online Übungssystem

Korrekturmoduladapter

Autor:
Immo Schulz-Gerlach, FeU-Softwaretechnik, ZMI

Version:
1.1 – 10. August 2023

FernUniversität in Hagen

Inhaltsverzeichnis

Was ist ein Korrekturmoduladapter?

Dieses Handbuch betrifft lediglich Entwickler von (Vor-)Korrekturmodulen (siehe Rubrik »Entwicklung eigener Korrekturmodule« im Betreuerzugang eines Übungssystem-Kurses).

Ein *Vorkorrekturmodul* wird bei der Einsendung eines Studenten zu einer Teilaufgabe aufgerufen und bekommt im Normalfall alle Eingaben des Studenten zu dieser Teilaufgabe übermittelt, um daraus eine Vorkorrektur zu erzeugen, die dem Studenten anschließend in seiner Quittungsseite direkt angezeigt werden sollte, aber auch später (nach Einsendeschluss-/Heftschließen) in die Korrekturseite mit aufgenommen werden kann.

Ein *Korrekturmodul* wird beim Erzeugen der Korrekturseite (beim Heftschließen) ausgeführt und bekommt normalerweise alle Eingaben des Teilnehmers zu allen eingesendeten Teilaufgaben der Aufgabe übermittelt, um daraus eine in die Korrekturseite aufzunehmende Autokorrektur zu erzeugen und dazu eine Bewertung in Punkten zu berechnen.

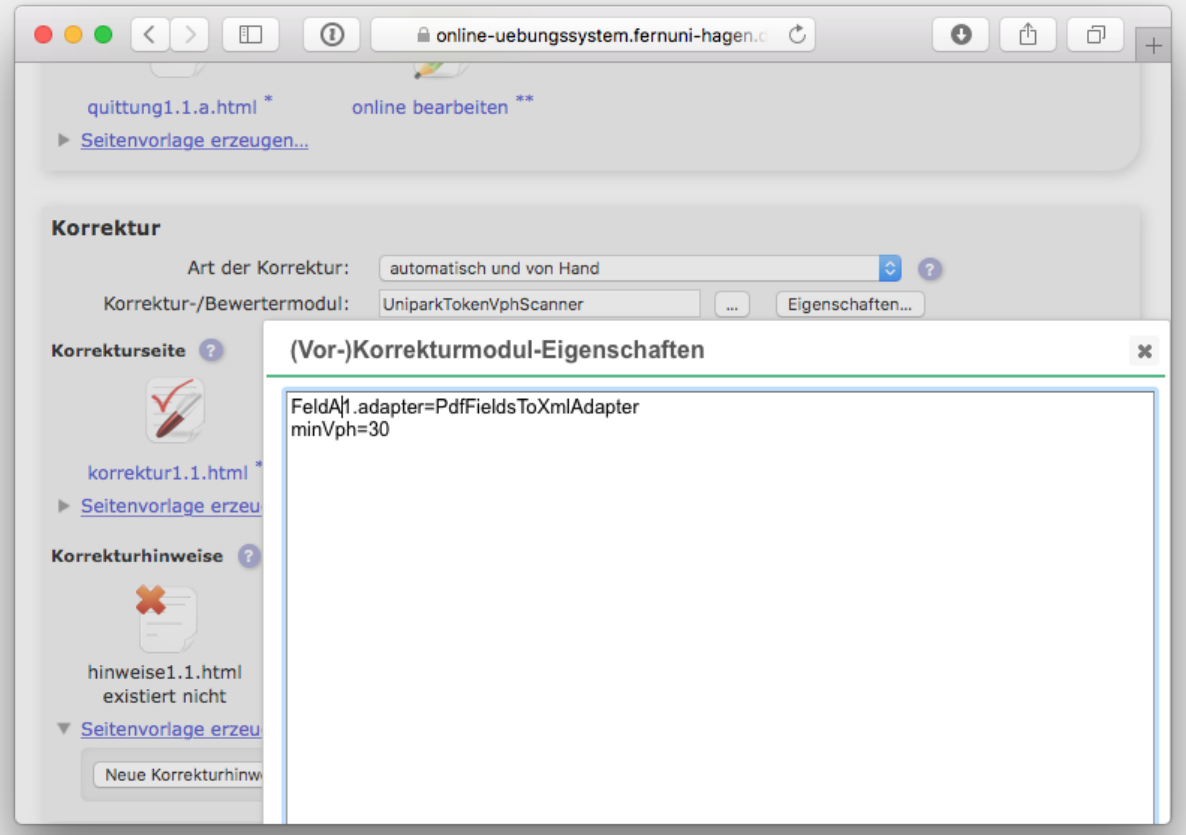
In beiden Fällen werden die Einsendungen des Studenten normalerweise unverändert ans (Vor-)Korrekturmodul übertragen. Bei Einsendungen kann es sich z.B. um einfache Texteingaben, HTML-formatierten Text aus WYSIYG-Editorfeldern oder auch um vom Studenten hochgeladene Dateien handeln.

