Funktionale Spezifikation von Websites

Torsten Gipp

Universität Koblenz-Landau
tgi@uni-koblenz.de

Abstract: Websites sind komplexe Software-Artefake, deren Erstellung und Pflege ohne ein wichtiges Hilfsmittel nicht mehr möglich ist: Abstraktionen, Modelle. Der vorliegende Text beschreibt einen Ansatz, nach dem ein großer Teil der Modelle als funktionale Programme erstellt werden. Hierdurch werden die Modelle zu ausdrucksstarken Spezifikationen, die sogar direkt ausgeführt werden können. Zur Implementierung verwenden wir die weit verbreitete funktionale Programmiersprache Haskell.

1 Einführung

Eine Website, auch Web-Präsenz genannt, ist eine Sammlung einzelner, miteinander durch Hyperlinks verknüpfter Dokumente, die mit Hilfe von Web-Technologien von einem Server zur Verfügung gestellt und mit einem Webbrowser abgerufen und angezeigt werden können. Die Erstellung einer solchen Website ist eine große Herausforderung. Realistisch große Websites zu erstellen ist ohne technische Hilfsmittel kaum möglich. Hier ist ein wohl-strukturiertes Vorgehen erforderlich, um der Komplexität Herr zu werden.

Die Qualitätsanforderungen an eine Website sind sehr hoch. Sie soll in der Regel einem großen Publikum zugänglich gemacht werden, so dass von einem Fehler viele Benutzer betroffen wären. Insbesondere bei sicherheitskritischen Anwendungen steht die Qualität ganz besonders im Vordergrund, zum Beispiel bei Websites, mit denen finanzielle Transaktionen getätigt werden. Gleichzeitig mit der Erfüllung der Qualitätsanforderungen wird eine immer kürzere Entwicklungszeit gefordert, und es muss gewährleistet sein, dass Erweiterungen schnell vorgenommen werden können.

All diesen Anforderungen lässt sich nur mit soliden, ingenieurmäßigen, softwaretechnischen Verfahren und Methoden begegnen. Ein gängiges Vorgehen besteht in der Verwendung von *Modellen*, also zweckbezogenen Abstraktionen. Die Idee besteht darin, das gewünschte Endprodukt auf einer abstrakten Ebene zu beschreiben und die gewünschte Website aus diesen Beschreibungen automatisiert zu *generieren* (vgl. z.B. [KPRR04]).

Der in diesem Text vorgestellte Ansatz vereint nun die Vorteile des Modell-basierten Vorgehens mit denen der *funktionalen Programmiersprachen*. Dies sind Programmiersprachen, die dem funktionalen Paradigma folgen, also Algorithmen ausschließlich in Form von (mathematischen) Funktionen beschreiben. Die Zusammenführung der Modell-basierten und der funktionalen Sicht ist der Kerngedanke dieser Arbeit, vorgestellt in der Dissertation [Gip06]. Darauf sei auch für weiterführende Informationen verwiesen, da sich der

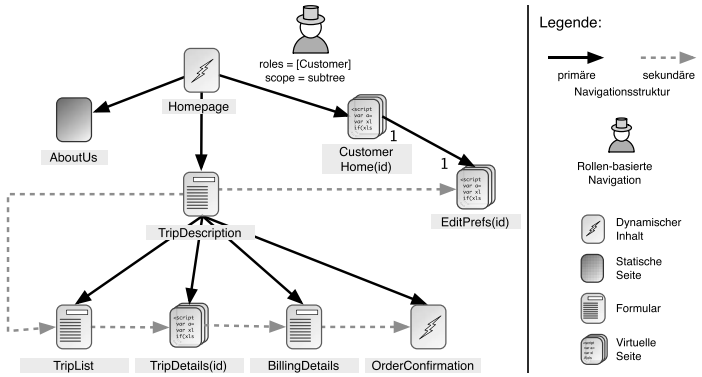

Abbildung 1: Navigationsstruktur. Nach: [Gip06, S. 141]

vorliegende Text aus Platzgründen auf das absolut Wesentliche konzentriert.

