Wiederverwendbarkeit von Programmieraufgaben durch Interoperabilität von Programmierlernsystemen

Sven Strickroth[1], Michael Striewe[2], Oliver Müller[3],
Uta Priss[4], Sebastian Becker[5], Oliver J. Bott[5], Niels Pinkwart[1]

[1] Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin
{sven.strickroth,pinkwart}@hu-berlin.de

[2] Paluno – The Ruhr Institute for Software Technology
Universität Duisburg-Essen
Gerlingstraße 16
45127 Essen
michael.striewe@paluno.uni-due.de

[3] Institut für Informatik
Technische Universität Clausthal
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld
oliver.mueller@tu-clausthal.de

[4] ZeLL – Zentrum für erfolgreiches Lehren und Lernen
Ostfalia Hochschule für angewandte Wissenschaften
Am Exer 2
38302 Wolfenbüttel
u.priss@ostfalia.de

[5] ZSW – E-Learning Center
Hochschule Hannover
Expo Plaza 12
30539 Hannover
sebastian.becker@hs-hannover.de
oliver.bott@hs-hannover.de

Abstract: Unterstützungssysteme für die Programmierausbildung sind weit verbreitet, doch gängige Standards für den Austausch von allgemeinen (Lern-) Inhalten und Tests erfüllen nicht die speziellen Anforderungen von Programmieraufgaben wie z. B. den Umgang mit komplexen Einreichungen aus mehreren Dateien oder die Kombination verschiedener (automatischer) Bewertungsverfahren. Dadurch können Aufgaben nicht zwischen Systemen ausgetauscht werden, was aufgrund des hohen Aufwands für die Entwicklung guter Aufgaben jedoch wünschenswert wäre. In diesem Beitrag wird ein erweiterbares XML-basiertes Format zum Austausch von Programmieraufgaben vorgestellt, das bereits von mehreren Systemen prototypisch genutzt wird und das mittelfristig den Austausch über ein gemeinsam genutztes Aufgabenrepository ermöglichen soll.

1 Einleitung und Problemstellung

Der Bedarf für Lehrende, Aufgaben für den Einsatz in verschiedenen Lernszenarien aus einem Pool von bestehenden Testinhalten zu nutzen, wächst mit steigendem E-Assessment Einsatz an Hochschulen. Dies gilt aufgrund des vergleichsweise hohen Entwicklungsaufwands im besonderen Maße für Programmieraufgaben. Dabei handelt es sich um offene Aufgabentypen, da es wichtig ist, dass die Lernenden eigene Lösungen

entwickeln und diese geprüft werden können [Ro07]. Neben der textuellen Beschreibung der jeweiligen Programmieraufgabe sind zur Vorbereitung der (automatisierten) Bewertung oder Generierung von Feedback in der Regel abhängig vom Programmbewertungssystem Tests, wie im Falle der Programmiersprache Java z. B. JUnit-Tests, zu implementieren oder auch im Falle von z. B. SQL oder Prolog Musterlösungen vorzubereiten. Der erforderliche, partiell sehr hohe Aufwand bei der Erstellung derartiger Aufgaben kann deutlich reduziert werden, wenn Testinhalte systemunabhängig genutzt und zwischen den Lehrenden ausgetauscht werden können. Obwohl seit deutlich mehr als zehn Jahren an Unterstützungssystemen für die Programmierausbildung bzw. (automatischen) Bewertungssystemen für unterschiedliche Programmiersprachen gearbeitet wird [KSZ02, Ed04, Hü05, Sp06, Mo07, Tr07, ASS08, HQW08, SBG09, Ih10, SOP11, PJR12, RRH12], gibt es bislang kein gemeinsames Austauschformat.

Trotz des großen Aufwands für die Konzeption von (Übungs-)Aufgaben besitzen nur wenige Systeme, die in der Programmierausbildung eingesetzt werden (z. B. DUESIE, eAIXESSOR, JACK), einen auf Wiederverwendung ausgerichteten Aufgabenpool oder streben diesen an. Insbesondere bei Systemen, von denen mehrere Instanzen existieren, könnte zumindest eine Import/Export-Funktion innerhalb dieser Systeme den Arbeitsaufwand durch Wiederverwendung von Aufgaben reduzieren. Dennoch ist ein Im- bzw. Export von Aufgaben bislang nur selten möglich (JACK, Praktomat). In anderen Bereichen (z. B. Mathematik) gibt es hingegen Learning Management Systeme (LMS, z. B. LON-CAPA), bei denen der (weltweite) Austausch einen zentralen Systembestandteil darstellt. Im Folgenden wird ein Ansatz für ein gemeinsames Austauschformat vorgestellt, das genau diese Lücke für Programmieraufgaben systemübergreifend schließen soll.

2 Anforderungen an ein Austauschformat und existierende Formate

In diesem Abschnitt werden anhand von im Einsatz befindlichen Systemen Anforderungen an ein systemübergreifendes Austauschformat hergeleitet, existierende Austauschformate vorgestellt und mit den hergeleiteten Anforderungen verglichen.