Ein *Korrekturmoduladapter* ist nun ein im Übungssystem fest integriertes Modul, das „dazwischengeschaltet" werden kann, also Einsendungen eines Studenten entgegennimmt und in ein anderes, ans Korrekturmodul zu sendendes Format konvertiert. Genauer werden Adapter feldweise definiert, d.h. es wird für jedes Eingabefeld einer Aufgabe einzeln festgelegt, ob die darin eigesandten Daten durch einen Adapter konvertiert werden sollen oder nicht, und wenn ja, durch welchen.

Derzeit gibt es genau einen Korrekturmoduladapter, und zwar den im Folgenden beschriebenen `PdfFieldsToXmlAdapter`. Für die Zukunft ist jedoch nicht ausgeschlossen, dass bei Bedarf weitere Adapter hinzukommen.

Adapter-Konfiguration

Um einem **Korrekturmodul** einen Adapter vorzuschalten, sind in der (fortgeschrittenen) Aufgabenerstellung die Eigenschaften zum eingestellten Korrekturmodul zu bearbeiten:



Eigenschaftsdialog für ein Korrekturmodul

Dort ist für jedes Eingabefeld der Teilaufgabe, dessen Eingaben durch einen Adapter konvertiert werden sollen, die Property `FeldA1.adapter` zu definieren, wobei – genau wie bei den Feldnamen im Aufgabenformular und den `$Feld...`-Variablen in der Korrekturschablone – `A` die Teilaufgabenkennung und `1` die Feldnummer darstellt, also diese Kennungen für andere Teilaufgaben oder Felder entsprechend anzupassen sind.

Der Property ist (durch `:` oder `=` getrennt) der Name der zu verwendenden Adapterklasse (s.u.) zuzuordnen.

Die Groß-/Kleinschreibung des Property-Namens spielt übrigens keine Rolle, Sie könnten also auch z.B. `felda1.adapter` schreiben, aber beim Property-Wert (Name der Adapterklasse, z.B. `PdfFieldsToXmlAdapter`) wird die Groß-/Kleinschreibung beachtet, muss also exakt korrekt eingegeben werden.

Solche Adapter-Konfigurationszeilen müssen den Beginn der Eigenschafen bilden, d.h. alle weiteren Einstellungen, die keine Korrekturmoduladapter festlegen, sondern zur Konfiguration des Korrekturmoduls dienen, müssen nach den Adaptereinstellugen folgen!

In obiger Beispielabbildung wird also der `PdfFieldsToXmlAdapter` fürs erste Eingabefeld der ersten Teilaufgabe eingestellt. Erst auf diese Adapterkonfiguration folgt eine zusätzliche Konfiguration spezifisch für das verwendete Korrekturmodul.

Um einem **Vorkorrekturmodul** einen Adapter vorzuschalten, gehen Sie praktisch genauso vor, nur bearbeiten Sie eben die Eigenschaften des jeweiligen Vorkorrekturmoduls. Da die Vorkorrekturmodulkonfiguration spezifisch für eine bestimmte Teilaufgabe vorgenommen wird,

können Sie dabei die Teilaufgabenkennung im Propertynamen weglassen (z.B. `Feld1.adapter` für Zuordnung eines Adapters zum ersten Eingabefeld).

Wenn Sie die Teilaufgabenkennung dennoch mit angeben, *muss* sie natürlich korrekt sein, also die Teilaufgabe bezeichnen, deren Vokorrekturmodulkonfiguration Sie gerade bearbeiten: Angaben zu Feldern anderer Teilaufgaben werden ignoriert. (Auf diese Weise ist es übrigens bei kombiniertem Einsatz eines Moduls sowohl als Korrekturmodul als auch als Vorkorrekturmodul für mehrere Teilaufgaben möglich, die gesamte Korrekturmodulkonfiguration unverändert auch in die Vorkorrekturmodulkonfigurationen zu kopieren.)