Der folgende Abschnitt führt nun zunächst ein Anwendungsbeispiel ein, auf das im weiteren Verlauf des Textes immer wieder zurückgegriffen wird. Danach stellt Abschnitt 3 die Vorzüge des Ansatzes in einer Übersicht zusammen. Der Fokus geht dann zu den Modellen, die im Zusammenspiel eine Website beschreiben und ihre automatische Erzeugung ermöglichen. Den aktuellen Stand der Forschung fasst Abschnitt 5 zusammen, weitere Literaturhinweise sind zudem direkt im Text vermerkt. Eine kurze Zusammenfassung und einen Ausblick auf weitere Forschungs- und Entwicklungsarbeit finden Sie abschließend in Abschnitt 6.

2 Anwendungsbeispiel

Wir betrachten im Folgenden ein (fiktives) System zur Reisebuchung, das so genannte TAS, kurz für “travel agency system”. Einige grundlegende Dienste dieses Systems sind über eine Website zugreifbar, die hier nun auszugsweise näher betrachtet wird. Mit Hilfe der Website kann sich ein Kunde eine Flugreise zusammenstellen und den Flug auch gleich buchen. Dazu gibt er zunächst die grundlegenden Daten wie Reisetermin und Reiseverlauf an, und das System bietet ihm daraufhin eine Auswahl mit passenden Flügen. Der Kunde kann sich Details zu den einzelnen Flügen anschauen, kann abschließend einen Flug wählen und wird dann auf weiteren Seiten durch den Buchungsvorgang geleitet.

Abbildung 1 zeigt die *Navigationsstruktur* dieser Website. Die Rechtecke mit den abgerundeten Ecken stehen für einzelne Seiten, die durchgezogenen Pfeile geben die so genannte *primäre* Navigationsstruktur an, die eine Hierarchie der Seiten definiert. Die genaue Syntax des Diagramms wird in Abschnitt 4.2 näher beschrieben.

Es gibt eine Einstiegsseite, Homepage, von der aus der Kunde auf die weiteren Seiten springen kann. So kann er beispielsweise auf der Seite TripDescription die grundlegenden Angaben zu seinem gewünschten Flug machen, wie z.B. das Abflugdatum und den Ziel-

Abbildung 2: Eingabe der Reisedaten (Seite TripDescription)

flughafen. Auf der Seite TripList bekommt er dann eine Liste mit möglichen Flügen präsentiert, von denen er sich einen auswählen kann. Auf der Seite TripDetails findet er dann weitere Informationen zu dem ausgewählten Flug. Ist er zur Buchung bereit, werden die abschließenden Schritte auf den Seiten BillingDetails und OrderConfirmation abgearbeitet.

Die einzelnen Seiten sind aus Bausteinen zusammen gesetzt, wie z.B. Textblöcke, Bilder oder Eingabeformulare. Die Seite TripDescription enthält beispielsweise ein solches Formular. Es ist in Abbildung 2 zu sehen. Das Formular wird aus einer abstrakten Beschreibung heraus generiert, die in Form eines funktionalen Programms vorliegt. Hier der Code:

```
tasTripDescriptionForm :: StatefulPageFunc
tasTripDescriptionForm _ = do
  (graph, session) ← get
  return $
    Form "TripDescriptionForm" []                                          (1)
      [ Text "From:"
      , Field "originCity" [] (OptionListField $ getOriginCities graph) [] ""   (2)
      , Text "To:"
      , Field "destCity" [] (OptionListField $ getDestCities graph) [] ""
      , Text "Departure:"
      , Field "dateOfDeparture" [] SimpleField [] ""
      -- einige ähnliche Zeilen übersprungen
      , Text "Sorting Order:"
      , Field "sortingPreference" []
        (OptionListField $ map show possibleTripSortingPreferences) [] ""
      , Field "submit" [] SubmitField [] ""
      ]
      tasTripList []
```

Die Bestandteile des Formulars, z.B. die Texte und die Eingabefelder, finden sich im Code wieder. Auf die Syntax kann an dieser Stelle aus Platzgründen nicht weiter eingegangen werden.

Die formale Repräsentation hat einige entscheidende Vorteile. Sie erschließen sich aus der Mächtigkeit der funktionalen Programmiersprache, die beispielsweise an jeder Stelle beliebige Funktionsaufrufe erlaubt, und dem Zusammenspiel mit den anderen Beschreibungen. Zudem kann das Formular direkt aus der formalen Beschreibung erzeugt werden und die Beschreibung ist sehr nahe an der Aufgabenstellung. Im folgenden Abschnitt finden Sie eine Gesamtübersicht über die angesprochenen Vorteile.