2.1 Anforderungen an ein Austauschformat für Programmieraufgaben

Programmieraufgaben können in sehr verschiedener Form und sehr unterschiedlichem Umfang gestellt werden: Im einfachsten Fall müssen möglicherweise nur wenige Zeilen Programmcode in einem vorgegebenen Codegerüst ergänzt werden, während in komplexen Fällen zahlreiche Klassen zu erstellen sind, die vorgegebene Schnittstellen realisieren oder selber vorgegebene Schnittstellen ansprechen. Zudem kann es je nach Programmiersprache und Entwicklungsumgebung Verzeichnis- und Dateistrukturen geben, die von den Lernenden beachtet werden müssen.

Anders als bei geschlossenen Fragetypen, bei denen die korrekten Antworten vorab bekannt sind und die Korrektheit einer Einreichung über einen Soll-Ist-Vergleich bestimmt werden kann, gehören Programmieraufgaben, sofern es sich nicht um sehr einfache Fill-in-the-Gap-Aufgaben handelt, zu den offenen Fragetypen, bei denen eine

Einreichung nach verschiedenen Kriterien beurteilt werden kann. Hier kann beispielsweise eine statische Analyse des Quellcodes erfolgen (mit verschiedenen Zielen, z. B. Kompilierfähigkeit, Verwendung bestimmter Konstrukte, Einhaltung eines Programmierstils). Zudem sind dynamische Analysen (z. B. Unit- oder Blackbox-Tests), mit denen die Korrektheit eines eingereichten Programms überprüft werden kann, denkbar [Al05]. Für die Durchführung derartiger Analysen werden möglicherweise verschiedene Ausführungsumgebungen oder zusätzliche Dateien benötigt, die für die Lernenden nicht sichtbar sind.

Zahlreiche existierende Systeme setzen meist unterschiedliche Schwerpunkte in ihren didaktischen Zielsetzungen, bieten oft nur die dafür nötigen Funktionen an und haben dementsprechend unterschiedliche Anforderungen an die Spezifikation einer Aufgabe. Ein sinnvolles Austauschformat für Aufgaben muss daher die Obermenge aller dieser Anforderungen abdecken können. Tabelle 1 gibt eine Übersicht über 11 Systeme und die sich aus ihren Funktionen ergebenden Anforderungen. Demnach muss spezifiziert werden können,

(A1) welche Codevorlagen den Lernenden zum Download zur Bearbeitung angeboten oder als Fill-in-the-Gap-Aufgaben o. ä. in einem Web-Interface präsentiert werden;
(A2) welche Dateien den Lernenden als Referenz zur Verfügung gestellt werden, ohne dass diese verändert eingereicht werden sollen (z. B. Bibliotheken);
(A3) in welcher Form und Struktur (einzelne Dateien, ZIP-Archiv, usw.) Dateien hoch- und heruntergeladen werden können;
(A4) welche zusätzlichen Dateien zu einer Aufgabe gehören (z. B. Testtreiber);
(A5) welche Prüfverfahren auf die eingereichte Lösung angewandt werden sollen und unter welchen Bedingungen diese durchgeführt werden;
(A6) welche zur Aufgabe bzw. Einreichung gehörigen Dateien welchem Prüfverfahren zur Verfügung gestellt werden;
(A7) welche Metainformationen (Titel, Aufgabenstellung, Bewertungsskala, u. ä.) mit einer Aufgabe verknüpft sind;
(A8) wie die Musterlösung der Aufgabe aussieht.

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**
ASB [Mo07]			x	x	x		x	
DUESIE [HQW08]				x	x		x	
eAixessor [ASS08]				x	x	x	x	
eClaus [WW07]				x	x		x	x
ELP [Tr07]	x						x	x
GATE [SOP11]	x	x	x	x	x	x	x	
JACK [SBG09]	x	x	x	x	x	x	x	
Marmoset [Sp06]				x			x	
Praktomat [KSZ02]			(x)	x	x	x	x	(x)
ViPS [Hü05]				x			x	x
Web-CAT [Ed04]				(x)		(x)	x	x

Tabelle 1: Verschiedene Systeme für die Bewertung von Programmieraufgaben und ihre Anforderungen an die Spezifikation der Aufgaben. Nicht ausgefüllte Zellen bedeuten, dass keine Informationen darüber vorliegen, ob das System diese Anforderung stellt. Geklammerte Kreuze deuten an, dass diese Anforderung versionsabhängig ist.

Die Anforderungen zerfallen dabei in drei Gruppen: Die erste Gruppe enthält mit A1 bis A3 drei Anforderungen, die das Dateimanagement zwischen System und Lernenden betreffen. Hier stellen eher wenige Systeme besondere Anforderungen, während die meisten das Hochladen beliebiger Dateien erlauben, die zuvor nicht als Template bereitgestellt wurden. Die zweite Gruppe enthält die Anforderungen A4 und A7, die nahezu von jedem System gestellt werden. Da es sich dabei um die textliche Aufgabenstellung und die Konfiguration der automatischen Tests handelt, ist dies wenig überraschend. ELP und Web-CAT stellen bezüglich A4 Ausnahmen dar, da sie Tests auf Basis von Musterlösungen generieren bzw. nach dem Prinzip der testgetriebenen Entwicklung von den Lernenden selbst eingereichte Tests verwenden. In die dritte Gruppe von Anforderungen fallen schließlich A5, A6 und A8, die in mäßig vielen Systemen vertreten sind. Hierbei handelt es sich um Anforderungen, die meist als optional anzusehen sind. Viele, aber nicht alle Systeme bieten Prüfverfahren optional an, so dass entsprechende Konfigurationen benötigt werden. Die beschränkte Verfügbarkeit von Dateien für einzelne Prüfverfahren dient dabei insbesondere der Schonung von Systemressourcen, indem nicht benötigte Dateien erst gar nicht an Prüfkomponenten übertragen werden müssen. Musterlösungen werden nur in wenigen Fällen zur Generierung der Tests benötigt (s. o.) und dienen ansonsten dem erweiterten Feedback an Lernende.