PdfFieldsToXmlAdapter

Anwendungsgebiet / Motivation

Dieser Adapter ist vorgesehen für den Fall, dass Studenten PDF-Dateien einsenden sollen und ein (Vor-)Korrekturmodul Informationen aus diesen PDF-Dateien verarbeiten sollen. Mit „Informationen“ ist hier gemeint:

- Eingaben in PDF-Formularfelder (AcroForms) oder
- PDF-Metadaten.

Prinzipiell könnte man zwar ohne Adapter arbeiten und die gesamte eingesandte PDF-Datei ans Korrekturmodul übermitteln, welches eigenständig die auszuwertenden Informationen direkt aus dem PDF extrahiert. Das hat aber verschiedene Nachteile, nicht zuletzt einen lizenzrechtlichen: Das Korrekturmodul muss in der Lage sein, PDF-Dateien zu verarbeiten, und dazu benötigte Libraries sind in der Regel lizenzpflichtig.

Das ZMI verfügt über eine iText <http://itextpdf.com> -Lizenz für den Server des Online-Übungssystems, die den Betrieb dieses auf iText basierenden Adapters abdeckt, während die Nutzung von iText in externen, auf anderen Servern betriebenen Korrekturmodulen nicht von der Lizenz gedeckt wäre. Das war der Hauptgrund für die Entwicklung dieses Adapters, aber natürlich bietet der Adapter noch weitere Vorteile: Die ihn nutzenden Korrekturmodule werden einfacher und das zu übertragende Datenvolumen zwischen Übungssystem- und Korrekturmodulserver fällt deutlich geringer aus.

Verwendung / Konfiguration

Der Adapter kann sowohl auf einzeln hochgeladene PDFs als auch auf ZIP-Dateien mit mehreren PDFs angewendet werden. Damit können Sie ihn insbesondere auch für „Multi-Datei-Uploads <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#multifileupload> “ anwenden, also Aufgabenformulare, in denen ein Student nach dem Upload einer PDF-Datei durch wiederholte Uploads weitere Dateien hinzufügen kann (statt die letzte Einsendung zu überschreiben). Bei Multi-Datei-Uploads werden alle im selben Eingabefeld (Dateiauswahlfeld) nacheinander hochgeladenen Dateien vom Übungssystem zu einer ZIP-Datei zusammengefasst.

Um den Adapter zu nutzen, ist – wie im obigen allgemeinen Abschnitt zur Konfiguration beschrieben – für das Eingabefeld, über das die PDF-Datei(en) einzusenden war(en), eine entsprechende Adapterzuweisung am Beginn der (Vor-)Korrekturmoduleinstellungen einzufügen, z.B.: `FeldA1.adapter=PdfFieldsToXmlAdapter`.

Eine weitere Konfiguration des Adapters ist nicht möglich/erforderlich, er sammelt Informationen (Metadaten und Formulareingaben) zu allen eingesandten PDFs und stellt diese dem Korrekturmodul als XML zur Verfügung. Wenn sich das Modul nur für bestimmte dieser Daten (z.B. nur für bestimmte Formularfelder und überhaupt nicht für Metadaten) interessiert, muss es die übermittelten (XML-)Informationen selbst entsprechend filtern.

Funktion

Der Adapter bekommt die Einsendung des Studenten aus einem Eingabefeld übermittelt und trifft zunächst folgende Fallunterscheidung:

1. Handelt es sich um eine PDF-Datei?

2. Handelt es sich um eine ZIP-Datei?
3. Oder ist es eine sonstige Einsendung (anderes Dateiformat oder gar keine Datei, z.B. Texteingabe)?

ad 1.: PDF-Datei

Der Adapter nimmt die PDF-Datei entgegen und durchsucht sie…

1. nach Formularfeldern (AcroForm Fields) und
2. nach Metadaten.

