Sven Hodapp

Exposé für eine Masterarbeit

Arbeitstitel: Entwicklung eines Dokumenten-Projektionseditors für Text und Graphen auf Basis von Webtechnologie

Institution: Hochschule Konstanz
Bereich: Fakultät Informatik
Version: 16. Januar 2014

1 Problemstellung

Die Erstellung von qualitativ hochwertigen Dokumenten ist keine einfache Sache. Das Computerzeitalter konnte zwar schon viel verbessern, aber es gibt noch immer Situationen, in denen die Dokumentenerstellung sehr mühselig werden kann.

Zum einen kostet es u.U. sehr viel Zeit, Querreferenzierungen im Dokument zu pflegen und konsistent zu halten. Hierfür hat insbesondere LaTeX eine solide Lösung, jedoch stößt man auf der Meta-Ebene (der LaTeX Code selbst) wieder auf das Problem. Beispiel: Man definiert ein Label und referenziert an anderen Stellen darauf. Wird das Label irgendwann umbenannt (weil sich z.B. die Überschrift geändert hat), muss es an allen referenzierten Stellen auch umbenannt werden (klassisches Refactoring) – und das geschieht leider nicht automatisch.

Zum anderen verstehen die Anwendungen oft nur die Domäne „Dokument“, wenn man z.B. chemische Formeln zeichnen will, müssen diese über externe Ressourcen hinzugefügt werden. Dabei gibt es keinen wirklichen Zugriff mehr auf die Meta-Informationen, die eigentlich durch das Domänenmodell – die chemische Formel – mitgeliefert wird. Das kann wiederum zu inkonsistenten Dokumenten führen. Wenn z.B. nachträglich an der chemischen Formel etwas geändert wird und nur die Abbildung von Benutzer aktualisiert wurde, dann sind u.U. die Verweise (z.B. auf die Masse des Moleküls der chemischen Formel) in den beschreibenden Sätzen nicht mehr korrekt; also fehlt das semantische Verständnis seitens des Dokuments.

Man beachte auch andere Punkte: z.B. Transformation in Formate wie Markdown, welche unter anderem gut für *distraction free writing* sein können, aber auch auch eine bessere Versionskontrolle mit Werkzeugen wie *git* ermöglichen. Zudem wäre Zugänglichkeit und Interaktion ohne zusätzliche Hürden zu einer vollwertigen Programmiersprache wünschenswert, um beispielsweise direkt während der Dokumentenerstellung Datenbankzugriffe bzw. algorithmische Auswertungen vorzunehmen.

Das wirft die Frage folgende Frage auf: *Kann man ein System erschaffen, welches die o.g. Defizite aufhebt?*

1.1 Vorhandene Vorarbeit

In meiner Bachelorarbeit habe ich mich schon mit ähnlichen Fragestellungen beschäftigt und möchte diese weiterentwickeln. Im Zuge dieser Bachelorarbeit ist bereits eine JavaScript Bibliothek entstanden, mit der Printdokumente im Webbrowser (weiterhin Browser-Dokument genannt) „gesetzt“ werden können. Zudem wurde eine, an TeX angelehnte, interne Scala DSL entwickelt, mit der solche Dokumente generiert werden können.

2 Ziele

Aus den Argumenten der Problemstellung und den vorhandenen Vorarbeiten ergeben sich folgende Ziele:

Es soll einen Prototyp erschaffen werden, welcher ermöglicht, das Browser-Dokument direkt im Browser ohne Umweg über die DSL zu modifizieren bzw. zu erweitern. Darüber hinaus soll das Browser-Dokument um Unterstützungen für domänenspezifische Editoren erweitert werden. Diese Editoren können z.B. eine grafische Repräsentation über ein Objekt ihrer Domäne sein und dieses auf (semantische) Korrektheit überprüfen. Bei der Dokumenten-Erstellung soll auch eine vollwertige Programmiersprache verfügbar sein, um das Dokument semantisch halten zu können.

Schlussendlich soll man ein Dokument erhalten, welches einfach und schnell erstellbar und änderbar ist, wie man es von z.B. Google Docs kennt. Jedoch soll es zudem relativ einfach um verschiedene Domänen erweiterbar sein, erreicht durch domänenspezifische Editoren. Ein Beispiel für einen solchen domänenspezifischen Editor ist Ketcher[1]. Er würde einem Dokument die Fähigkeit hinzufügen, chemische Strukturformeln darzustellen. Da unter der Dokumentengenerierung eine vollwertige Programmiersprache liegt, hat der Benutzer auch Zugriff darauf und kann sein mit der Domäne angereichertes Dokument beliebig automatisieren.