2.2 Bestehende Austauschformate

Das Format für Online-Lernmaterialien mit der größten Verbreitung ist das IMS Content Packaging (IMS CP) Format[1] des IMC Global Learning Consortiums [LQ09]. Dieses Format spezifiziert, wie Lernmaterialien zu einem Paket (ZIP-Datei inkl. Manifest) kombiniert werden. Aufbauend auf diesem Format gibt es die Standards „Sharable Content Object Reference Model“ (SCORM[2]) und „Question and Test Interoperability Specification“ (QTI[3]), eine Spezifikationen von Cesarini u.a. [CMM04] sowie zwei Spezifikationen von Leal und Queirós [LQ09, QL11] speziell für Programmieraufgaben.

Das SCORM Referenzmodell beschränkt sich auf austauschbare elektronische Lerninhalte und spezifiziert konkret eine Laufzeitumgebung für LMS (RTE), Inhaltsaggregation und Navigations- sowie Reihenfolgen für die Präsentation. Die RTE besitzt lediglich Get- und Set-Methoden für einfache Variablen zwecks Kommunikation der Lernressourcen mit dem LMS (z. B. zum Speichern von Lernfortschritten) und unterstützt folglich die formulierten Anforderungen aus Abschnitt 2.1 nicht. QTI hingegen ermöglicht den Austausch und die automatisierte Auswertung von geschlossenen Fragetypen. Halboffene Fragetypen (d. h. Freitextantworten) können modelliert werden. Für diese ist jedoch keine automatisierte Evaluation vorgesehen. Insbesondere ist das Format nicht auf Programmieraufgaben ausgelegt und erfüllt daher nur (partiell) die Anforderungen A1 und A7.

Cesarini u.a. [CMM04] entwickelten eine Spezifikation für Lernobjekte (auch für Programmieraufgaben), die von verschiedenen sog. „Test-Engines“ überprüft werden

[1] http://www.imsglobal.org/content/packaging/, zuletzt abgerufen am 13.03.2014
[2] http://scorm.com/scorm-explained/technical-scorm/, zuletzt abgerufen am 13.03.2014
[3] http://www.imsglobal.org/question/, zuletzt abgerufen am 13.03.2014

können (erfüllt A5 und A7 partiell). Jedoch erlaubt die Spezifikation nur jeweils einen Test pro Lernobjekt und keine automatisierte Evaluation von Programmieraufgaben.

Der erste Ansatz von Leal und Queirós [LQ09] wurde im Rahmen eines EU-Projektes entwickelt und erlaubt es, Programmieraufgaben mit Tests als Lernobjekte, genauer gesagt als IMS CP Metadaten, zu speichern. Das Datenmodell ist jedoch nur für eine einzige, vorgegebene Evaluations-Engine ausgelegt (EduJudge/Mooshak, [Ve11]) und unterstützt die Anforderungen A2, A4, A5 und A7 lediglich partiell. Die Projekt-Webseite ist nicht mehr erreichbar und über das Format bzw. den Einsatz sind keine aktuellen Informationen erhältlich.

Der zweite Ansatz von Queirós und Leal [QL11] spezifiziert die „Programming Exercises Interoperability Language“ (PExIL). Das Ziel dieser Spezifikation ist die Abdeckung des gesamten Lebenszyklus von Programmieraufgaben – von der Aufgabenerstellung bis zum Feedback. Mit Hilfe von PExIL können Aufgaben und Tests (inkl. Befehle für Kompilierung und Ausführung) für mehrere, fest vorgegebene, (imperative) Programmiersprachen modelliert werden (lediglich die Anforderungen A1, A4, A5, A7 und A8 werden erfüllt). Tests bzw. Eingabedaten werden direkt in XML kodiert und sind somit von einer speziellen Evaluations-Engine abhängig, die nur Blackbox-Tests zu unterstützen scheint. Ferner scheint die Spezifikation nur Aufgaben vorzusehen, die aus einer einzigen Lösungsdatei bestehen. Als Besonderheit sei angemerkt, dass dieses Format auch Feedback und Bedingungen, wann ein bestimmtes Feedback angezeigt werden soll (z. B. ab drittem Versuch), modellieren kann. Ein XML-Schema ist vorhanden, jedoch unzureichend dokumentiert. Das einzige bekannte System, das diese Spezifikation unterstützt, ist PETCHA [QL12] von den Autoren dieser Spezifikation.

Unabhängig von IMS CP wurde das Peach Exchange Format [Ve08] für „Programming-Contest“-Systeme entwickelt. Dieses Format hat eine starke Abhängigkeit vom Peach-System und unterstützt zwar unterschiedliche Programmiersprachen für eine Aufgabe, aber nur reine Blackbox-Tests (Eingabe/Ausgabe-Tests, Skript-gesteuert). Die Anforderungen A1, A2, A3, A6 und A8 werden nicht bzw. nur teilweise erfüllt.

