Wirtschaft und Informatik, Ricklinger Stadtweg 120, 30459 Hannover, robert.garmann@hs-hannover.de
[3] Hochschule Hannover, Fakultät IV Wirtschaft und Informatik, Ricklinger Stadtweg 120, 30459 Hannover, felix.heine@hs-hannover.de

mat, in das sich Fremdformate für spezielle Grader an genau vorgesehenen Stellen der XML-Datei einbinden lassen. Eine ProFormA-Datei kann mit allgemein verfügbaren Werkzeugen erstellt und modifiziert werden. Eine XML-Datei zu editieren ist jedoch alles andere als benutzerfreundlich. Zudem gestaltet sich der Prozess (ZIP-Archiv entpacken, XML-Datei editieren, ZIP-Archiv neu packen) umständlich.

Wir verwenden den Begriff *task* für eine als ProFormA-Datei vorliegende Programmieraufgabe. Bei der Nutzung einer task sind i. W. drei Rollen beteiligt: ein Aufgabenautor erstellt die task; eine Lehrperson nutzt die task und passt sie ggf. an den Lehrkontext an; ein Student konsumiert die Aufgabe als Lernobjekt, kommt aber nur mit Teilen einer task in Berührung (Aufgabentext, vorgegebene Codefragmente und Bibliotheken, vgl. Abb. 1). In diesem Beitrag stellen wir einen Editor vor, der die Aktivitäten *task erstellen* und *task anpassen* auf benutzerfreundliche Weise unterstützen kann. Den Grad der Benutzerfreundlichkeit beurteilen wir dabei intuitiv vor dem Hintergrund eigener Erfahrungen mit verschiedenen Gradern. Wir gehen davon aus, dass insbesondere Lehrpersonen von diesem Editor profitieren. Darüber hinaus kann der Editor auch Aufgabenautoren Vorteile verschaffen, insbesondere, wenn der jeweilige Grader keine Build-Mechanismen für die Erstellung einer ProFormA-task anbietet. Der vorgestellte Editor soll die Erstellung und Anpassung einer Aufgabe benutzerfreundlich gestalten und technische Details des ProFormA-Formats vor dem Nutzer verbergen. Die o. g. Offenheit des ProFormA-Formats stellt die besondere Anforderung an den Editor, dass a priori unbekannte Fremdformatelemente auf benutzerfreundliche Weise angezeigt, editiert, hinzugefügt und entfernt werden können. Darüber hinaus muss der Editor sicherstellen, dass das gespeicherte Ergebnis wieder eine gültige ProFormA-Aufgabe ist, die allen Integritätsbedingungen des ProFormA-Formats sowie allen Integritätsbedingungen der referenzierten Fremdformate genügt.

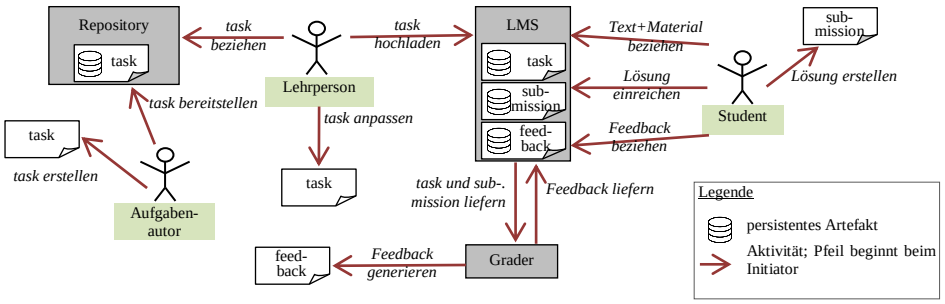

Abb. 1: An Erstellung und Nutzung einer task beteiligte Rollen und Systeme

Dieser Beitrag beginnt mit einer Beschreibung der Anforderungen an den Editor (Abschnitt 2) und der Betrachtung verwandter Arbeiten (Abschnitt 3). Abschnitt 4 stellt den Basiseditor für die Standardelemente einer task vor. Danach gehen wir auf zwei alternativ einsetzbare Lösungen ein, mit denen unser Editor der Offenheit des ProFormA-Formats begegnet. Abschnitt 5 beschreibt eine generische Editorkomponente für beliebige Grader und deren durch ein XML-Schema beschriebenes Aufgabenformat. Danach

skizzieren wir in Abschnitt 6 das in unserem Editor geschaffene Plugin-System, in das sich für spezielle Grader handgefertigte Editorkomponenten einbinden lassen.