Es wäre also wie wenn man Google Docs, LaTeX, eine vollwertige Programmiersprache mit einer Prise projectional editing in einen Mixer gibt...

[1] JavaScript basierter chemischer Strukturediteor. http://ggasoftware.com/opensource/ketcher

2.1 Konkrete Ziele

Primäres Ziel:

- Funktionstüchtige Schleife, um ein Dokument projektiv zu editieren: Anzeige und Editierung des Dokuments im Browser wird in ein abstraktes Modell abgebildet. Aus dem Modell wird ggf. Code generiert und in das laufende System eingebracht, so dass das aktualisierte Modell wieder im Browser zur weiteren Editierung bereitsteht.
- Aus dem abstrakten Modell heraus sollen auch andere Ausgabeformate produzierbar sein, statt Webtechnologie auch z.B. Markdown oder Quellcode (beispielsweise als Scala DSL), welche gut zur Versionskontrolle sind. (Projektive Eigenschaft)

Sekundäres Ziel:

- Einbindung eines domänenspezifischen Editors mit Anbindung zur darunterliegenden Programmiersprache.
- Einfacher Fall: Ketcher für Chemiedarstellung.
- Komplexer Fall: Spray[2] für beliebige Domänen. Neben dem Eclipse-Plugin existiert hier auch schon eine Arbeit, die den Spray Editor ins Web bringt. Daher wäre eine Verwendung von Spray als Domäneneditor innerhalb eines Dokuments besonders spannend; weil z.B. die Dokumentierung des Modells enorm erleichtert würde.

Tertiäres Ziel:

- Erweiterungen und Weiterentwicklungen am Browser-Dokument Framework.
- Eine Umstrukturierung muss es durch die o.g. Anforderungen ohnehin geben, aber gemeint sind eher Zusatzfunktionen wie Fußnoten oder neue konzeptuelle Ideen wie z.B. scala-js[3].

[2] Framework zur Erstellung visueller DSL Editoren (modellgetriebene Softwareentwicklung). https://code.google.com/a/eclipselabs.org/p/spray/

[3] Scala zu (client-side) JavaScript Compiler. https://github.com/lampepfl/scala-js

2.2 Hypothesen

1 Aktoren sind ein geeignetes und elegantes Mittel, um einen AST[4] für projectional editing bereitzustellen.

2 Aktoren sind ideal um die Graphenstruktur des Dokuments abzubilden. Das heißt das Aktorensystem kann auch Referenzen[5], die es innerhalb des Dokuments gibt, auflösen. Wenn der Dokumentenbearbeiter Dokumentenentitäten[6] bearbeitet, bedeutet das, dass lediglich ein Knoten im Graphen ausgetauscht werden muss.

2.3 Wissenschaftliche Fragestellungen

Fragen die von wissenschaftlichen Interesse sein könnten und während der Masterarbeit in Erwägung gezogen werden:

1. Kann eine Semantik, ähnlich der von projectional editing workbenches, auf die Domäne Dokumentenerstellung bzw. Dokumente sinnvoll übertragen werden? Welche Möglichkeiten eröffnen sich dadruch?
2. Kann man mehr Semantik schon beim Schreiben des Dokuments erreichen? Mehr Semantik als mit bisherigen Lösungen?
3. Können Aktoren[7] vernünftig als Modellinstanz[8] für ein solches Dokument dienen?
4. Kann diese Modellinstanz zur Laufzeit modifiziert werden, so dass *projectional editing* ermöglicht wird?
5. Eignen sich Aktoren als flexibler Codegenerator?
6. Ist es möglich, dass zur Laufzeit Code generiert, compiliert und wieder in das laufende (Aktor)-System eingebracht wird?

[4] Abstract Syntax Tree

[5] z.B. Verweise auf Bilder automatisch in eine Nummerierung umwandeln vlg. LaTeX.

[6] z.B. Bilder, Textabschnitte, Überschriften, etc.

[7] Ein Software Transactional Memory System. Eine populäre Implementierung ist `http://akka.io`.

[8] Ein formales Modell, welches eine Instanz eines Metamodells ist.

7. Kann ein solches Dokumentenerstellungssystem als Plattform zur Kooperation fungieren? Ist das dann eine neue Klasse von Dokumenten bzw. eine neue Art von Desktop-publishing?

8. Wie kann eine Anbindung an domänenspezifische Editoren stattfinden? Wie kann von der Programmiersprache aus auf die Objekte des Editors zugegriffen werden? Was eröffnen sich dadurch für Möglichkeiten?

