aus

Oliver J. Bott, Peter Fricke, Uta Priss, Michael Striewe (Hrsg.)

Automatisierte Bewertung in der Programmierausbildung

Digitale Medien in der Hochschullehre Band 6
2017, 420 Seiten, br., 42,90 €, ISBN 978-3-8309-3606-0

Waxmann Verlag GmbH
www.waxmann.com info@waxmann.com

23 Integration mithilfe der Middleware Grappa

Peter Fricke

Zusammenfassung
Die Vielzahl von Lernmanagementsystemen (LMS) hat in der Vergangenheit stark zugenommen. Ebenso werden immer mehr Programmiersprachen populärer, so dass der Bedarf an Programmen, die eine automatische Bewertung von Programmcode vornehmen („Grader"), zunimmt. Grappa[a] übernimmt dabei die Aufgabe einer Middleware – die Anbindung von Gradern an Lernmanagementsysteme und andersherum. Grappa spezifiziert die Schnittstelle zum LMS und die zum Grader und stellt dabei Tools zur effizienteren Anbindung zur Verfügung. Im Folgenden wird zunächst die Idee und danach die genaue Funktionsweise vorgestellt.

a Der Begriff entstand aus dem Kofferwort Grapper (Grader Wrapper), welches anschließend mit der Intention, Genuss zu assoziieren, abgewandelt wurde.

23.1 Motivation und Anforderungen

An der Hochschule Hannover (HsH) kommen seit 2010 für SQL und die Programmiersprache Java verschiedene Grader zum Einsatz. aSQLg („automated SQL grader" – Kapitel 12) untersucht und bewertet studentische Abgaben zu SQL-spezifischen Aufgaben. Graja („Grader for java programs" – Kapitel 11) prüft studentische Java-Programme auf deren Funktion und deren Ressourcenverbrauch. Beide Grader wurden über unterschiedliche Frontends angesprochen und den Lehrenden als auch Studenten zur Verfügung gestellt.

Dieses Vorhaben wird aus Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01PL11066D gefördert. Die Verantwortung für den Inhalt dieses Beitrags liegt bei dem Autor.

Bei mehreren Evaluationen der Hochschule Hannover (siehe auch [Stö+13] und [Stö+14]) wurde von Studierenden angemahnt, dass sie mehrere Systeme zum Hochladen ihrer Lösungen brauchen. Ebenso wurde die Darstellung und der Detailgrad des Feedbacks als „zu technisch“ benannt. Es entstand ein Bedarf die Benutzerschnittstelle zu vereinfachen und zu vereinheitlichen.

Ohne Middleware müsste für jeden vorhandenen oder gewünschten Grader eine Anbindung an das LMS konzeptioniert und programmiert werden. Das Kernkonzept von Grappa ist die Standardisierung der verschiedenen Grader-Schnittstellen auf eine einfache, für das Internet, und somit für LMS nutzbare Schnittstelle. Dabei sind über diese Schnittstelle gleich mehrere Grader ansprechbar. Der Programmieraufwand am LMS reduziert sich auf nur eine Erweiterung für Grappa. Über eine REST-Schnittstelle (siehe Kapitel 23.4) werden XML-Dateien zwischen Grappa und dem LMS ausgetauscht. Die Grader selbst werden von Grappa über eine Java-Schnittstelle angesprochen (näher beschrieben in Kapitel 23.5), welche für jeden Grader integriert werden muss.

Grappa soll unabhängig von **einem** konkretem LMS oder Grader arbeiten können. Um der zentralen Funktion gerecht zu werden, eine studentische Lösung zu einer Aufgabe vom LMS entgegennehmen und durch einen Grader bewerten zu lassen und das Ergebnis dem LMS zurückzuliefern, müssen verschiedene Anforderungen seitens der Grader als auch der LMS berücksichtigt werden.

Die Middleware soll die Beschaffenheit einer Aufgabe (Programmieraufgabe, Datenbankaufgabe, Modellierungsaufgabe, etc.) nicht explizit kennen. Grappa muss mögliche einfache und komplexe Eigenschaften von LMSen und Gradern handhaben können. Dabei müssen sowohl didaktische als auch technische Ausprägungen in einer Aufgabe und deren Bewertung dargestellt werden. Ein Grader sollte weitere über die Aufgabenstruktur hinausgehende Aspekte (bspw. syntaktische oder semantische Korrektheit), hinzufügen können. Grappa soll diese Aufgaben permanent kennen und speichern können. Die Bewertungsskala zwischen Grader und LMS kann sich unterscheiden. Grappa soll vorzugsweise mit einer offenen Wertung umgehen und ggf. mit Unterstützung des LMS eine entsprechende Umrechnung vornehmen.

Ein Grader benötigt oft über die Aufgabenbeschreibung hinausgehende Konfigurationsdateien. Grappa muss diese Dateien ebenfalls in Bezug auf die hinterlegten Aufgaben vorhalten und zur Verfügung stellen können.

Die Ergebnisse der Bewertung eines Graders können sehr einfach oder komplex gestaltet sein. Etwaige Bewertungskommentare können allgemein oder spezifisch sein. Dabei sollte zwischen Feedback für Lehrende und Studierende und dem Typ des Kommentars unterschieden werden. Ebenso muss Grappa zwischen LMS und

Grader die Darstellung der Bewertungskommentare in den Formaten PDF, XML, HTML und Text vermitteln, oder im besten Falle konvertieren.