2 Anforderungen an den Editor

Der hier vorgestellte Editor entstand aufgrund der folgenden Anforderungen:

R1	Der Editor kann Dateien im ProFormA-Format (XML oder ZIP) laden, editieren und speichern sowie neue ProFormA-Dateien erzeugen.
R2	Fremdformate werden als XML-Schemadefinition (XSD) mit jeweils eigenen XML-Namensräumen vom Editor erkannt. Der Editor erlaubt Laden, Anzeigen und Editieren der Fremdformatelemente.
R3	Fremdformatelemente können hinzugefügt und entfernt werden.
R4	Fremdformatelemente sind benutzerfreundlich editierbar.
R5	Die Eingaben werden auf Gültigkeit geprüft. Dies betrifft die standardisierten ProFormA-Elemente und die Fremdformatelemente.
R6	Alle Elemente werden mit einem Hilfetext ausgestattet. Auch dies betrifft die standardisierten ProFormA-Elemente und die Fremdformatelemente.

3 Verwandte Arbeiten

Das ProFormA-Format ist noch jung. Daher gibt es bisher nur einen uns bekannten Ansatz, tasks mit einem Editor zu bearbeiten [PRB17]. Durch Verwendung der Programmiersprache Javascript ist dieser Editor direkt in ein webbasiertes LMS integrierbar und daher für Lehrpersonen gut zugänglich. Der Editor unterstützt die meisten Standard-Elemente des ProFormA-Formats. Darüber hinausgehende Fremdformatelemente werden für den Grader Praktomat unterstützt. Die Erstellung von Praktomat-Aufgaben ist offenbar auch der derzeitige Haupteinsatzzweck des Editors. Weitere Fremdformate werden nicht unterstützt.

Die Anforderungen bzgl. der Fremdformatunterstützung legen einen Vergleich mit Editoren nahe, die aus einer gegebenen XSD dynamisch eine Benutzeroberfläche generieren, mit der zugehörige XML-Dateien erstellt oder bearbeitet werden können. Beispielhaft sei das kommerzielle Produkt *Generic Visual XML Editor* von oxygen[4] genannt. Derartige generische Editoren sind ausgereifte Werkzeuge, haben jedoch zwei Nachteile. Zum einen handelt es sich in der Regel um hochkomplexe Werkzeuge, in die sich nicht jede Lehrperson und nicht jeder Aufgabenautor einarbeiten will. Zum Zweiten sind solche Editoren darauf angewiesen, dass jegliches Domänenwissen zum ProFormA-Format und zu den Fremdformaten in den XSD abgebildet ist. Manche Zusammenhänge zwischen Fremdformat und ProFormA-Format lassen sich jedoch nicht eindeutig in XSD

[4] www.oxygenxml.com

abbilden. Wenn bspw. in Elementen des ProFormA-Formats Bezeichner enthalten sind, die von Elementen des Fremdformats referenziert werden, lässt sich dies nicht in den beteiligten Schemas so definieren, dass referentielle Integrität geprüft werden kann.