3 Vorzüge des vorgestellten Ansatzes

Die Kombination der modellbasierten Sichtweise und der funktionalen Programmiersprachen hat insbesondere folgende Vorzüge (vgl. [GE07]):

- *Deklarative Beschreibungen in Verbindung mit Modellen ermöglichen ausführbare Spezifikationen.*
  Die funktionalen Programme, die wir verwenden, um die einzelnen Teile der Website zu beschreiben, sind gleichzeitig ihre Spezifikation. In Kombination mit den Modellen sind die Programme *ausführbar* und ihre Ausführung ergibt als Resultat die gewünschte Website. Dies ist möglich, weil die Spezifikation hinreichend präzise ist.

- *Die Formalisierung der Anforderungen erfolgt in möglichst frühen Phasen.*
  Wir schlagen vor, die Anforderungen (requirements) an die Website so früh wie möglich zu formalisieren, d.h. so präzise wie möglich zu notieren. Dies verhindert eine Abweichung der Spezifikation von den Anforderungen. So wird die Website später auch tatsächlich so, wie es die Anforderungen vorsehen. Die Formalisierung erfolgt zudem *deklarativ*, so dass sich viele Anforderungen ohne großen Aufwand formalisieren lassen.

- *Es können jederzeit neue Abstraktionsebenen eingeführt werden.*
  Der Komplexität einer Website kann nur durch Abstraktion begegnet werden. Funktionale Programmiersprachen haben genau hier eine ihrer Stärken, denn es ist sehr einfach, jederzeit neue Abstraktionen selbst zu definieren. Ein Beispiel sind Funktionen höherer Ordnung, mit denen beispielsweise eigene Kontrollstrukturen definiert werden können. Auch Bibliotheken mit Kombinatoren (z.B. [Han06]) und so genannte “domain-specific embedded languages” (DSEL) (z.B. [Thi05]) helfen dabei, Abstraktionen zu schaffen und unnötige Implementierungs-Details zu verbergen.

- *Eine strikte Trennung der Belange verhilft zu mehr Überblick.*
  Wir unterteilen die Modellierungsaufgabe in sechs Teile, nämlich in den Content, die Navigation, die Seiten, die Queries und Updates, die Präsentation und die Dynamik. So kann den spezifischen Herausforderungen der einzelnen Teile besser begegnet werden.

- *Unterstützung für Tests und Simulation.*
  Funktionale Programme lassen sich sehr gut testen. Testfälle lassen sich direkt aus den Anforderungsdokumenten herleiten. Gleichfalls ist eine Simulation des Benutzerverhaltens realisierbar, denn die einzelnen Schritte bei der Bedienung einer Website sind nichts anderes als eine Folge von Funktionsaufrufen mit konkreten Parametern.

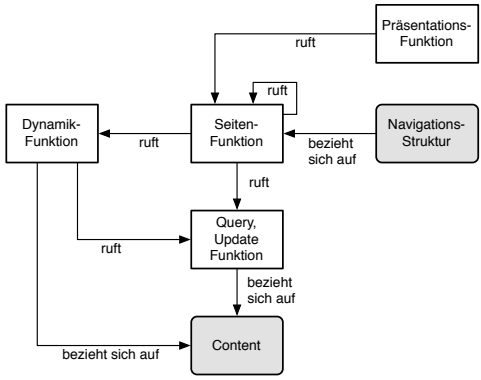

Abbildung 3: Überblick über die Modelle. Nach: [Gip06, S. 60]

4 Vorstellung der Modelle

Wir unterteilen die Modellierung einer Website, dem Prinzip der "Trennung der Belange" folgend, in sechs verschiedene Teilmodelle. Abbildung 3 zeigt die Teilmodelle und ihren Zusammenhang. Jedes Rechteck repräsentiert ein Teilmodell. Rechtecke mit abgerundeten Ecken stehen für 'traditionelle' konzeptuelle Modelle. Die normalen Rechtecke bezeichnen jeweils ein Modell, das als funktionales Programm vorliegt.