Der Prozess der Bewertung von Programmieraufgaben kann unterschiedliche Zeitspannen in Anspruch nehmen. Eine Dauer von wenigen Sekunden (Prüfung einer simplen Textausgabe bspw. in Java), bis hin zu mehreren Minuten (komplexere JOIN-Abfragen mit SQL) ist dabei denkbar. Um eine zügige Verarbeitung der Bewertungsvorgänge zu gewährleisten, sollte Grappa mehrere Einreichungen parallel vom LMS annehmen und diese, wenn unterstützt, ebenso parallel an den Grader schicken. Sollte der Grader keine parallele Bewertung unterstützen, oder zu viele Bewertungsvorgänge auf einmal bearbeitet werden müssen, soll Grappa diese Vorgänge in einer Warteschlange dem Grader vorhalten und für das LMS asynchron aufrufbar machen.

Weitere Anforderungen um den Umgang mit der Parametrierung von Aufgaben zu spezifizieren wurden in [GHW15] vorgestellt.

23.2 Fachdatenmodell

In diesem Kapitel wird das Fachdatenmodell von Grappa ausführlich beschrieben. Um allen Anforderungen (siehe Kapitel 23.1) gerecht zu werden, unterscheidet Grappa in vier Hauptentitäten, welche in Tabelle 23.1 gezeigt und in den folgenden Unterkapiteln näher erläutert werden.

Entität	**Mögliche Interpretation**	**Querbezüge zu**
Problem	Ein Aufgabenblatt oder eine Aufgabe	*GrdCfg* (optional)
GrdCfg	Dateien für das GraderBackend zur Bewertung oder Konfiguration eines, mehrerer oder aller Aspekte des Problems	Graderabhängig: *Problem*
Submission	Studentische Lösung	*Problem* (nur temporär an Schnittstellen)
GradingResult	Ergebnis der Bewertung	*Problem*

Tabelle 23.1: Die vier Hauptentitäten von Grappa

23.2.1 Konfigurationsdateien (GrdCfg)

Um einen Grader zu benutzen bedarf es häufig verschiedener Konfigurationsdaten, in der Grappa Begriffswelt GraderConfigurations – *GrdCfgs*. Grappa persistiert diese *GrdCfgs* um eine Wiederverwendung zu ermöglichen. Abbildung 23.1 zeigt die Struktur einer *GrdCfg*. *GrdCfgs* werden von Grappa nicht interpretiert. Sie werden gespeichert und bei Bedarf an das *BackendPlugin* (siehe Kapitel 23.5) weitergeleitet.

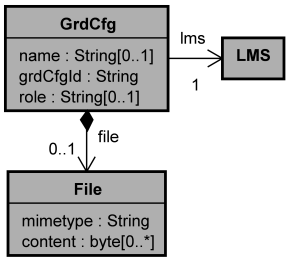

Abbildung 23.1: Fachdatenmodell der *GrdCfg*

name kann genutzt werden um eine Identifizierung der *GrdCfg* für den Benutzer zu ermöglichen.

grdCfgId und die LMS-Id beschreiben eine Konfigurationsdatei eindeutig.

role dient zur Identifikation der Funktion der *GrdCfg* für das BackendPlugin.

file In diesem Objekt befindet sich die eigentliche Konfigurationsdatei, diese kann, wenn es der Grader verlangt, mit einem MimeType versehen werden. *GrdCfg* können beliebige Dateien beinhalten, einfache Properties-Files, Java-Programme und benötigte Bibliotheken sind vorstellbar. Die Dateigröße variiert dabei zwischen wenigen Bytes und mehreren Megabytes.

23.2.2 Aufgabenstruktur (Problem)

Eine Aufgabe kann aus Sicht des LMS in mehrere Unteraufgaben oder Teilaspekte aufgeteilt sein. Diese Struktur spiegelt sich in der Bewertung der Aufgabe wider (siehe Kapitel 23.2.3). Der Grader kann dieser Struktur noch weitere Ebenen hinzufügen, indem zu einer Teilaufgabe bestimmte Aspekte, wie z. B. syntaktische oder semantische Korrektheit oder stilistische Fragen, hinzufügt werden. Grappa

soll diese Aufgabenstruktur kennen und persistent speichern. Ein *Problem* (Abbildung 23.2) definiert so eine Aufgabe mit verschiedenen Unteraufgaben oder Teilaspekten in *ProblemNodes*. Diese ProblemNodes können hierarchisch aufgebaut und konfiguriert werden. Ähnlich der *GrdCfg* ist eine Aufgabe über die *problemId* und die *lmsId* eindeutig bei Grappa identifiziert. Jede *ProblemNode* muss vom LMS einen *nodeKey* zugewiesen werden. Dieser ist für alle Kinder eines Knoten nur auf einer Ebene eindeutig. Der *nodeKey* ist eine Referenz auf eine Aufgabe, die das Backend und der Grader nutzen können, um Verknüpfungen zwischen bspw. einer Musterlösung oder einem speziellen Test und dieser (Teil-)Aufgabe (-Aspekt) herzustellen. Weiterhin wird für jede *ProblemNode* die maximale erreichbare Punktzahl, *scoreMax*, hinterlegt. Für unterschiedliche *ProblemNodes* bedarf es verschiedener *GrdCfg* für den Grader. Des Weiteren kann über *acceptedRDKinds* ein Wunsch an den Grader oder das *BackendPlugin* gestellt werden, wie das Ergebnis dargestellt werden soll. Schließlich können über die *computingResources* Einschränkungen für die maximale Laufzeit, Arbeitspeicher- und Festplattenbelegung der Teilaufgabe oder -aspekts an das GraderBackend gegeben werden.