4 Unterstützungsfunktionen für ProFormA-Standardelemente

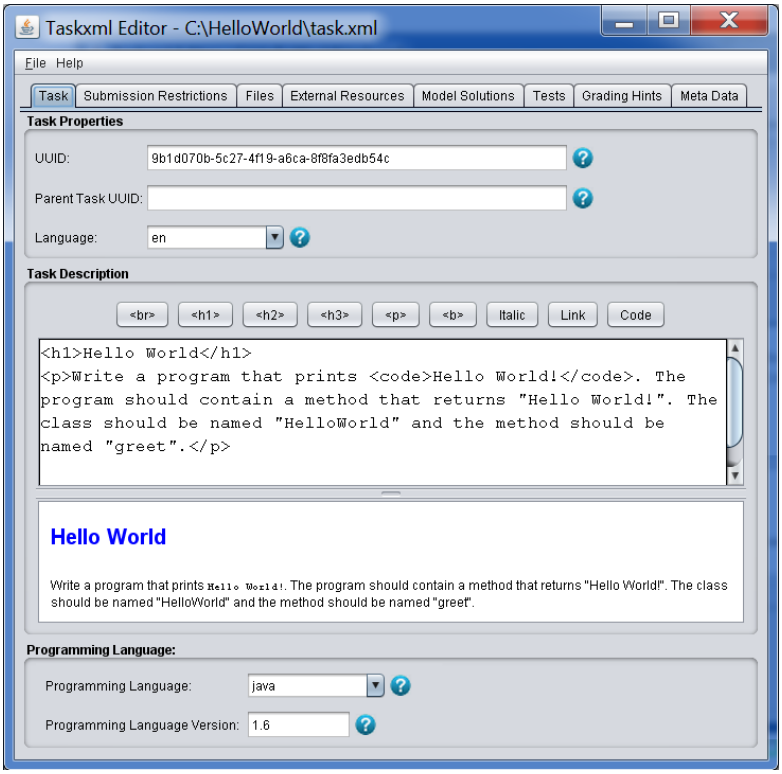

Abb. 2: Editor für das ProFormA-Format

Der in Java programmierte Editor kann tasks in XML- oder ZIP-Gestalt laden, editieren und speichern[5] (R1). tasks werden über eine grafische Benutzerschnittstelle abgebildet, die auf das ProFormA-Format zugeschnitten ist. Die Verzweigungen der Hauptelemente des ProFormA-Formats stellen eine logische Gruppierung dar, nach der sich der Editor richtet. Für die Gruppierung der einzelnen Hauptelemente werden Tabs (Karteireiter) verwendet (Abb. 2). Innerhalb der Tabs werden Komponenten eingebunden, die sich auf die Darstellung, das Editieren und die Validierung von einzelnen Unterelementen spezialisieren. Die Komponenten bieten eine benutzerfreundliche Schnittstelle zu den

[5] Das Laden des ZIP-Formats ist derzeit noch nicht umgesetzt aber fest eingeplant.

einzelnen Elementen an. Beispielsweise wird für die Aufgabenbeschreibung eines tasks eine HTML-Vorschau des Beschreibungstextes angezeigt. Über entsprechende Buttons lassen sich HTML-Formatierungen in den Beschreibungstext einfügen.

Zu jedem Eingabefeld wird eine abrufbare Hilfestellung für das jeweilige ProFormA-Element bereitgestellt. Eingabefelder, die einen Pflichtwert oder einen Wert in einem vorgegebenen Wertebereich erwarten, werden vom Editor auf Eingabefehler untersucht. Benutzer werden auf betroffene Stellen mit entsprechenden Fehlerhinweisen verwiesen.