Der *Content* (Inhalt) wird durch ein konzeptuelles Modell beschrieben. Man betrachtet dazu die *Anwendungsdomäne* und identifiziert die relevanten Konzepte und die Beziehungen zwischen ihnen. Für die Beispielanwendung TAS sind Konzepte wie "Flugreise" oder "Kunde" relevant. Die Konzeptualisierung der Anwendungsdomäne hilft dabei, eine präzise Begriffswelt zu schaffen, auf die die folgenden Modelle aufbauen können.

Die *Navigationsstruktur* definiert die Verknüpfung der einzelnen Seiten.

Die *Seiten* sind das zentrale Element einer Website. Jede Seite ist durch eine Funktion definiert, deren Auswertung eine abstrakte Beschreibung der fertigen Seite ergibt. Die Funktion stellt die Seite aus einzelnen Fragmenten zusammen.

Die *Queries und Updates* haben den Zweck, Inhaltsbausteine zu liefern bzw. Inhalte zu verändern. Sie Seitenfunktionen greifen auf sie zurück, um auf den Content zuzugreifen. Queries und Updates werden ebenfalls als Funktionen definiert.

Die *Präsentation* kümmert sich um die konkrete Darstellung der Website. Alle anderen Modelle behandeln die Website unabhängig von jedweden gestalterischen Aspekten. Die Präsentation ist eine Abbildung der Seiten in ein konkretes Ausgabeformat und wird ebenfalls in Form einer Funktion notiert.

Die *Dynamik* einer Website ist das Anwendungs-spezifische Verhalten, die so genannten "Geschäftslogik". Für das TAS wäre dies z.B. die Durchführung einer Reisebuchung oder auch die Gültigkeitsprüfung der Eingaben des Kunden. Auch die Dynamik wird durch Funktionen repräsentiert.

4.1 Content

Schemaebene. Es entspricht dem aktuellen Stand der Technik, die Schemaebene, also das eigentliche Content-Modell, mit Hilfe eines konzeptuellen Modells zu erfassen. In einem solchen Modell wird die Anwendungsdomäne durch Konzepte (Klassen) und Beziehungen zwischen ihnen repräsentiert. Als Notation haben sich UML Klassendiagramme durchgesetzt. Das Content-Modell definiert präzise, welche Begriffe für die Anwendungsdomäne relevant sind und wie sie zusammen hängen. Durch das Modell wird die Struktur des Contents definiert, das (Typ-)Schema.

Instanzebene. Bei der Modellierung einer Website wird in aller Regel vom eigentlichen, konkreten Inhalt abstrahiert. Die wesentlichen Aspekte werden vom Content-Modell auf Schemaebene berücksichtigt. Der Content selbst ist eine *Instanz* des Content-Modells. Unser Ansatz stützt sich hierfür auf *Graphentechnologie*. Der Content ist in einem typisierten Graphen gespeichert, dessen Typschema durch das Content-Modell vorgegeben ist. Graphen sind eine mächtige mathematische Struktur, und zeigen ihre Stärke hier insbesondere beim Abfragen (Querying) der Inhalte, was benötigt wird, um zu definieren, welche Inhalte auf den einzelnen Seiten zu sehen sein sollen. Das Querying wird durch die Query-Funktionen erledigt, die in Abschnitt 4.4 beschrieben werden.

4.2 Navigationsstruktur

Abbildung 1 zeigt die Navigationsstruktur für die Beispielanwendung. Es ist eine visuelle Repräsentation der Hyperlinkstruktur der Website.

Jedes Rechteck steht für eine Einzelseite oder für eine Klasse von gleichartigen Seiten. Die durchgezogene Pfeile definieren eine Hierarchie unter den Seiten, die so genannte *primäre Navigationsstruktur*. Durch die Hierarchie gibt es einen eindeutigen Pfad von der Wurzel zu jeder Seite. Auch lässt sich anhand der Hierarchie automatisch ein “Inhaltsverzeichnis” (site map) der Website erzeugen. Gestrichelte Pfeile stehen für Hyperlinks, die unabhängig von der Hierarchie zwischen den Seiten existieren.