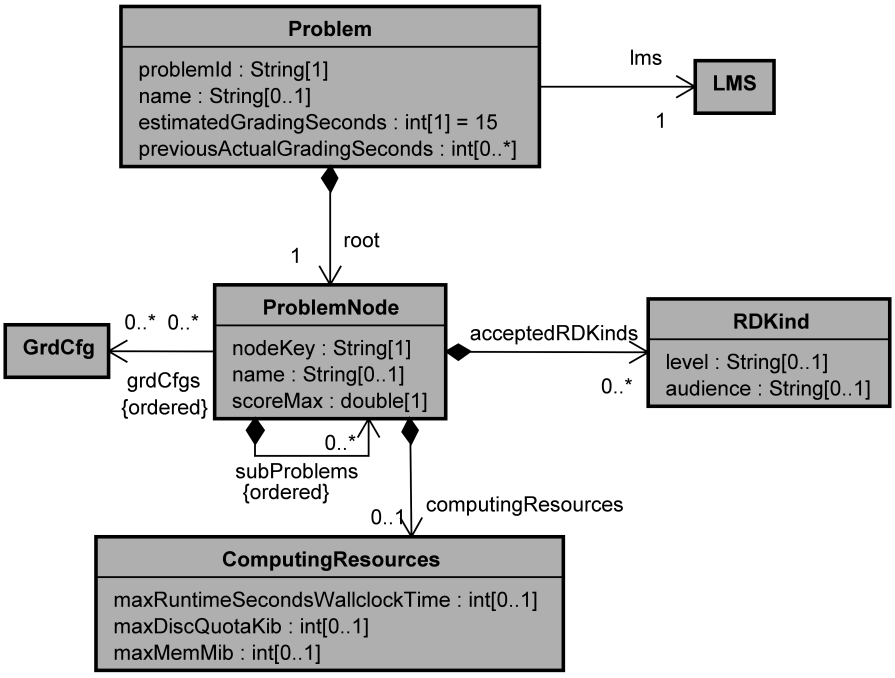

Abbildung 23.2: Fachdatenmodell eines Problem

Grappa persistiert diese *Problems* im derzeitigen System. Somit ist Grappa in der Lage, die Konsistenz zwischen *ProblemNodes* und *GrdCfgs* (23.2.1) zu erhalten. Es ist nicht möglich *GrdCfg*s zu löschen, die von einem oder mehreren *Problems* referenziert werden. Ein weiterer Vorteil ist die schnellere Wiederverwendbarkeit bei der asynchronen Bewertung (siehe Kapitel 23.6) wenn mehrere studentische Lösungen zu einer Aufgabe in kurzer Zeit abgegeben werden.

23.2.3 Studentische Lösung und Ergebnis der Bewertung (Submission & GradingResult)

Eine studentische Lösung kann aus einer Datei oder mehreren Dateien bestehen. Die Zuordnung zur richtigen Aufgabe, dem *Problem*, geschieht über den Aufruf der eindeutigen *problemId* innerhalb der *RunGrader*-Methode (siehe Kapitel 23.4).
Grappa stellt das Ergebnis des Gradens eines *Problems* als *GradingResult* dar. Abbildung 23.3 zeigt, dass ein *GradingResult* genauso hierarchisch aufgebaut ist wie ein *Problem*. Es gibt ein *root-Result*-Objekt, welches wiederum *Result*-Objekte in beliebiger Baumtiefe haben kann. Auf der oberste Ebene verweist jedes *Result*-Objekt auf die zugehörige *ProblemNode*. Darunterliegende *Results* dienen zur Darstellung von Teilaspekten, die gegebenenfalls vom Grader zusätzlich generiert wurden.

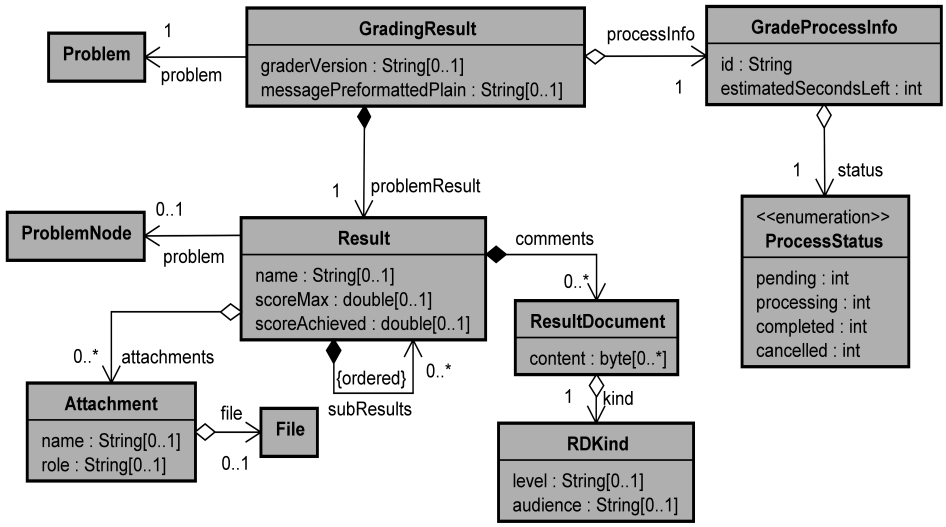

Abbildung 23.3: Darstellung eines GradingResults

name beschreibt die Art des Teilaspekts, oder den Aufgabennamen.

scoreMax zeigt die maximal erreichbare Punktzahl für diesen Teilbaum.

scoreAchieved zeigt die erreichte Punktzahl nach dem grade-Aufruf.

comments bestehen aus ein oder mehreren *ResultDocuments*. Diese dienen zur Darstellung des Grader-Feedbacks und können verschiedene Ausprägungen und Detailgrade haben.

attachments können optional bei Bedarf vom Grader an ein Result angehängt werden. Sie sind unspezifiziert und werden von Grappa nicht interpretiert und lediglich an das Frontend weitergegeben. Beispiele für *attachments* wären statistische Informationen über den *grade*-Aufruf oder ähnliches.