9. Kann der Spray Modellierungseditor so eingebunden werden, dass vom Dokument aus direkt und konsistent auf das Modell referenziert werden kann? Nach Möglichkeit mit den Eigenschaften von projectional editing?

10. Da Modelle, die mit Spray modelliert wurden, u.U. sehr große Diagramme ergeben, kann es aus Dokumentationsinteresse wichtig sein, nur Ausschnitte des Diagramms im Dokument aufzuführen, um diese zu erklären. Ist es möglich aus einem großen Modell verschiedene Teilausschnitte im Dokument zu präsentieren und zu referenzieren?

Zu (1) Mehr Semantik durch saubere Referenzierung von Objekten (z.B. das Objekt `caffeine` vom Typ `Chemistry`, kann Methoden wie `.editable_repr` (ergibt einen Moleküleditor direkt im Dokument), `.mass` oder `.IUPACName` anbieten; oder sogar chemisches Rechnen mit anderen Chemisty Objekten ermöglichen). Damit die Variablen/Objekte immer bei Änderungen/Refactoring synchron sind, werden diese auch in Aktoren abgebildet, so dass jede Variable intern einem eindeutigen Aktor entspricht. Beispiel: wenn z.B. das `caffeine` Objekt in `caffeine_x` umbenannt wird, wird überall im Dokument diese Variable transparent umbenannt; das wäre dann eine Eigenschaft des projectional editing.

Zu (2) Das Dokument kann verschiedene Domänen verstehen, zudem konsistente Referenzierung von z.B. Abbildungen, Erkennung von speziellen Arealen in Dokumenten wie z.B. Titelblatt, Hauptinhalt oder „hier wird gerade zitiert"-Bereichen. Durch Transformation in verschiedene Formate oder Ansichten, wie Printlayout (mit Webtechnologie), normales Webseitenlayout oder mindmap-artig, kann eine andere Übersicht über das Dokument erhalten werden. Dadurch können ggf. Beziehungen, die es innerhalb des Dokuments oder zu anderen Dokumenten existieren, besser aufgedeckt werden.

Zu (3) und (4) Aktoren fungieren als Modellinstanz und gleichzeitig als Generatoren in verschiedene Formate wie Markdown, JSON oder Quellcode. Die Aktoren können quasi als AST angesehen werden, welcher als Basis für projectional editing dienen kann.

Zu (6) Ermöglicht es, in der Dokumentenerstellungsumgebung Technologien wie Python und Scala kooperativ zu nutzen. Das heißt, es wird eine Art Metaprogramming ermöglicht – der eine Code beeinflusst zu seiner Laufzeit den Quellcode des Anderen. Solch ein Verhalten könnte griffig als „kooperative Codegenerierung“ bezeichnet werden.

3 Methode

Es soll ein funktionierender Prototyp programmiert werden. Bevorzugte Programmiersprachen sind dabei Scala, Python oder JavaScript. Darauf folgt eine Analyse des Prototyps, ob und wie die Fragestellungen gelöst werden konnten und welche Qualität die gefunde Lösung hat.

Zuvor muss jedoch herausgefunden werden, ob Aktoren sich als AST eignen oder ob alternativ dazu eine (NoSQL) Datenbank besser geeignet ist.

4 Gliederungsentwurf

1. Einleitung
2. Technologien (Grundlagen und Grundbegriffe)
    - 2.1. Gestaltungsregeln wiss. Dokumente
    - 2.2. Projectional Editing
    - 2.3. Aktoren
3. Entwicklung des Prototypen (Praxisteil)
    - 3.1. Anforderungen
    - 3.2. Architektur und Design
    - 3.3. Benutzeroberfläche
    - 3.4. Abstrakter Syntax Baum

    3.5. Generierung und Persistenz
    3.6. Verbindungscode zum Domäneneditor
4. Ergebnisse und Analyse
5. Fazit
6. Ausblick

5 Zeitplan

An unserer Fakultät liegt der vorgesehene Arbeitsaufwand für eine Masterarbeit bei vier bis sechs Monaten und darf maximal um drei Monate verlängert werden, daher folgender Zeitplan:

Woche	Phase	Kommentar
1–4	Recherche	Literatur, Ideen, Programmierversuche
4–8	Architektur	Anforderungsanalyse, Architekturplan
8–16	Prototyp	Programmierung, Tests
16–20	Niederschrift	
20–24	Abschluss	Korrekturlesen, Ausbessern