5 Generische Unterstützung von Erweiterungen

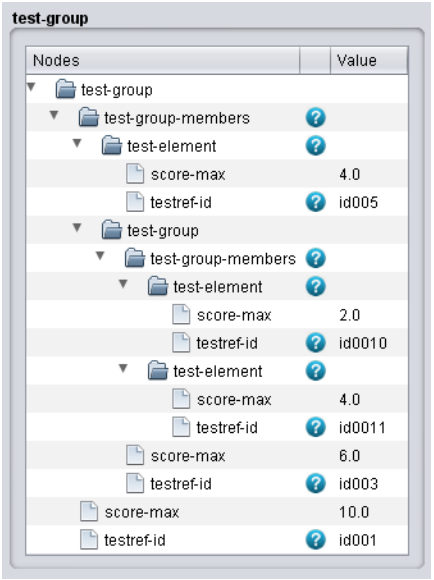

Abb. 3: Generische Schnittstelle

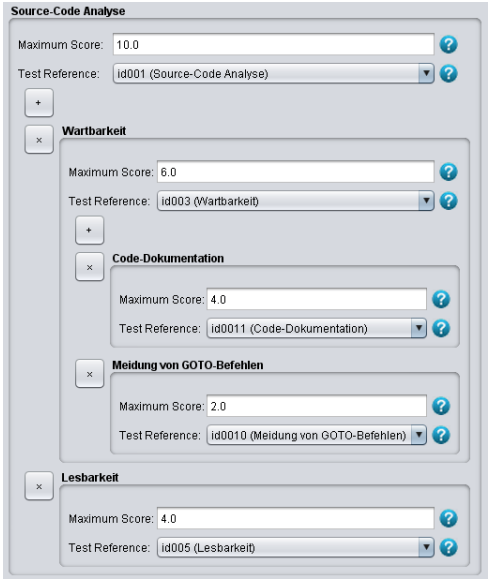

Abb. 4: Handgefertigte Plugin-Schnittstelle

Elemente aus Fremdformaten lassen sich an vorgesehenen Stellen in eine task einbinden. Die Definitionen solcher Elemente sind in XML-Schemadefinitionen (XSD) beschrieben, die über die task verknüpft sind. Im Gegensatz zum standardisierten ProFormA-Format kann der Editor nicht im Voraus wissen, wie Fremdformatelemente definiert sind. Für die Darstellung solcher Elemente weicht der Editor deshalb auf eine generische Darstellung aus. Elemente werden dazu dynamisch über die Document Object Model-Schnittstelle (DOM) geladen und als Baumstruktur angezeigt (Abb. 3). Da Elemente in der Strukturlänge und -tiefe sehr groß werden können, ist es möglich, einzelne Strukturteile ein- und auszublenden, damit Benutzer sich selektiv auf den für sie relevanten Teil konzentrieren können. Für die bloße Darstellung genügen unserem Werkzeug die in einem Element gespeicherten Informationen. Für das Editieren von Fremdformatelemen-

ten greift unser Werkzeug auf die zugehörigen XSD-Dateien zurück, die vom Nutzer zu diesem Zweck vorher in das Arbeitsverzeichnis des Editors gelegt wurden (R2). Der Editor wertet die Typdefinitionen von Fremdformatelementen in einer task aus und erzeugt eine dynamische Editorkomponente, die nur solche Änderungen an der Struktur und an dem Wert eines Elementes erlaubt, die den Vorschriften der Typdefinition genügen (Abb. 5). Solche Änderungen können u. a. die Kindelemente und -attribute eines Elternelementes betreffen, sowie minimale und maximale Häufigkeit, in der Kindelemente auftreten dürfen. Dialogelemente passt der Editor an die Typdefinition an. Beispielsweise präsentiert der Editor dem Benutzer beim Datentyp boolean eine Checkbox statt eines Texteingabefeldes. In Kombination mit einer XML-Validierung garantiert dieses Verfahren, dass bei einem Speichervorgang einer task eine gültige XML-Datei in Hinsicht auf die in der task verwendeten Fremdformate erzeugt wird (R5). Mit Kenntnis der XSD-Dateien kann der Editor zudem entscheiden, welche Fremdformatelemente in eine task hinzugefügt und entfernt werden können (R3). Als besonderen Service für den Benutzer legt der Editor ein neues Fremdformatelement immer gleich mit allen lt. XSD notwendigen Attributen und Kindelementen an. Optionale Attribute und Kindelemente kann der Benutzer leicht über das Kontextmenü hinzufügen.

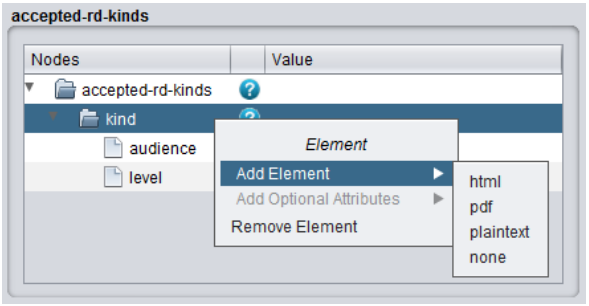

Abb. 5: Kindelement hinzufügen

Die generische Editorkomponente bietet potentiell für alle Fremdformatelemente abrufbare Hilfetexte an (R6). Hilfestellungen sind für Elemente, die generisch gehandhabt werden, besonders nützlich, weil hier nach jeder Möglichkeit gesucht werden muss, die Bedienbarkeit eines generischen Editors zu verbessern. Autoren von Schemadefinitionen werden ermutigt, zusätzliche Hilfetexte bereitzustellen, die der Editor aus dem Schema extrahiert und für einzelne Elemente auf Benutzeranforderung anzeigt. Ermöglicht wird dies durch die XML-Elemente *xs:annotation* und *xs:documentation*.