Diese werden dann in einer XML-Struktur der folgenden Art zusammengefasst:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pdfdata>
 <file name="test.pdf">
  <meta name="CreationDate">D:20160614064347Z</meta>
  <meta name="Keywords"/>
  <meta name="Producer">Mac OS X 10.11.5 Quartz PDFContext</meta>
  <meta name="Author">Immo Schulz-Gerlach</meta>
  <meta name="Creator">Safari</meta>
  <meta name="ModDate">D:20161012130815+02'00'</meta>
  <meta name="Title">Der Titel</meta>
  <meta name="Subject">Hallo Welt</meta>
  <formfield name="first">Hello</formfield>
  <formfield name="second">World</formfield>
 </file>
 <file name="defekt.pdf">
  <invalid>InvalidPdfException: Rebuild failed: trailer not found.; Original
message: xref subsection not found at file pointer 184968</invalid>
 </file>
</pdfdata>
```

Der erste `file`-Eintrag stellt beispielhaft den XML-Aufbau von in einem gültigen PDF gefundenen Nutzinformationen dar, der zweite ist ein Beispiel für den Fall, dass die PDF fehlerhaft war und die Informationsextraktion daraus nicht funktioniert hat. In dem Fall findet sich genau ein `invalid`-Kindelement, dessen Inhalt eine Fehlermeldung ist (die Sie ignorieren oder irgendwie ausgeben können).

Dieses XML wird dann (mit Content-Type `text/xml`) an Stelle der Dateieinsendung ans (Vor-)Korrekturmodul übertragen.

Für Interessierte: Das XSD (XML Schema Definition) zu dieser XML-Struktur lautet wie folgt:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
 <xs:element name="pdfdata">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="0" maxOccurs="unbounded" ref="file"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="file">
  <xs:complexType>
   <xs:choice>
    <xs:element name="invalid" type="xs:string"/>
    <xs:sequence>
     <xs:element minOccurs="0" maxOccurs="unbounded" name="meta"
type="namedvalue"/>
     <xs:element minOccurs="0" maxOccurs="unbounded" name="formfield"
type="namedvalue"/>
    </xs:sequence>
   </xs:choice>
   <xs:attribute name="name" use="required" type="xs:string"/>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="namedvalue">
  <xs:simpleContent>
   <xs:extension base="xs:string">
    <xs:attribute name="name" use="required" type="xs:string"/>
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:schema>
```

ad 2.: ZIP-Datei

Ist die Einsendung eine ZIP-Datei, wird diese nach allen in ihr (in beliebigen Unterverzeichnissen) enthaltenen PDF-Dateien durchsucht und am Ende ein XML mit derselben Struktur wie oben unter *ad 1.* gezeigt gesendet, nur eben mit einem `<file…>`-Element pro im ZIP gefundener PDF-Datei. Das können also mehrere `<file…>`-Elemente, genau eines oder – falls das ZIP keine PDF-Dateien enthält – auch gar kein `<file…>`-Element sein. (Daher steht auch in obigem XSD, dass 0 bis unendlich viele `file`-Elemente in einem `pdfdata`-Element auftreten dürfen.)

ad 3.: Sonstige Einsendung

In diesem Fall bleibt der Adapter untätig und übermittelt die Feldeinsendung unverändert an das Korrekturmodul.

Falls z.B. das Eingabefeld, dem der Adapter zugeordnet wurde, gar kein Dateiupload-Feld, sondern ein Textfeld ist, bleibt die Adapterzuordnung somit unwirksam, ist aber auch unschädlich. Handelt es sich jedoch tatsächlich um ein Dateiupload-Feld, so kann es immer passieren, dass ein Student eine ungültige Datei hochlädt – selbst bei Dateityp-Einschränkung <https://online-uebungssystem.fernuni-hagen.de/download/FileUploadAccept/FileUploadAccept.html> des Feldes auf PDF- und/oder ZIP-Dateien ist das

nicht ausgeschlossen.