Neben diesen explizit für den systemübergreifenden Austausch vorgesehenen Formaten besitzen einige Systeme die Möglichkeit, Aufgaben oder Tests in einem XML-Format zu definieren. Diese richten sich jedoch ausschließlich nach den Anforderungen „ihrer“ Systeme. Im ELP-System [Tr07] können Fill-in-the-Gap Aufgaben für Java und C++ inkl. Hinweisen und Lösung spezifiziert werden. Ein XML DTD oder Schema ist nicht verfügbar. Kein Export von Aufgaben, aber eine XML-Spezifikation für Java-Blackbox-Tests, ist bei eClaus zu finden [WW07]. Damit müssen Tests zum Beispiel nicht als JUnit-Test programmiert, sondern können in XML abgebildet werden. JACK und die aktuelle Praktomat-Version erlauben den Export sowie Import von Aufgaben in einem eigenen XML-Format (vgl. Tabelle 1).

Folglich fehlt ein universelles Aufgabenformat für Programmieraufgaben, welches die im Abschnitt 2.1 genannten Anforderungen erfüllt und auch systemübergreifend eingesetzt werden kann.

3 Ein XML-Austauschformat für Programmieraufgaben

Das vorgeschlagene Austauschformat[4] spezifiziert für eine Aufgabe (*task*) einen Aufgabentext (inkl. Festlegung der Sprache zwecks Internationalisierung), die Programmiersprache, zugehörige Dateien, technische Details zur Einreichung, Lösungsvorschläge und optional Hinweise zur Bewertung und Metadaten. Zu einer Aufgabe können mehrere Tests gehören, welche jeweils einen Test-Typ (konkretes Bewertungsverfahren) und eine spezifische Testkonfiguration beinhalten. Tests werden in der Regel durch übliche Software Engineering Werkzeuge (Compiler, Unit-Tests, FindBugs, CheckStyle usw.) ausgeführt. Das Austauschformat selbst muss also im Wesentlichen die Bedingungen für die Ausführbarkeit der Tests festlegen, nicht aber den eigentlichen Testinhalt, welcher im Format der benutzten Werkzeuge gespeichert wird (z. B. skript-basierte Blackbox-Tests oder Konfigurationen für Werkzeuge). Ein importierendes System kann folglich anhand der Test-Typen entscheiden, welche Tests es (direkt) unterstützt und wie diese ggf. ausgeführt werden.