Die generische Handhabung von Fremdformatelementen führt jedoch zwangsläufig zu einer geringeren Benutzerfreundlichkeit. Da Elemente, Kindelemente und -attribute in einer generischen Ansicht auf die gleiche Weise dargestellt werden (Abb. 3), können bspw. besonders relevante Strukturteile nicht visuell hervorgehoben werden, da die Information, welche Daten besonders wichtig sind, nicht über eine Schemadefinition bereitgestellt werden kann. Des Weiteren werden die Bezeichner von Elementen und Attributen direkt aus der Schemadefinition entnommen. Benutzer werden oftmals kryptischen

Namensbezeichnern ausgesetzt, die von Schemaautoren bei der Namensgebung für Elemente und Attribute verwendet werden.

6 Plugin-System

Die mit der generischen Handhabung einhergehende mangelnde Benutzerfreundlichkeit kann durch eine Anbindung von speziellen Plugins aufgehoben werden (R4). Plugins enthalten handgefertigte Benutzerschnittstellen, die sich auf die Darstellung und das Editieren von Elementen aus Fremdformaten spezialisieren. Kryptische Elementnamen können durch entsprechende Bezeichner ersetzt werden. Eine Internationalisierung des Plugins ermöglicht, Bezeichner in verschiedene Sprachen zu übersetzen. Besonders relevante Elementstrukturteile können hier hervorgehoben dargestellt werden.

Bei der generischen Handhabung ist die Anzeige grundsätzlich auf die in einem Element gespeicherten Informationen beschränkt. Durch einen sorgfältigen Entwurf der Plugin-Schnittstelle unseres Editors werden Pluginentwickler in die Lage versetzt, für jedes Fremdformatelement kontextuell relevante Informationen aus allen anderen Teilen einer task hinzuzuziehen und anzuzeigen.

In Abb. 3 ist die generische Ansicht eines Elementes namens *test-group* abgebildet, das die Middleware Grappa [GHW15] verwendet, um Punktzahlen für automatisierte Prüfungen festzulegen. In Abb. 4 wird eine maßgeschneiderte Plugin-Benutzerschnittstelle für dasselbe Element gegenübergestellt. Das inhaltliche Verständnis für die Funktionsweise des Elements ist in Abb. 3 trotz angebotener Hilfestellung zu einzelnen Zweigen der Elementstruktur nicht sofort ersichtlich. Durch eine intelligente Anordnung und Gruppierung einzelner grafischer Dialogelemente lässt sich die Verständlichkeit deutlich erhöhen. Kontextuell relevante Informationen aus anderen Teilen der task erhöhen die Verständlichkeit des Dialogs – hier etwa die Überschrift „Wartbarkeit“ oder der in Klammern angegebene Titel der *Test Reference* „id003“ (Wartbarkeit). Um den Preis des zusätzlichen Entwicklungsaufwandes für ein Plugin entsteht eine intuitive Benutzerschnittstelle für ein andernfalls schwer durchschaubares XML-Element.

Vorteilhaft ist, dass inhaltliche Prüfungen in die Benutzerschnittstellen einprogrammiert werden können, die über eine XML-Validierung gegen eine Schemadefinition nicht umsetzbar sind. Da Java für die Pluginentwicklung verwendet wird, sind hier prinzipiell keine Grenzen gesetzt. Betrachtet man bspw. die Schnittstelle aus Abb. 4, wird ersichtlich, dass die Punktzahl eines Elternelementes die Summe der einzelnen Punktzahlen von Kindelementen bildet. Eine zusätzliche Prüfung kann sicherstellen, dass nicht versehentlich die falsche Summe in ein Elternelement eingetragen wird. Genauso gut könnte man das Plugin so implementieren, dass die Summe eines Elternelementes automatisch aus der Eingabe der Punkte für die Kindelemente ermittelt wird.