Wir unterscheiden vier Arten von Seiten, die durch unterschiedliche Symbole kenntlich gemacht werden. Rechtecke mit einem Blitzsymbol stehen für *dynamische Seiten*, die ihren Inhalt dynamisch verändern, also bei jeden Zugriff einen anderen Inhalt liefern. Rein *statische Seiten* haben ein undekoriertes Rechteck. Enthält eine Webseite ein *Formular*, so markieren wir dies durch ein stilisiertes Formular im Rechteck. Schließlich gibt es noch das Symbol, das einen Stapel von Seiten darstellt. Dies repräsentiert *virtuelle Seiten*, die jeweils für eine ganze Klasse von Seiten stehen. In der Abbildung 1 ist dies beispielsweise CustomerHome(id). Diese steht für personalisierte Homepages für jeden Kunden. Für jeden gültigen Wert von id gibt es eine eigene Seiten-*Instanz*.

Das Diagramm erlaubt die Notation von weiteren Angaben zu den Hyperlinks, insbesondere durch die Angabe von Berechtigungen zum Zugriff auf Seiten oder Bereiche.

4.3 Seitenfunktionen

Der zentrale Baustein einer Website sind die einzelnen Seiten. In unserem Ansatz werden die Seiten durch Funktionen beschrieben. Ein kurzes Seitenfragment sahen Sie bereits in Abschnitt 2. Seitenfunktionen liefern eine abstrakte Repräsentation der endgültigen Seite, die anschließend durch eine Präsentationsfunktion (siehe Abschnitt 4.5) in die endgültige Ausgabesprache überführt wird. Wir nennen die abstrakte Repräsentation der Seiten APD (abstract page description) und nutzen den gleichnamigen Datentyp zur internen Darstellung. Eine Seitenfunktion liefert also einen Term vom Typ APD.

Das Zusammenfügen einer Seite aus einzelnen Bausteinen kann mit der funktionalen Programmiersprache auf sehr präzise und einfache Weise erfolgen. Gleichzeitig kann jederzeit die volle Mächtigkeit der Sprache ausgenutzt werden, um beispielsweise Berechnungen durchzuführen. So gesehen handelt es sich um eine “eingebaute Skriptsprache”. Viele Systeme bieten solche Skriptsprachen an, um dynamische Seiten zu beschreiben. Wir jedoch schlagen den durchgängigen Gebrauch einer funktionalen Sprache vor, die damit gleichzeitig Spezifikations- und Skriptsprache ist. Dies erlaubt eine größere Konsistenz der Spezifikation.

4.4 Query- und Updatefunktionen

Die Instanzdaten werden in einer Graphstruktur gespeichert. Ein Knoten des Graphen repräsentiert jeweils ein Inhaltsobjekt, die Kanten repräsentieren die Beziehungsinstanzen. Der Graph ist eine Instanz des Content-Schemas.

Das Wiederfinden von Informationen in Graphen lässt sich mit *Querying* komfortabel realisieren. Graphanfragesprachen sind in der Regel sehr ausdrucksmächtig. Uns genügt eine kleine Menge von Funktionen, um die notwendigen Queries zu formulieren.

Änderungen am Datenbestand, die erforderlichen Updates, geschehen analog mit Funktionen, die den Graphen verändern.

4.5 Präsentationsfunktionen

Die Transformation der abstrakten Seitenbeschreibungen in das gewünschte Ausgabeformat wird von Präsentationsfunktionen übernommen. Diese bestimmen damit das “Aussehen” der Website. In der Regel ist die Zielsprache XHTML, was durch den Ansatz jedoch in keiner Weise vorgeschrieben wird. Abbildung 2 zeigt das Ergebnis einer sehr einfachen Umwandlung der APD für ein Eingabeformular in XHTML.

Die Präsentationsfunktionen sind jederzeit austauschbar, und die Ausgabe kann sogar abhängig vom aktuellen Systemzustand variiert werden. Auf diese Weise können beliebig viele Ausgabemedien berücksichtigt werden, und auch die benutzerspezifische Adaption der Seiten (Personalisierung) kann hiermit realisiert werden.

4.6 Dynamik

Die Geschäftslogik oder *Dynamik* hinter einer Website wird in den Anforderungsdokumenten erfasst. Es werden zum Beispiel Anwendungsfälle oder Szenarios beschrieben, die das erwartete Verhalten der Website festlegen.

Anhand der Terminologie, die durch das Content-Modell festgelegt ist, können viele Aussagen über das Verhalten formalisiert werden. Wir schlagen vor, dies mit funktionalen Spezifikationen zu tun.