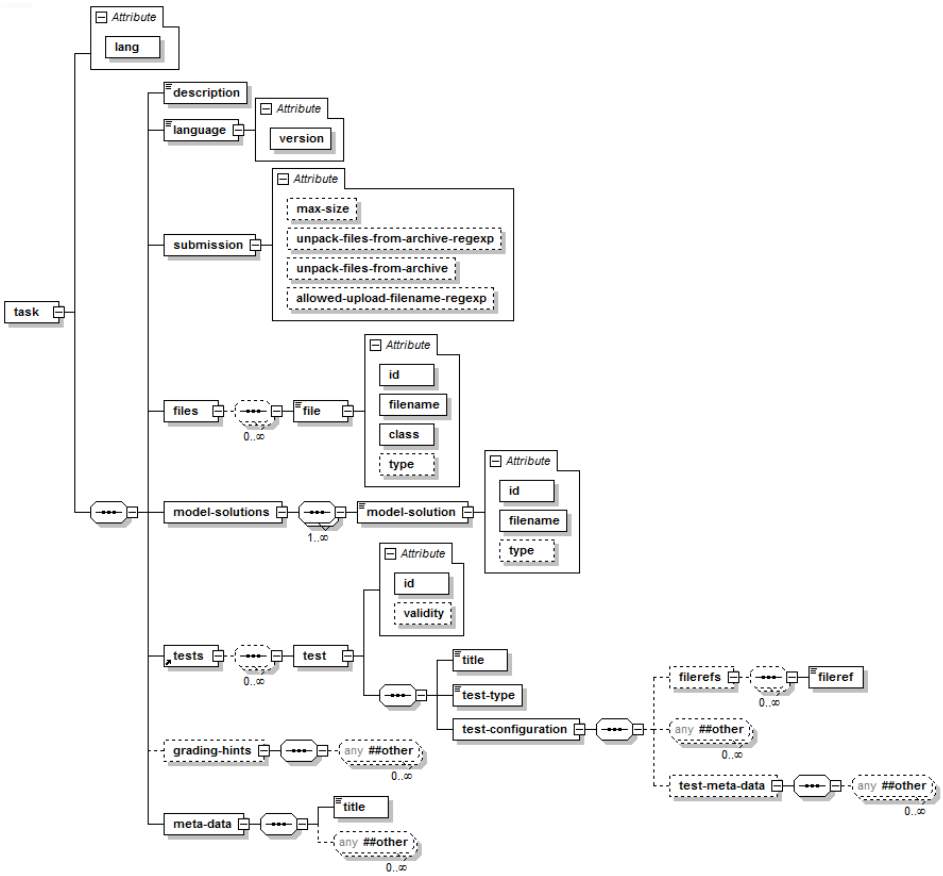

Abbildung 1: Strukturbaum des Austauschformates

[4] http://teach.ecult.me/proforma/

Abbildung 1 zeigt die Struktur des Austauschformates. Dabei wird die Schachtelung der definierten Elemente und Attribute ersichtlich. Elemente, bei denen in der linken oberen Ecke drei stilisierte Linien zu sehen sind, können direkt mit Inhalten gefüllt werden (z. B. */task/description*). Optionale Elemente sind mit einer gestrichelten Linie umgeben dargestellt (z. B. */task/grading-hints*). XML-Sequenzen werden durch achteckige Kästen symbolisiert, wobei die Kardinalitäten direkt darunter angegeben sind (sofern diese von 1 abweichen). Insbesondere sollen hier auch die Boxen mit „any ##other“ erwähnt werden: Das vorgeschlagene Austauschformat definiert lediglich das Grundgerüst, den gemeinsamen Nenner für Programmieraufgaben, und soll gleichzeitig zum Beispiel für neue Programmiersprachen, Testverfahren und besondere Fähigkeiten von implementierenden Systemen erweiterbar sein. Zu diesem Zweck sind ähnlich zu Programmierframeworks „Hotspots“ vorgesehen, an denen weitere Elemente aus anderen XML-Namespaces importiert und genutzt werden können – somit bleibt das Format für systemspezifische Konfigurationen und (neue) Bewertungsverfahren flexibel lokal erweiterbar. Zudem ist die vorgeschlagene Spezifikation versioniert (über die Namespace URI), um spätere Ergänzungen bzw. Änderungen der Spezifikation zu erlauben.

Zum besseren Verständnis wird im Folgenden genauer auf einige ausgewählte Elemente und auf die Erfüllung der Anforderungen aus Abschnitt 2.1 eingegangen. Listing 1 zeigt dazu ein konkretes Minimalbeispiel einer Aufgabe im vorgeschlagenen Format, wie sie in einem typischen Einführungskurs in die Java-Programmierung zum Thema Rekursion oder Schleifen vorkommen kann: Die Berechnung der n-ten Fibonacci-Zahl.

Dateianhänge können entweder direkt in das XML-Dokument eingebunden oder über die Angabe des entsprechenden Dateinamens in einem ZIP-Archiv referenziert werden (im letzteren Fall muss sich das XML-Dokument als *task.xml* im Wurzelverzeichnis befinden). Für jede Datei wird ein *file* Element unterhalb des */task/files* Elements eingefügt. Für jede Datei muss eine eindeutige ID zur späteren Referenzierbarkeit (z. B. bei der Testdefinition; A6) innerhalb des XML-Dokuments und eine Klasse (Attribut *class*) angegeben werden, die den Zweck und die Zugangsrechte beschreibt (A1, A2, A4). Vorgesehen sind hier (Code-)Vorlagen (*template*), Bibliotheken (*library*), Eingabedaten (*inputdata*) zusätzliche Informationen bzw. Anweisungen zur Bearbeitung einer Aufgabe (*instruction*) und interne Dateien (z. B. Testtreiber, *internal)*. „Intern“ bezieht sich dabei auf die Einschränkung, dass diese Dateien für Lernende grundsätzlich nicht zugänglich sind. Ob eine Datei direkt eingebunden ist oder im ZIP-Archiv referenziert wird, wird durch das *type* Attribut festgelegt. Für eine Aufgabe muss des Weiteren mindestens eine Musterlösung hinterlegt werden – diese werden unterhalb des Elements */task/model-solutions* separat behandelt (A8). Ein *model-solution* Element steht für eine konkrete Musterlösung. Diese kann aus einer Menge an Dateien bestehen (z. B. eine Musterlösung einer Java-Programmieraufgabe, bestehend aus einer Menge .java-Dateien). Die Einbettung erfolgt analog zum *file* Element.

Anzuwendende Tests bzw. Bewertungsverfahren werden unterhalb des Elements */task/tests* in *test* Elementen definiert (A5). Neben der Angabe, welche Arten von Tests für eine Aufgabe zur Verfügung stehen (z. B. Compile/Syntax- oder JUnit-Tests, Element *test/test-type*) und der Angabe eines Titels (Element *test/title*), lässt sich die genaue Konfiguration eines Tests beschreiben (*test/test-configuration*). Zur Konfiguration eines

```
<task lang="de" xmlns="urn:proforma:task:v0.9.1">
  <description>Berechnen Sie die n-te Fibonacci-Zahl. Zur Erinnerung: die
  ersten beiden Fibonacci-Zahlen sind 0 und 1. Die darauffolgenden sind
  die Summe der vorherigen beiden Zahlen (0, 1, 1, 2, 3, 5, 8, 13, 21...).
  Die Klasse soll Fibonacci heißen. Die zu schreibende Methode fibonacci
  hat als Eingabeparameter einen int.</description>
  <language version="1.4">java</language>
  <submission allowed-upload-filename-regexp="Fibonacci\.java" />
  <files>
    <file class="internal" filename="FibonacciTest.java" id="1">
      import junit.framework.TestCase;
      public class FibonacciTest extends TestCase {
        public void testPos() { assertEquals(13, Fibonacci.fibonacci(7)); }
        public void testNull() { assertEquals(0, Fibonacci.fibonacci(0)); }
      }
    </file>
  </files>
  <model-solutions>
    <model-solution id="1" filename="Fibonacci.java">
      public class Fibonacci {
        public int fibonacci(int i) {
          if (i &lt;= 0) return 0;
          else if (i == 1) return 1;
          return fibonacci(i - 2) + fibonacci(i - 1);
        }
      }
    </model-solution>
  </model-solutions>
  <tests>
    <test id="1">
      <title>Kompilierbarkeit</title>
      <test-type>java-compilation</test-type>
      <test-configuration />
    </test>
    <test id="2">
      <title>Berechnungstest</title>
      <test-type>java-junittest3</test-type>
      <test-configuration xmlns:ju="urn:proforma:tests:junit3:v0.1">
        <filerefs>
          <fileref>1</fileref>
        </filerefs>
        <ju:junit3test>
          <mainclass>FibonacciTest</mainclass>
        </ju:junit3test>
      </test-configuration>
    </test>
  </tests>
</task>
```