Ein XML-Element wird über den Elementnamen und den Uniform Resource Identifier (URI) des Namensraums identifiziert, dem es angehört. Eine Pluginbenutzerschnittstelle

bezieht sich immer auf genau ein Element inklusive aller Kind- und Kindeskindelemente. Die Beziehung zwischen der Benutzerschnittstelle und dem Element wird über Elementname und Namensraum-URI bei der Erstellung des Plugins definiert. Plugins werden in Form von .jar-Dateien in das Arbeitsverzeichnis des Editors gelegt. Bei Bedarf lädt der Editor die entsprechenden Plugins für Fremdformatelemente und bettet sie auf seiner Benutzeroberfläche mit ein.

Zur Identifizierung verschiedener Versionen einer Schemadefinition enthält die Namensraum-URI in der Regel eine Versionsnummer. Ändert sich die URI, so ändern sich in der Regel auch die definierten Elemente. Derartige Änderungen ziehen ggf. Änderungen am zugehörigen Plugin und den durch das Plugin realisierten Dialogelementen nach sich.

Für weitere Details der Plugin-Schnittstelle verweisen wir aus Platzgründen auf [Rei17].

7 Zusammenfassung und Ausblick

In diesem Beitrag wurde ein Editor vorgestellt, mit dem Autoren und Dozenten automatisch bewertbare Aufgaben erstellen und anpassen können. Der Editor arbeitet dabei mit dem ProFormA-Format, welches ein einheitliches Aufgabenformat unabhängig vom verwendeten Grader definiert.

Das ProFormA-Format ist flexibel erweiterbar, um spezielle Aspekte für bestimmte Zielsprachen oder Systeme unterstützen zu können. Diese Erweiterbarkeit stellt eine Herausforderung für die Oberfläche dar, die einerseits generisch alle Teile des Formats unterstützen muss, andererseits aber eine verständliche und an die jeweiligen Inhalte angepasste Benutzerführung bieten soll. Ein generischer Editor, ähnlich wie allgemeine XML-Editoren, kann jede mögliche Erweiterung unterstützen. Allerdings leidet die Benutzerfreundlichkeit durch den generischen Ansatz. Auch die Möglichkeiten komplexer Eingabevalidierung bzw. -unterstützung sind limitiert. Ein über das XML-Schema realisiertes Hilfesystem schafft hier nur teilweise Abhilfe. Daher bietet der hier vorgestellte Editor neben der generischen Oberfläche die Möglichkeit an, für bestimmte Erweiterungen per Plugin spezifische Oberflächenelemente zu integrieren.

Der Editor ist als erste Beta-Version mit der generischen Komponente und dem Plugin-System sowie einem exemplarischen Plugin funktionstüchtig, unterstützt allerdings derzeit nur das Editieren von ausgepackten Aufgaben. Als nächste Schritte sind die Unterstützung des Erstellens und Editierens von ZIP-Paketen sowie der Aufruf aus einem LMS über Web-Start geplant. Interessant könnte auch eine direkte Kopplung mit den verwendeten Gradern über Grappa [GHW15] sein, um Aufgaben aus dem Editor heraus direkt testen zu können.

Literaturverzeichnis

[St15] Strickroth, S. et al.: ProFormA: An XML-based exchange format for programming tasks. eleed e-learning & education, 11(1), 2015.

[PRB17] Priss, U.; Rod, O.; Borm, K.: ProFormA/formatEditor: A Javascript editor for the exchange format for programming exercises, https://github.com/ProFormA/formatEditor, 6.04.2017.

[Rei17] Reiser, P.: Entwicklung eines generischen XML-Editors für ein interoperables Programmieraufgabenformat. Bachelorarbeit, Hochschule Hannover, http://nbn-resolving.de/urn:nbn:de:bsz:960-opus4-10866, 19.06.2017.

[GHW15] Garmann, R.; Heine, F.; Werner, P.: Grappa – Die Spinne im Netz der Autobewerter und Lernmanagementsysteme. In: DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik. Bd. 247, LNI, GI, 169-181, 2015.