Dem *GradingResult* wird zusätzlich eine *graderVersion* zugewiesen. Sollte es bereits vor dem grade-Aufruf zu Fehlern innerhalb von Grappa oder des Backend-Plugins kommen, werden diese im *messagePreformattedPlain* Feld abgelegt. Somit kann dem LMS ein expliziter Fehler im BackendPlugin oder Grader selbst mitgeteilt werden.

Die *processInfo* enthält Informationen über den aktuellen Stand der Bearbeitung. Gültige Zustände sind:

Pending Die *Submission* befindet sich in der Warteliste und die Bearbeitung hat noch nicht begonnen.

Processing Die *Submission* wurde an das Backend gegeben und wird gerade bearbeitet.

Completed Der grade-Aufruf wurde beendet. Ob ein vollständiges Ergebnis vorliegt, ist den Result-Objekten und ggf. dem *messagePreformattedPlain* zu entnehmen.

Cancelled Die Bewertung wurde durch das Frontend abgebrochen (siehe *cancel*-Methode in Kapitel 23.4).

Der Status ist aus Sicht von LMS zu Grappa zu sehen und beschreibt lediglich den Verarbeitungszustand des *grade*-Befehls. Etwaige Fehler des Graders, oder BackendPlugins werden damit nicht beschrieben. GradingResults werden temporär von Grappa persistiert. Holt ein LMS das Ergebnis nicht innerhalb einer vorkonfigurierten Zeitspanne ab, wird es gelöscht und der grade-Aufruf muss erneut durchgeführt werden. Nähere Informationen darüber sind dem Kapitel 23.6 zu entnehmen.

RDKind

Das *RDKind* erfüllt zwei Aufgaben. Zum einen kann beim Anlegen eines Problems ein Wunsch gegenüber dem Grader ausgesprochen werden, in welchem Format und in welchen Detailgrad das Ergebnis zurückgegeben werden soll. Zum anderen wird ein Ergebnisdokument (*ResultDocument* Kapitel 23.3) mit einem *RDKind* versehen, um die Darstellung und Ausprägung zu spezifizieren.

level stellt dabei den Detailgrad des *ResultDocuments* dar. Mögliche Ausprägungen sind *fatal*, *error*, *warning*, *info* und *debug*.

audience beschreibt die Zielgruppe des *ResultDocuments*. Mögliche Ausprägungen sind *teacher*, *student* und *both*. Dadurch ist das LMS in der Lage das Feedback der richtigen Zielgruppe anzuzeigen.

representation dient zur Identifikation des Formates des *ResultDocuments*, derzeit sind drei Darstellungen möglich: *Html*, *Pdf* und *PlainText*.

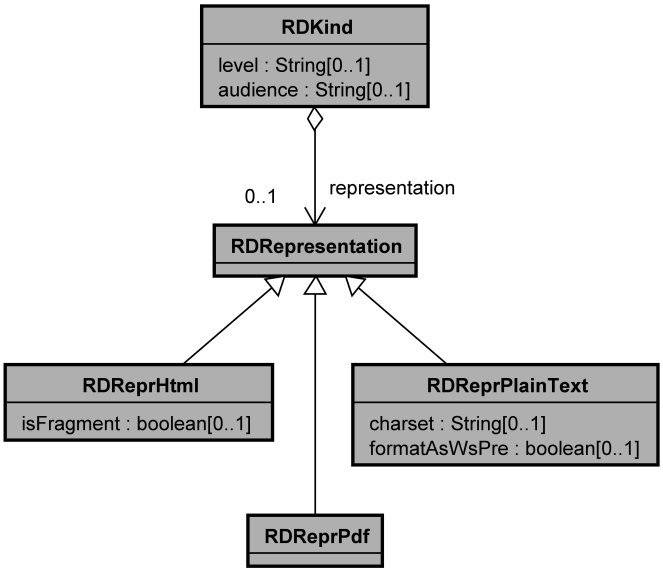

Abbildung 23.4: Arten von ResultDocuments

23.3 Komponenten

Die Middleware Grappa bietet dem Frontend insgesamt zwei und dem Backend eine Schnittstelle an. Mit diesen Schnittstellen müssen die jeweiligen Komponenten in Form von Erweiterungen (Plugins) kommunizieren. Abbildung 23.5 zeigt das Komponentendiagramm einer typischen Grappa-Installation.

LMS Das LMS dient dem Nutzenden als Benutzeroberfläche und sollte erweiterbar sein durch bspw. eine Plugin-Architektur.

LMSToGrappaPlugin Dieses Plugin nutzt die öffentlichen Schnittstellen von Grappa und übersetzt diese Informationen in die Domäne des LMS.

Setup Über die Setup-Schnittstelle erfolgt das Anlegen und Bearbeiten von Konfigurationsdateien und Programmieraufgaben in Grappa.

RunGrader Öffentliche Schnittstelle zum Einreichen und Bewerten studentischer Lösungen.

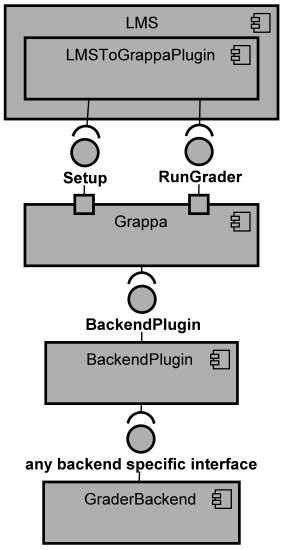

Abbildung 23.5: Fachdatenmodell eines Grappa-nutzenden Systems inklusive aller Komponenten

BackendPlugin Dieses Plugin übersetzt die Begriffswelt von Grappa in möglicherweise mehrere GraderBackend Aufrufe und übersetzt dessen Antwort zurück.