Listing 1: Beispielinstanz einer Programmieraufgabe

Tests gehört, welche weiteren Dateien (z. B. Testtreiber) benötigt werden. Über IDs in *filerefs/fileref* Elementen können dazu definierte *file* Elemente referenziert werden (A6). Darüber hinaus ist direkt unterhalb des *test-configuration* Elements vorgesehen, dass dort für jeden Test-Typ ein eigener XML-Namespace genutzt werden kann, um test-spezifische Parameter austauschen zu können (z. B. den Namen der aufzurufenden Testklasse für JUnit-Tests, vgl. Listing 1). Ferner ist vorgesehen, dass unterhalb des *test-configuration/test-meta-data* Elements über die Nutzung eines eigenen XML-Name-

spaces systemspezifische Metadaten (z. B. Beschränkungen für Tests) für einen Test hinterlegt werden können.

Das */task/submission* Element bietet die Möglichkeit, Einschränkungen für die Einreichung von Lösungen zu definieren. Dabei kann z. B. die maximale Größe (in Byte) einer zur Abgabe der Lösung im System hochzuladenden Datei festgelegt werden (Attribut *max-size*). Bedingungen an Dateinamen sind über reguläre Ausdrücke spezifizierbar (Attribut *allowed-upload-filename-regexp*). Dies ist insbesondere für Java-Aufgaben relevant, da dort abhängig vom Klassennamen Anforderungen an den Dateinamen gestellt werden. Möchte ein Lernender eine Lösung einer Java-Aufgabe in Form eines Dateiuploads innerhalb eines Systems einreichen, so kann zunächst vom System verifiziert werden, ob der Name der hochzuladenden Datei sämtliche Bedingungen des regulären Ausdrucks erfüllt. Ist dies nicht der Fall, so kann die Datei nicht hochgeladen und/oder eine Warnung gezeigt werden. Somit kann u. a. sichergestellt werden, dass Tests nicht aufgrund eines falschen Dateinamens fehlschlagen (A3).

Eine Instanz des *description* Elements enthält die Aufgabenstellung. Für eine Aufgabe lassen sich außerdem ein Bewertungsschema (Element */task/grading-hints*) sowie weitere Metadaten (Element */task/meta-data*) angeben. Analog zum *test-meta-data* Element kann im */task/grading-hints* Element ein eigener XML-Namespace für systemindividuelle Angaben zu Bewertungsschemata benutzt werden. Dadurch können aufgaben- bzw. testbezogene Attribute spezifiziert werden, die für ein bestimmtes System benötigt werden, für andere Systeme aber nicht relevant sind. Systemindividuelle Angaben gehen dadurch beim Export nicht verloren. Ein vollständiger Export einer Aufgabe in das Austauschformat mit anschließendem verlustfreien Reimport in dasselbe System ist somit prinzipiell möglich. Wird eine aus einem System exportierte Aufgabe in ein anderes System importiert, so ist sichergestellt, dass nur relevante Daten importiert werden, die auch vom letztgenannten System berücksichtigt und verarbeitet werden können (A7).

4 Einsatz des Austauschformats und Diskussion

Die vorgeschlagene Format-Spezifikation wird bereits seit ca. einem Jahr von drei Systemen (GATE, JACK und einer Praktomat-Version) zum Export unterstützt (hauptsächlich für Java-Aufgaben) – ferner wurden beim Entwurf auch Anforderungen weiterer Systeme (u. a. ViPS mit Prolog, LON-CAPA Externalresponse[5]) beachtet. Somit ist sichergestellt, dass das Format grundsätzlich nicht nur die Anforderungen eines einzelnen Systems und lediglich die dort unterstützten Programmiersprachen implementiert, wie es für die meisten bestehenden Formate der Fall ist. Zudem wurde die vorgeschlagene Spezifikation nicht als fixes „one-size-fits-all“ Format angesehen und systemspezifische Aspekte nicht ignoriert, sondern Erweiterungen (einzelner Systeme oder neue Testverfahren) können über eigene, frei definierbare XML-Namespaces eingebettet werden. Eine Erweiterbarkeit über XML-Namespaces birgt jedoch grundsätzlich auch die Gefahr einer „Zerfaserung“ des Austauschformats, wenn elementare

[5] https://loncapa.msu.edu/adm/help/author.manual.pdf, zuletzt abgerufen am 30.01.2014

Aspekte dort mehrfach oder unterschiedlich definiert werden. Diesem Aspekt wurde zum einen dadurch Rechnung getragen, dass wesentliche Aspekte bereits im vorgeschlagenen Format selbst definiert wurden. Zum anderen können durch die Versionierung der Spezifikation später noch Änderungen und Erweiterungen vorgenommen werden, um „verbreitete“ Aspekte mit in das Format aufzunehmen. Bewusst wurden Metadaten zur Kategorisierung in der ersten Version nicht in das Format aufgenommen. Für Metadaten gibt es zum einen etablierte Standards (vgl. IEEE LOM, Dublin Core) und zum anderen wurden diese oft nicht bzw. nur halbherzig benutzt [Go04]. Ein spezieller Editor wird für das Format nicht benötigt, da jedes System bereits eine eigene Authoring-Funktionalität bereitstellt – eine Entwicklung ist aber vorgesehen.