Als einfaches Beispiel diene die Anforderung im Rahmen des TAS, dass die Eingaben des Benutzers zur Suche nach einer Reise zwei Plausibilitätsregeln genügen müssen: (a) Start- und Zielflughafen dürfen nicht identisch sein; und (b) Das Abreisedatum liegt später als das Rückreisedatum. Die folgende Funktion beschreibt genau diese Kriterien:

```
checkWellformedness :: TripDescriptionRecord → Bool
checkWellformedness td =
  (originCity td ≠ destinationCity td)                                    (1)
  && (
    if (isJust (dateOfReturn td)) −− Wurde das Rückreisedatum angegeben?
    then (fromJust (dateOfReturn td) > (dateOfDeparture td))              (2)
    else True)
```

Regel (a) wird in Zeile (1) geprüft, und Zeile (2) lässt die Funktion genau dann True zurückliefern, wenn Regel (b) ebenfalls erfüllt ist.

5 Stand der Forschung

Modell-basierte Ansätze für die Spezifikation von Websites haben eine lange Tradition. Sie können grob anhand ihrer Wurzeln klassifiziert werden: Einige verwenden objektorientierte Modelle, andere Entity-Relationship-Modelle, andere sind eher Dokumentenzentriert. Die einflussreichsten "Schulen" sind der graphbasierte Strudel-Ansatz [FFLS00], das TSIMMIS-Projekt [CGMH+94], die ER-basierte RMM [ISB95], Araneus [MMA99], OOHDM [SR95], WebML [Cer00], und UWE. Die Integration von Modellen in eine Programmiersprache anhand einer Domänen-spezifischen Sprache (domain specific language) wird in [NS06] beschrieben.

Keiner der uns bekannten Ansätze beruht in dem hier vorgeschlagenen Maße auf funktionalen Spezifikationen. Die Erzeugung von HTML und XML mit funktionalen Programmiersprachen kann mit komfortablen Bibliotheken erfolgen (z.B. [Thi02]), die zusätzlich die syntaktische Korrektheit der erzeugten Dokumente garantieren. Durch das mächtige Typsystem von Haskell gelingt es sogar, die (Pseudo-)Validität eines HTML-Dokuments zur Compilezeit überprüfen zu lassen. Es werden auch vollständige Webserver in funktionalen Sprachen implementiert (insbesondere [Thi07], [Han06]). Diese Systeme erlauben es, Webanwendungen zu programmieren, ohne sich um komplexe Fragen des konsistenten Session-Handlings kümmern zu müssen (z.B. was geschieht, wenn der Benutzer die

"Zurück"-Taste des Browsers benutzt und ein Formular ein zweites Mal absendet). Die genannten Ansätze berücksichtigen aber nicht den Modellierungsaspekt, sondern fokussieren auf die Implementierungsebene.

6 Zusammenfasssung und Ausblick

Der hier vorgestellte Ansatz verwendet funktionale Spezifikationen zur Modellierung von Websites. Wir folgen einer strikten Trennung der Belange und identifizieren sechs Kernbereiche, die getrennt voneinander betrachtet und modelliert werden können. Die Bereiche sind der Content, die Navigation, die Seiten, die Queries und Updates, die Präsentation und die Dynamik. Das Content-Modell und die Navigationsstruktur sind durch 'herkömmliche' Modelle erfasst, nämlich einem objekt-orientierten, konzeptuellen Modell respektive einer visuellen Sprache. Die anderen vier Aspekte werden formal, mit Hilfe einer funktionalen Sprache, notiert. Wir verwenden die funktionale Programmiersprache Haskell.

Durch die funktionale Sprache werden die Spezifikationen konzis, leichter wartbar und wiederverwendbar. Jederzeit können neue Abstraktionsebenen definiert werden, was die Modellierung erheblich vereinfacht. Der wichtigste Punkt ist jedoch, dass die Spezifikationen direkt ausführbar sind.

Unser Ansatz soll weiterentwickelt werden. Die Speicherung des Content im Graphen erfolgt derzeit noch ohne die Berücksichtigung von Typinformationen. Die durchgängigere Verwendung von Typinformationen würde die Spezifikation noch weiter verbessern. Die Integration unseres Ansatzes in einen funktionalen Webserver, wie bereits in Abschnitt 5 erwähnt, würde neue Möglichkeiten der dynamisierten Ausgabesteuerung eröffnen.