GraderBackend Denkbar sind hier mehrere GradingTools, oder nur ein Grader, die wiederum vom *BackendPlugin* aufgerufen werden.

Grappa wurde als Java-Servlet entwickelt und kann in einen typischen Application-Server, wie z. B. Tomcat, JBoss oder Glassfish, eingesetzt werden. Eine Anbindung an eine Datenbank ist ebenfalls notwendig, da Grappa *GrdCfgs* und *Problems* speichert. *GrdCfgs* können so mehrfach genutzt werden und das Bewerten mehrerer Lösungen zur gleichen Zeit kann schneller erfolgen.

23.4 Öffentliche Schnittstelle

Das Frontend von Grappa bietet zwei Schnittstellen für das LMS. Beide Schnittstellen werden über REST-Aufrufe angesprochen.

Die Setup-Schnittstelle dient zum Anlegen, Ändern und Löschen von *GrdCfg*s und *Problem*s. Das getrennte Anlegen von *GrdCfg*s und *Problem* wurde gewählt, um eine Wiederverwendbarkeit von *GrdCfg*s zu gewährleisten.

Einem LMS ist es möglich, eine *GrdCfg* anzulegen, die von mehreren Problems benutzt werden kann. Die interne Struktur von Grappa erlaubt kein Löschen einer *GrdCfg* die von einem *Problem* referenziert wird.

Für alle drei Operationen bekommt das LMS eine XML-Datei und einen HTTP-Status zurückgeliefert. Diese beinhaltet entweder eine Erfolgsmeldung, oder im Fehlerfall eine detaillierte Fehlerbeschreibung.

Die RunGrader-Schnittstelle kann erst genutzt werden, wenn ein *Problem* vollständig angelegt wurde. Sie dient dem LMS, um eine studentische Abgabe über Grappa dem Grader zur Verfügung zu stellen und zu bewerten. Der Schnittstellenparameter subm stellt den Bezug zum Problem her. Grappa kann intern mit einem asynchronen Bewertungsmechanismus laufen. Das LMS muss eine in der Warteschlange befindliche *Submission* per Polling abfragen. Grappa soll keine Kenntnisse über die Funktionsweise der verschiedenen LMS haben. Durch bspw. einen Callback-Mechanismus, gäbe es eine zu enge Kopplung zwischen LMS und Grappa.

Beim Grading, oder Polling (siehe 23.6) liefert Grappa dem LMS ein *GradingResult* in Form einer XML-Datei zurück. Diese beinhaltet ein vollstän-

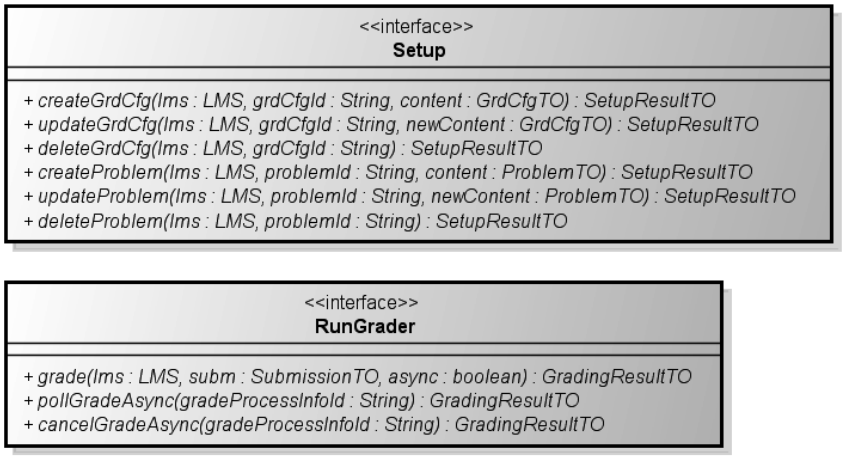

Abbildung 23.6: Methodenaufrufe der öffentlichen Schnittstellen;

diges Ergebnis, sofern der Vorgang abgeschlossen ist. Befindet sich die *Submission* weiterhin in der Warteschleife, so wird nur eine Prozessinformation (*Pending*, *Processing*, *Completed*, *Cancelled*) inklusive einer Id und einer voraussichtlichen Wartezeit übergeben. Das LMS kann auf diese Informationen dementsprechend reagieren und dem Studierenden oder Lehrenden eine Fehlermeldung anzeigen.

23.4.1 REST-Aufrufe

Die Kommunikation zwischen LMS und Grappa geschieht über HTTP-REST und XML-Dateien. Diese XML-Dateien unterliegen mehreren grappaspezifischen Schemadateien. Nehmen wir an, unter einer beliebigen URL läuft ein Grappa-Server (URL abgekürzt durch *Grappa*) und ein LMS mit der pseudo LMS-Id `hsh` wollte die in Abb. 23.6 gezeigten Methoden aufrufen. Dabei stellen Werte mit `_id_` den, vom LMS erzeugten, Identifikator für die jeweilige Ressource dar. Die *grdcfgsIds*, *problemIds* und *gradeProcessInfoIds* werden von Grappa erzeugt. Folgende Aufrufe sind für die Setup-Schnittstelle gültig[1]:

GrdCfg anlegen POST Grappa/hsh/grdcfgs/ new-grdcfg.xml

GrdCfg updaten PUT Grappa/hsh/grdcfgs/**_id_grdcfg** updated-grdcfg.xml

GrdCfg löschen DELETE Grappa/hsh/grdcfgs/**_id_grdcfg**

1 Derzeit nur vorgesehen, jedoch nicht implementiert sind GET-Aufrufe für eine komplette Liste der *GrdCfg*s und *Problem*s.