Neben dem Export wurde für die drei aufgeführten Systeme zudem ein Import implementiert. In diesem Zusammenhang wurden für diese Systeme systemübergreifende Importe (Export aus einem System und Import in ein anderes System) und systeminterne Importe (Export aus einem System und Reimport in dasselbe System) zu Testzwecken durchgeführt. Tabelle 2 zeigt zusammenfassend die wesentlichen Ergebnisse dieser Tests, die im Folgenden näher erläutert werden.

	Export gemäß 0.9.1	**externer Import**	**systeminterner Import**
GATE	X	X	X
JACK	(X)	X	X
Praktomat	X	X	X

Tabelle 2: Unterstützung des vorgeschlagenen Formates durch bestehende Systeme

Ein Export ist prinzipiell aus allen drei aufgeführten Systemen möglich. Beim JACK-System werden jedoch die notwendigen Musterlösungen zum aktuellen Zeitpunkt noch nicht exportiert. GATE und Praktomat erzeugen vollständig valide XML-Dokumente gemäß Version 0.9.1 der Spezifikation.

Ein systeminterner und systemübergreifender Import ist grundsätzlich für jedes der drei Systeme möglich (vgl. „Import“ Spalten). Die (vollständige) Ausführbarkeit einer importierten Aufgabe, die zuvor aus einem anderen System exportiert wurde, ist jedoch nicht immer gewährleistet. Dies ist zum einen durch die unterstützten Bewertungsverfahren und zum anderen durch besondere Systemfeatures begründet. Beispielsweise werden alle drei Systeme für Java-Programmieraufgaben eingesetzt. GATE und Praktomat unterstützen beide JUnit3-Tests (daher auch die Verwendung von JUnit3 im Listing 1) – zwischen diesen beiden Systemen lassen sich daher Aufgaben samt Syntax- und JUnit3-Tests vollständig austauschen und ausführen. Das JACK-System hingegen unterstützt keine JUnit-Tests, sondern setzt auf andere Bewertungsverfahren, die zurzeit exklusiv im JACK-System Anwendung finden. Dennoch lassen sich durch die Möglichkeit systemspezifische Elemente in Form von Metadaten zu exportieren (Elemente */task/meta-data*, Element */task/tests/test/test-configuration/test-meta-data*), wie weiter oben bereits erwähnt, wesentliche Elemente von Programmieraufgaben systemübergreifend austauschen und zugleich systeminterne Importe verlustfrei durchführen.

Das Format erlaubt grundsätzlich neben dem Austausch von Aufgaben zwischen reinen Assessment-Systemen auch die Nutzung als Beschreibung für Aufgaben zwischen einem LMS und einem Evaluations-System (serviceorientierte Architektur oder einer Middleware): Ein LMS kann die Verwaltung sowie Darstellung der Aufgaben übernehmen und zur Evaluation die studentischen Einreichungen zusammen mit der Aufgabenbeschreibung an einen Evaluationsservice bzw. eine durch Plug-Ins erweiterbare Middleware, die mehrere Programmbewertungssysteme einbinden kann, schicken (vgl. [PJR12, Be13]). Hierfür ist jedoch noch eine genaue Spezifikation erforderlich.

5 Zusammenfassung und Ausblick

In diesem Artikel wurden spezielle Anforderungen an ein Austauschformat für Programmieraufgaben sowie ein Ansatz vorgestellt, der diese Anforderungen berücksichtigt. Dabei handelt es sich nicht um eine „one-size-fits-all“ Spezifikation für vorher festgelegte Sprachen und Testtypen, sondern um ein durch XML-Namespaces erweiterbares Format. Damit bietet das Format eine gute Grundlage zur Erhöhung der Interoperabilität zwischen verschiedenen Systemen. Ein Export und Import von Aufgaben mit der vorgeschlagenen Spezifikation ist bereits aus mehreren Systemen möglich. Ebenso lassen sich wesentliche Elemente von Aufgaben, die in das Austauschformat exportiert wurden, in diese Systeme importieren. Ein systemübergreifender Austausch von Aufgaben unter Verwendung des vorgeschlagenen Austauschformats ist durchführbar.

Nächste Schritte beinhalten die Verfeinerung der im Austauschformat beschriebenen Elemente sowie die Integration in weitere Systeme. Für einen einfachen Austausch wird der Aufbau eines Repositories für Aufgaben angestrebt, so dass Aufgaben z. B. direkt aus einem System heraus gesucht und importiert werden können. Ein Repository bietet die Chance, die Qualität der Lehre durch (wiederholte) Nutzung guter Aufgaben insbesondere auch über Hochschulgrenzen hinweg zu steigern. Hierfür muss nicht unbedingt ein eigenes Repository aufgesetzt werden, eine Nutzung bestehender Repositories wird explizit nicht ausgeschlossen. Von besonderer Bedeutung für den Einsatz eines Repositories für Programmieraufgaben ist, dass ein Zugriff auf die vollständigen Aufgabenspezifikationen für Lernende ausgeschlossen ist (vgl. LON-CAPA). Einige der in diesem Paper betrachteten Tools und das hier vorgestellte Austauschformat sind in vom BMBF geförderte Projekte („Qualitätspakt Lehre“) eingebunden, welche den Einsatz der Tools, des Austauschformats und deren Weiterentwicklung und verstärkte Nutzung auch längerfristig garantieren.