Das Korrekturmodul muss daher selbst unterscheiden, ob es ein vom Adapter erzeugtes XML-Dokument erhalten hat oder etwas ganz anderes – und wie in letzterem Fall seine Vorkorrektur aussehen sollte. Als Beispiel für ein solches Abfangen ungültiger Einsendungen sei hier eine Hilfsmethode (der Vollständigkeit halber mit zugehörigen Importen) aus einem existierenden Java-Korrekturmodul zitiert:

```
import WebAssign.korrektur.adapter.pdfscanner.xml.File;
import WebAssign.korrektur.adapter.pdfscanner.xml.Pdfdata;
import WebAssign.korrektur.adapter.pdfscanner.xml.PdfdataXmlParser;
import WebAssign.korrektur.soapserver.FeldEinsendung;
import WebAssign.korrektur.soapserver.TeilaufgabenEinsendung;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;
import javax.xml.bind.JAXBException;

private List<File> getFilesFromTae(TeilaufgabenEinsendung tae) throws
KorrekturServiceException {
 try {
  return tae.getFeldeinsendungen().stream()
   .filter(FeldEinsendung::isDateiEinsendung)
   .filter(fe -> fe.getMimetype().startsWith("text/xml"))
   .map(FeldEinsendung::getStringEinsendung)
   .map(s -> {
    try {
     return PdfdataXmlParser.parse(s);
    } catch (JAXBException ex) {
     throw new WrappedCheckedException(ex);
    }
   }).map(Pdfdata::getFile)
   .flatMap(List::stream)
   .collect(Collectors.toList());
 } catch (WrappedCheckedException ex) {
  throw new KorrekturServiceException("XML mit Daten zu eingesandten PDF-
Formularen konnte nicht erfolgreich geparst werden!", ex);
 }
}
```

Dieses Beispiel greift nicht auf ein bestimmtes Eingabefeld (wie `FeldA1`) zu, sondern betrachtet eine Liste aller eingesandten (und ggf. vom Adapter modifizierten) Einsendungen und filtert dort alle Nicht-XML-Einsendungen aus. Das ist natürlich nicht für jedes Korrekturmodul in dieser Form sinnvoll.

Neben dem Ausfiltern von Nicht-XML-Einsendungen zeigt das Beispiel auch noch das Aufrufen des im nächsten Abschnitt vorgestellten XML-Parsers für von diesem Adapter generierte XML-Dokumente.

XML-Parser-Library für Java

Ein mit diesem Adapter zu kombinierendes Korrekturmodul muss natürlich das vom Adapter erzeugte XML parsen können. Um den XML-Parser nicht selbst entwickeln zu müssen, stellt das Online-Übungssystem zumindest für in Java implementierte Korrekturmodule eine fertige, aus dem XSD dieses XML-Formats generierte Parser-Library `korrekturmoduladapter.pdfscanner.xmlparser.jar` als Download zur Verfügung (im Betreuerzugang Ihres Übungssystem-Kurses unter »Entwicklung eigener Korrekturmodule«, Rubrik »SOAP-WebService«, wo auch die Java-Sourcen mit Demoprojekten zur SOAP-Korrekturmodulmodul-Entwicklung zum Download stehen).

Die Verwendung ist einfach:

- Die statische Methode `PdfdataXmlParser.parse(s)` bekommt den XML-String `s` aus der Feldeinsendung übergeben und gibt dafür ein Objekt der Klasse `Pdfdata` zurück, die das XML nach oben abgebildetem Aufbau repräsentiert.
- Das `Pdfdata`-Objekt hat nur eine Methode: `List<File> getFile()` gibt eine Liste von `File`-Objekten zurück, von denen jedes die Daten zu einer PDF-Datei repräsentiert, vgl. `<file…>`-Element in obigem XML.
- Ein `File`-Objekt wiederum verfügt über vier Getter zum Attribut und den möglichen Kindknoten:
  - `String getName()`: Dateiname der PDF-Datei, vgl. `name`-Attribut des `<file>`-Elements
  - `List<Namedvalue> getFormfield()`: Liste der gefundenen Formularfelder, vgl. `<formfield>`-Kindelemente in obigem XML
  - `List<Namedvalue> getMeta()`: Liste der gefundenen Metadaten, vgl. `<meta>`-Kindelemente in obigem XML.
  - `String getInvalid()`: Wenn die PDF fehlerhaft war, gibt diese Methode eine Fehlermeldung als String aus (während die beiden Listen oben leer bleiben). Zu intakten PDFs ist die Rückgabe `null`!
- Die Einträge für Formularfelder ebenso wie für Metadaten sind also von derselben Klasse `Namedvalue`, die wiederum zwei Getter-Methoden besitzt:
  - `String getName()`: Name des Metafeldes bzw. Formularfeldes
  - `String getValue()`: Wert des Metafelds bzw. Eingabe in einem Formularfeld.

Der Aufruf der `parse`-Methode wird bereits im Quellcodeausschnitt im vorigen Abschnitt demonstriert.