Problem anlegen POST Grappa/hsh/problems/ new-problem.xml

Problem updaten PUT Grappa/hsh/problems/**_id_problem** updated-problem.xml

Problem löschen DELETE Grappa/hsh/problems/**_id_problem**

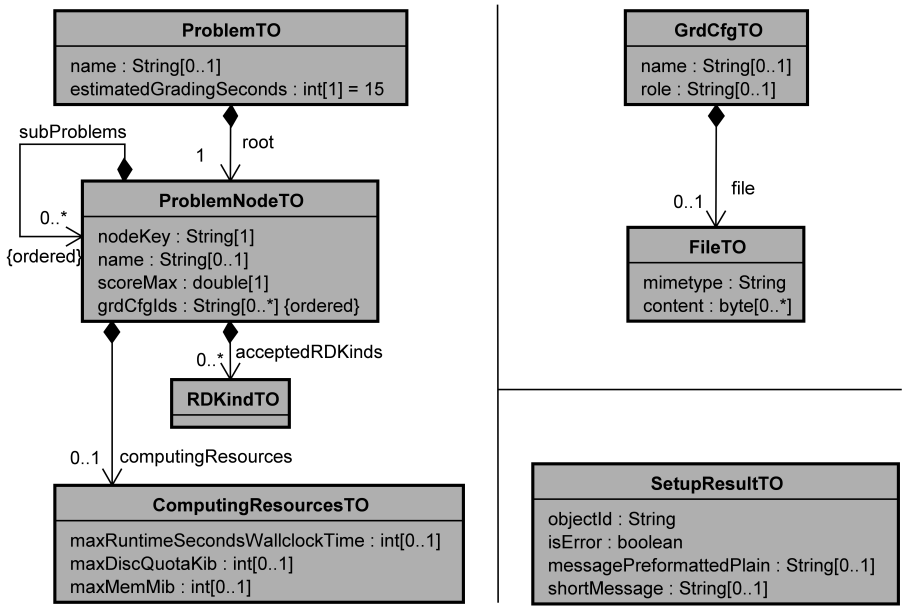

Abbildung 23.7: Darstellung aller XML-Dokumente der Setup-Schnittstelle

Dabei antwortet Grappa immer mit einem Ergebnisdokument in Form eines *SetupResults* (siehe Abbildung 23.7). Beim Anlegen von *GrdCfg*s oder Problems beinhaltet diese XML-Datei ebenfalls die erzeugte Objekt-ID, damit eine spätere Referenzierung möglich ist. Etwaiges Fehlverhalten oder Erfolgsmeldungen werden als unformatierter String in *messagePreformattedPlain* und ggf. *shortMessage* dem LMS zur Verfügung gestellt.
Die Grader-Schnittstelle:

Submission graden POST Grappa/hsh/gradeprocesses?async=*true* submission.xml

Grading pollen GET Grappa/hsh/gradeprocesses/**gradeProcessInfoId**

Grading löschen/abbrechen DELETE Grappa/hsh/gradeprocesses/**gradeProcessInfoId**

Grappa antwortet auch im Fehlerfall immer mit einem *GradingResult* in Form einer XML-Datei. Bei beiden Schnittstellen überträgt Grappa übliche HTTP-Statuscodes.

23.4.2 Tools

Im Rahmen der Entwicklung eines Moodle-Plugins für Grappa (Kapitel 18.3.1) wurde eine PHP-Clientbibliothek entworfen um den Programmieraufwand für LMS weiter zu vereinfachen. Der „Grappa-PHP-Client“ übernimmt die Übersetzung von PHP-Objekten in das XML-Format und zurück. Der Client stellt ebenfalls Methoden zur Kommunikation mit Grappa zur Verfügung. Dabei ist der komplette Polling-Mechanismus für den Apache-Webserver unter Linux und Windows integriert. Der Grappa-PHP-Client dient als weitere Schnittstelle zwischen *LMSToGrappaPlugin* und der *Setup* und *RunGrader*-Schnittstelle in der gezeigten Architektur (Abbildung 23.6).

23.5 Grader-Schnittstelle – BackendPlugin

Die Kommunikation zwischen einem Grader und Grappa wird über ein *BackendPlugin* ermöglicht. Für jede Art von Grader muss ein Java-Plugin entwickelt werden, welches das in Abbildung 23.8 dargestellte Interface implementiert. Dieses Plugin übernimmt Übersetzungsaufgaben und die Kommunikation zwischen Grappa und Grader in beide Richtungen.