Literaturverzeichnis

[Al05] Ala-Mutka, K.: A Survey of Automated Assessment Approaches for Programming Assignments. In Computer science education, 15(2), 2005, S. 93-102.

[ASS08] Altenbernd-Giani, E.; Schroeder, U.; Stalljohann, P.: eAixessor-A Modular Framework for Automatic Assessment of Weekly Assignments in Higher Education. In Proc. 7th IASTED International Conference. Vol. 610, 2008, S. 99

[Be13] Becker, S. et. al.: Prototypische Integration automatisierter Programmbewertung in das LMS Moodle. In Proc. Workshop APB'13, 2013.

[CMM04] Cesarini, M.; Mazzoni, P.; Monga, M.: Learning Objects and Tests. In The IASTED International Conference on Web-Based Education, 2004, S. 520-524.

[Ed04] Edwards, S. H.: Using software testing to move students from trial-and-error to reflection-in-action. In ACM SIGCSE Bulletin 36(1), 2004, S. 26-30.

[Go04] Godby, C. J.: What Do Application Profiles Reveal about the Learning Object Metadata Standard? In Adriane Article in eLearning Standards, 2004.

[HQW08] Hoffmann, A.; Quast, A.; Wismüller, R.: Online-Übungssystem für die Programmierausbildung zur Einführung in die Informatik. In Proc. DeLFI, 2008, S. 173-184.

[Hü05] Hügelmeyer, P. et. al.: Integration des Virtuellen Prüfungssystems ViPS in die Lehr-/Lernplattform Stud.IP. In Proc. Workshop on e-Learning 2005, HTWK Leipzig, S. 187-196.

[Ih10] Ihantola, P. et. al.: Review of recent systems for automatic assessment of programming assignments. In Proc. Koli Calling'10, S. 86-93.

[KSZ02] Krinke, J.; Störzer, M.; Zeller, A.: Web-basierte Programmierpraktika mit Praktomat. Softwaretechnik-Trends 22(3), 2002, S. 51-53.

[LQ09] Leal, J. P.; Queirós, R.: Defining Programming Problems as Learning Objects. In Proc. ICCEIT'09, Venice, Italy.

[Mo07] Morth, T.; Oechsle, R.; Schloß, H.; Schwinn, M.: Automatische Bewertung studentischer Software. In Proc. Pre-Conference Workshops der 5. e-Learning Fachtagung Informatik DeLFI 2007, 2007.

[PJR12] Priss, U.; Jensen, N.; Rod, O.: Software for E-Assessment of Programming Exercises. In Goltz et al. (Hrsg.) Informatik 2012, GI LNI, P-208, S. 1786-1791.

[QL11] Queirós, R.; Leal, J. P.: Pexil: Programming exercises interoperability language. In Proc. Conferência - XML: Aplicações e Tecnologias Associadas (XATA), 2011.

[QL12] Queirós, R.; Leal, J. P.: PETCHA: a programming exercises teaching assistant. In: Proc. ACM ITiCSE'12, 2012. S. 192-197.

[Ro07] Romeike, R.: Three Drivers for Creativity in Computer Science Education. In Proc. of Informatics, Mathematics and ICT: a 'golden triangle'. Boston, US, 2007.

[RRH12] Rodríguez-del-Pino, J.; Rubio-Royo, E.; Hernández-Figueroa, Z.: A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. In Proc. CSREA EEE'12, 2012.

[SBG09] Striewe, M.; Balz, M.; Goedicke, M.: A Flexible and Modular Software Architecture for Computer Aided Assessments and Automated Marking. In Proc. CSEDU'09, 2009, S. 54-61.

[SOP11] Strickroth, S.; Olivier, H.; Pinkwart, N.: Das GATE-System: Qualitätssteigerung durch Selbsttests für Studenten bei der Onlineabgabe von Übungsaufgaben? In Proc. DeLFI'11, 2011, S. 115-126.

[Sp06] Spacco, J. et. al.: Experiences with Marmoset: Designing and using an advanced submission and testing system for programming courses. In ACM SIGCSE Bulletin 38(3), 2006, S. 13-17.

[Tr07] Truong, N.: A Web-Based Programming Environment for Novice Programmers. PhD Thesis, 2007: http://eprints.qut.edu.au/16471/1/Nghi_Truong_Thesis.pdf

[Ve08] Verhoeff, T.: Programming Task Packages: Peach Exchange Format. In Olympiads in Informatics, 2, 2008, S. 192–207.

[Ve11] Verdú, E. et. al.: A distributed system for learning programming on-line. In Computers & Education 58(1), 2011, S. 1-10.

[WW07] Weicker, N.; Weicker, K.: Automatische Korrektur von Programmieraufgaben – Ein Erfahrungsbericht. In Flexibel integrierbares e-Learning - Nahe Zukunft oder Utopie, Proc. Workshop on e-Learning 2007, S. 159-173.