Nicht zuletzt sollte der Ansatz um eine komfortablere Benutzungsschnittstelle erweitert werden. Das Verfassen der funktionalen Spezifikation in einem Texteditor entspricht nicht dem, was einem nicht versierten Benutzer zugemutet werden kann. Hier könnte eine visuelle Sprache, mit entsprechender Werkzeugunterstützung, einen großen Fortschritt bringen. Das Werkzeug würde dann beispielsweise die Seitenfunktionen als Ausgabe liefern, und sie könnten direkt ausgeführt werden. Hier möchten wir bestehende Modellierungsansätze auf geeignete Modellierungssprachen untersuchen, um eine Integration dieser Ansätze mit dem unseren zu vollziehen.

Literatur

[Cer00] Stefano Ceri. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):137–157, 2000.

[CGMH+94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman und Jennifer Widom. The TSIMMIS Project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, Seiten 7–18, Tokyo, Japan, 1994.

[FFLS00] Mary Fernández, Daniela Florescu, Alon Y. Levy und Dan Suciu. Declarative Specification of Web Sites with Strudel. *VLDB Journal*, 9(1):38–55, 2000.

[GE07] Torsten Gipp und Jürgen Ebert. Functional Web Applications. In Piero Fraternali, Luciano Baresi und Geert-Jan Houben, Hrsg., *7th International Conference on Web Engineering (ICWE 2007)*, LNCS (to appear). Springer-Verlag, 2007.

[Gip06] Torsten Gipp. *Functional Web Site Specification.* Logos Verlag Berlin, Berlin, 2006.

[Han06] Michael Hanus. Type-Oriented Construction of Web User Interfaces. In *Proc. of the 8th International ACM SIGPLAN Conference on Principle and Practice of Declarative Programming (PPDP'06)*, Seiten 27–38. ACM Press, 2006.

[ISB95] Tomás Isakowitz, Edward A. Stohr und P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM*, 38(8):34–44, 1995.

[KPRR04] Gerti Kappel, Birgit Pröll, Siegfried Reich und Werner Retschitzegger, Hrsg. *Web Engineering: Systematische Entwicklung von Web-Anwendungen*. dpunkt.verlag, Heidelberg, 2004.

[MMA99] Giansalvatore Mecca, Paolo Merialdo und Paolo Atzeni. Araneus in the Era of XML. *IEEE Data Engineering Bulletin*, 22(3):19–26, September 1999.

[NS06] Demetrius A. Nunes und Daniel Schwabe. Rapid prototyping of web applications combining domain specific languages and model driven design. In *ICWE '06: Proceedings of the 6th international conference on Web engineering*, Seiten 153–160, New York, NY, USA, 2006. ACM Press.

[SR95] Daniel Schwabe und Gustavo Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 38(8):45–46, 1995.

[Thi02] Peter Thiemann. A Typed Representation for HTML and XML Documents in Haskell. *Journal of Functional Programming*, 12(4 and 5):435–468, July 2002.

[Thi05] Peter Thiemann. An Embedded Domain-Specific Language for Type-Safe Server-Side Web-Scripting. *ACM Transactions on Internet Technology*, 5(1):1–46, 2005.

[Thi07] Peter Thiemann. Web Authoring System Haskell (WASH). http://www.informatik.uni-freiburg.de/~thiemann/haskell/WASH/, February 2007.

Dr. Torsten Gipp wurde am 27.12.1975 in Boppard am Rhein geboren. Nach der Erlangung der Allgemeinen Hochschulreife in 1995 begann er mit dem Studium der Informatik an der Universität Koblenz-Landau, das er im Jahre 2000 mit dem Diplom erfolgreich beendete. Im Anschluss arbeitete er als wissenschaftlicher Mitarbeiter am Institut für Softwaretechnik der Universität Koblenz-Landau und promovierte im Jahr 2006 zum Dr. rer. nat.

In den Jahren zwischen 1995 bis 2002 arbeitete Herr Gipp als Softwareentwickler in einem Softwarehaus. Seit 2005 ist er an der Universität Koblenz-Landau mit der Universitäts-weiten Einführung eines Content-Management-Systems beauftragt.

In seiner Freizeit fährt Herr Gipp mit dem Mountain-Bike, joggt, er kocht gerne und tanzt Salsa. Seit etwa einem Jahr beschäftigt er sich zudem mit der spanischen Sprache.