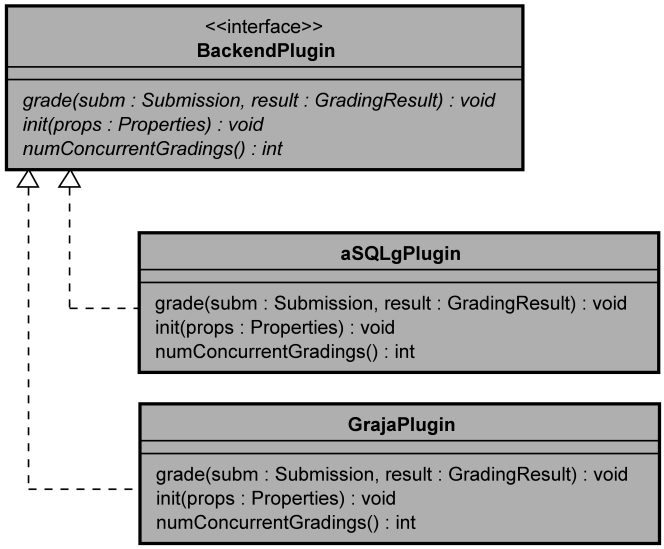

Abbildung 23.8: Fachdatenmodell der Plugin-Schnittstelle

Grappa liefert dem *BackendPlugin* die Rohdaten des Problems in Form eines *Result*-Objektes (erläutert in Kapitel 23.2.3). Das Plugin kann nun über die verschiedenen Teilaufgaben iterieren und je nach Einsatzmöglichkeit die komplette oder jede einzelne Teilaufgabe vom Grader bewerten lassen. Die Resultate des Graders werden anhand der verfügbaren *acceptedRDKinds* entweder direkt vom Grader oder vom *BackendPlugin* transformiert und an das entsprechende *Result*-Objekt angehängt. Es bleibt dem Grader und *BackendPlugin* überlassen, ob etwaige Teilaspekte des Resultates ebenfalls angehängt werden.

23.6 Ablauf

Bevor vom LMS eine studentische Lösung an Grappa übergeben werden kann, muss ein Lehrender alle Konfigurationsdateien (*GrdCfgs*) anlegen und eine Aufgabe (*Problem*) erstellen. Kapitel 18.3.1 zeigt diese Erstellung beispielhaft mit Moodle. Da diese Daten lediglich in der Grappa-Datenbank abgespeichert werden, bedarf es hier keiner detaillierten Beschreibung.

Sobald ein *Problem* erstellt wurde, kann ein Studierender über das LMS eine Lösung für diese Aufgabe abgeben. Abbildung **??** zeigt einen verkürzten jedoch genauen (asynchronen) Ablauf innerhalb des Grappasystems.

Sobald die studentische Lösung erfolgreich vom LMS entgegen genommen wurde, kann das LMS anhand der gespeicherten *problemId* und der expliziten Grappa-Grader-Instanz einen *grade*-Befehl an die *RunGrader*-Schnittstelle absenden. Grappa prüft nun das *Problem* inklusive aller Konfigurationsdateien am Datenbankserver und validiert somit Zuordnung der *Submission* an das *Problem*. Grappa stellt anhand der Größe der Warteschlange (*Queue*) die ungefähre Dauer des Gradingvorgangs fest und liefert dieses Ergebnis mit dem *ProcessStatus* „waiting“ und einer *GradeProcessInfoId* dem LMS zurück. Das LMS ist somit angewiesen, nach dieser Zeit ein Polling zum gegebenen Identifikator bei Grappa vorzunehmen.

Bei Grappa wird zeitgleich die Warteschlange von oben herab abgearbeitet. Dabei wird der nächste *GradingTask* bearbeitet und zunächst von der Datenbank abgerufen und per *grade*-Methode an das *BackendPlugin* weitergegeben. Das *BackendPlugin* kann nun das *Problem* untersuchen und die einzelnen *ProblemNodes* mit zugehörigen *GrdCfgs* für das Graderbackend unter Umständen umwandeln und bereitstellen. Mit *any Backend* können ebenfalls mehrere externe Tools aufgerufen werden, dies obliegt der Implementierung des BackendPlugins. Sobald das *BackendPlugin* die Ergebnisse der/des Grader/s verarbeitet und in die

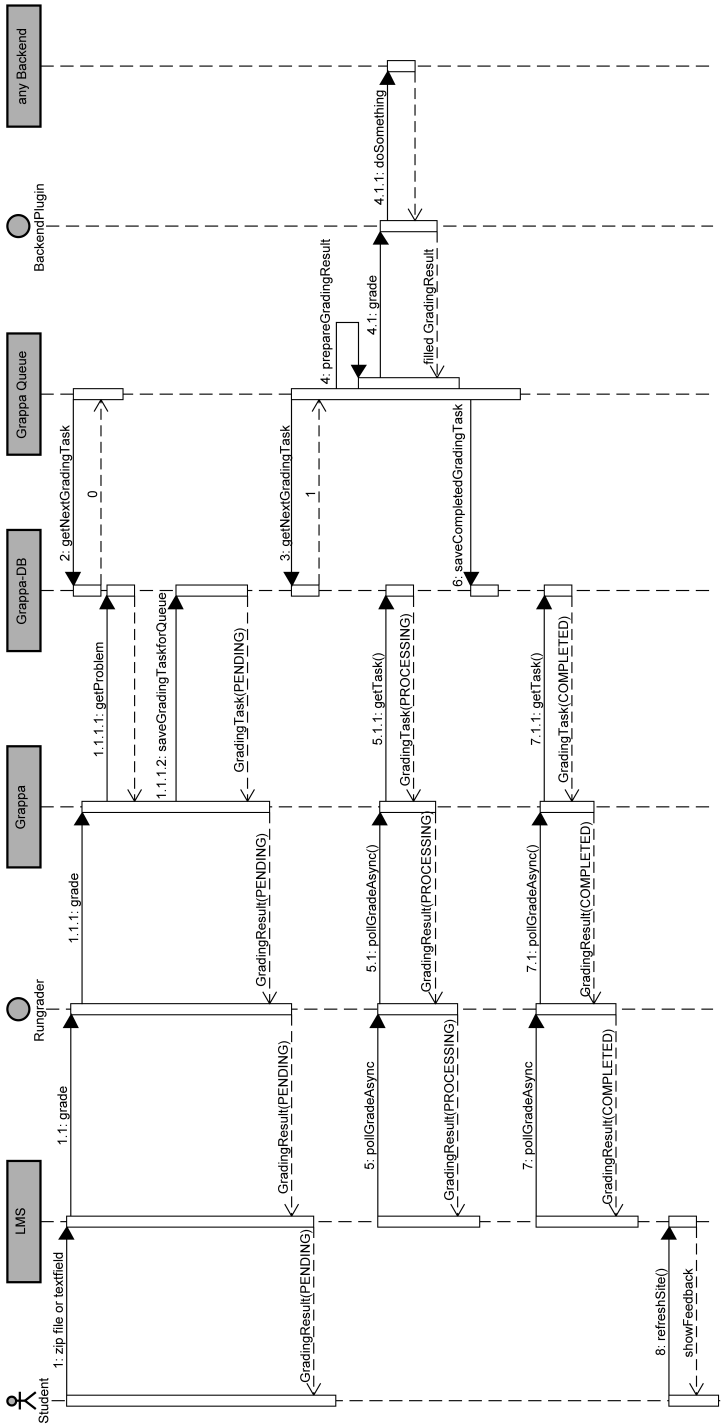

Abbildung 23.9: Verkürzte Darstellung des asynchronen Bewertungsablaufs

expliziten *Result*-Objekte ein- oder angefügt hat, speichert Grappa diese *Grading-Result* in der Datenbank ab. Der Prozessstatus ist nun *completed*.

Das LMS könnte während dieser Phase durchgängig mit Grappa kommunizieren. Sämtliche Interaktionen mit Grappa wurden asynchron implementiert. Die dementsprechende Logik, wenn *GrdCfgs* oder *Problems* während eines *grade*-Vorganges geändert, oder gelöscht werden, wurde ebenfalls bedacht. Nachdem das LMS (womöglich nach mehreren Polling-Versuchen) den Status *completed* erhält, kann mit der Aufbereitung der Ergebnisdokumente begonnen werden (Abschnitt 23.4) und dem Studierenden zur Verfügung gestellt werden.

23.7 Zusammenfassung und Ausblick

Grappa kann durch die gezeigte Architektur alle Anforderungen von Gradern abbilden. Spezielle Anforderungen können durch Konfigurationsdateien und durch eine dementsprechende Funktion im BackendPlugin realisiert werden. Es ist gelungen eine stabile Middleware zu entwerfen und bisher an zwei Backends und ein LMS anzubinden. Die Anbindung weiterer LMS (derzeit LON-CAPA und ggf. Stud.IP) ist in Planung. Ebenso sollen weitere BackendPlugins für andere Grader erstellt werden, so dass die Unterstützung von weiteren Programmiersprachen erfüllt wird.

Derzeit laufen intensive Arbeiten um das in Kapitel 24 vorgestellte Austauschformat zu unterstützen. Durch die unterschiedliche Darstellung der Aufgaben in Grappa und im Austauschformat bedarf es entweder einer Anpassung im Austauschformat (vgl. [GFB16]) oder eines eigenen Namespaces in den dafür vorgesehenen XML-Räumen. Ein eigener Namespace hätte den Nachteil, dass Grappa-Aufgaben nur in andere Grappa-Instanzen mit den gleichen Gradern übertragen werden können, jedoch nur teilweise graderübergreifend funktionieren würden.

In eCULT+ soll ein Repository für Programmieraufgaben entwickelt werden. Ziel ist es, dieses Repository für Grappa ansprechbar zu machen, so dass der Lehrende im besten Fall nur eine Aufgabe auswählen muss, statt diese manuell anzulegen.

Weiterhin ist geplant, die Middleware ProFormA-Server (Kapitel 22) und die Middleware Grappa (Kapitel 23) mit einer wechselseitigen Schnittstelle auf der Basis des Aufgaben-Austauschformats (Kapitel 24) auszustatten, so dass am Ende alle von einer Middleware unterstützten LMSe mit allen von der jeweils anderen Middleware unterstützten Gradern kombiniert werden können.

Ebenso ist eine gemeinsame Definition der LMS und Grader-Schnittstellen für eCULT+ geplant. Dabei soll vorzugsweise eine Anbindungen weiterer bestehender Grader an Grappa stattfinden.

Grappa wurde zur Unterstützung von parametrierbaren Aufgaben bereits konzeptionell vorbereitet ([GHW15]) und in einer Beta-Version getestet. Es bedarf noch mehrerer Tests und Evaluationen, um diesen Ansatz fest zu integrieren.

Literatur für dieses Kapitel

[GFB16] Robert Garmann, Peter Fricke und Oliver Bott. „Bewertungsaspekte und Tests in Java-Programmieraufgaben für Graja im ProFormA-Aufgabenformat“. In: *DeLFI 2016 – Die 14. E-Learning Fachtagung Informatik*. Bd. 262. LNI. GI, 2016, S. 215–220.

[GHW15] Robert Garmann, Felix Heine und Peter Werner. „Grappa – die Spinne im Netz der Autobewerter und Lernmanagementsysteme“. In: *DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik*. Bd. 247. LNI. GI, 2015, S. 169–181.

[Stö+13] Andreas Stöcker u. a. „Evaluation automatisierter Programmbewertung bei der Vermittlung der Sprachen Java und SQL mit den Gradern *aSQLg* und *Graja* aus studentischer Perspektive.“ In: *DeLFI 2013 – Die 11. E-Learning Fachtagung Informatik*. Bd. 218. LNI. GI, 2013, S. 233–238.

[Stö+14] Andreas Stöcker u. a. „Die Evaluation generischer Einbettung automatisierter Programmbewertung am Beispiel von Moodle und aSQLg.“ In: *DeLFI 2014 – Die 12. e-Learning Fachtagung Informatik*. Bd. 233. LNI. GI, 2014, S. 301–304.