Einsatz Domänen-spezifischer Sprachen für Komponenten-basierte Web Anwendungen

Martin Nussbaumer
University of Karlsruhe
Engesserstr. 4
76128 Karlsruhe,
Germany
+49 (721) 608-8073
nussbaumer@tm.uka.de

Patrick Freudenstein
University of Karlsruhe
Engesserstr. 4
76128 Karlsruhe,
Germany
+49 (721) 608-8042
freudenstein@tm.uka.de

Martin Gaedke
University of Karlsruhe
Engesserstr. 4
76128 Karlsruhe,
Germany
+49 (721) 608-8076
gaedke@tm.uka.de

(Übersetzung von "Web Application Development Employing Domain-Specific Languages", IASTED Conference on Software Engineering, 2006, pages: 13-18)

In Projekten zur Entwicklung verteilter Web-basierter Systeme stellt die Spezifikation der zu entwickelnden Lösung eine zeitintensive Aufgabe dar, bei der häufig Kommunikationsschwierigkeiten zwischen Kunden, Anwendern und Entwicklern auftreten. Domänen-spezifische Sprachen (DSLs) bieten sich hier aufgrund ihrer Nähe zur Problemdomäne und der damit einhergehenden leichteren Erlernbarkeit und Verständlichkeit als ideale Alternative zu schwergewichtigen Modellierungsmethoden an. Dieser Beitrag präsentiert einen Ansatz zur Entwicklung von Web Anwendungen durch Komposition hochgradig konfigurierbarer Fachkomponenten und deren Konfiguration mittels DSL-Programmen.

1 Einleitung

Klar und zutreffend spezifizierte Anforderungen sowie die kontinuierliche und intensive Zusammenarbeit mit allen betroffenen Stakeholdern gehören zu den fünf bedeutendsten Erfolgsfaktoren in Softwareentwicklungs-Projekten (vgl. The Standish Group International 1994-2005). Eine funktionierende und effektive Kommunikation stellt dabei die Grundlage zur Umsetzung dieser Faktoren dar. Besonders im Bereich der Anforderungsanalyse und des konzeptionellen Entwurfs ist das Vermeiden von Missverständnissen zwischen allen Projektbeteiligten unabdingbar.

In den letzten Jahren hat sich im Bereich des Web Engineerings eine Vielfalt an Methodiken herausgebildet, die eine möglichst umfassende, systematische und

formale Spezifikation der verschiedenen Aspekte einer verteilten Web-basierten Lösung anstreben (vgl. Schwabe et al. 1996; Ceri et al. 2000, S. 137-157). Diese Ansätze versuchen mit ihren Modellen meist die Eigenschaften und Problemstellungen ihrer Domäne so umfassend wie möglich abzudecken und bieten hierfür eine sehr umfangreiche Menge an Konzepten und Notationen. Aufgrund der damit verbundenen Mächtigkeit und Ausdruckskraft der Modelle eignen sich diese sehr gut zur Spezifikation und als Mittel zur Kommunikation innerhalb eines Entwicklungsteams.

Diese „schwergewichtigen“ Modellierungsansätze sind jedoch auch mit einem hohen Lernaufwand verbunden. Insbesondere für Personen, die über keine Erfahrungen im Bereich der Softwareentwicklung oder über verschiedenste akademische und nicht-akademische Hintergründe verfügen, ist der anfänglich notwendige Lernaufwand unverhältnismäßig hoch. Für die gemeinsame Anwendung mit Kunden und Endnutzern sind derartige Ansätze und Modelle daher nur wenig geeignet.

Eine Alternative dazu stellen Domänen-spezifische Sprachen (“Domain-specific Languages (DSLs)”), auch bekannt unter dem Begriff „Little Languages“ (vgl. Bentley 1986, S. 711-721), dar. Deursen, Klint und Visser definieren sie als Programmiersprachen oder ausführbare Spezifikationssprachen, die ihre Ausdrucksstärke durch die Verwendung geeigneter Notationen und Abstraktionen unter Fokussierung und Beschränkung auf eine klar definierte Problemdomäne erlangen (vgl. Deursen et al. 2000, S. 26-36). Aufgrund ihrer starken Fokussierung auf einen kleinen Aspekt der zu entwickelnden Lösung und durch den Einsatz dedizierter Konzepte und grafischer Notationen aus der Problemdomäne sind DSLs einfach zu erlernen, zu verstehen und einzusetzen – sowohl für die Entwickler als auch für Kunden und Endanwender. Durch die Nutzergruppen-spezifische Integration dedizierter grafischer Notationen und begleitender Editoren kann die Verständlichkeit und „Benutzerfreundlichkeit“ einer DSL noch weiter verbessert werden. DSL-Programme können ähnlich wie bei Java oder C# mittels eines speziellen DSL-Compilers in ausführbaren Code transformiert werden.

Unser Ziel ist es, Stakeholder und Domänenexperten durch den Einsatz von DSLs in die Lage zu versetzen, Aspekte eines verteilten, Web-basierten Systems eigenständig zu verstehen, zu validieren, zu ändern und sogar selbst zu entwickeln. Regelmäßig auftretenden Missverständnissen und Schwierigkeiten bei der Kommunikation kann durch eine dadurch möglich werdende, intensive Integration von Stakeholdern in den Entwicklungsprozess begegnet werden.

Die Struktur dieses Beitrags ist wie folgt gegliedert: In Kapitel 2 motivieren wir unsere Idee des DSL-basierten Web Engineerings anhand unserer Erfahrungen im Rahmen eines großen universitätsweiten Enterprise Application Integration (EAI) Projekts. Anschließend präsentieren wir in Kapitel 3 unseren Ansatz, bestehend aus einem evolutionären DSL Framework und einer darunter liegenden

technischen Plattform. In Kapitel 4 skizzieren wir einen Ausschnitt unseres DSL Katalogs und beschreiben ausführlich eine DSL zur Spezifikation dynamischer, formularbasierter Benutzerinteraktion. Schließlich zeigen wir in Kapitel 5 die Anwendung unseres Ansatzes anhand eines Szenarios aus der Praxis und fassen in Kapitel 6 unsere Arbeit zusammen und geben einen Ausblick über die zukünftig geplanten Aktivitäten.

2 Erfahrungen aus der Praxis

Im Folgenden berichten wir von unseren Praxiserfahrungen aus mehreren Projekten zur Entwicklung komplexer Web Anwendungen und insbesondere aus einem großen universitätsweiten Enterprise Application Integration (EAI) Projekt namens “Karlsruher Integriertes Informationsmanagement (KIM)” (vgl. Juling 2005). Dabei fokussieren wir besonders auf die Aspekte im Bereich der Kommunikation zwischen den verschiedenen Projektbeteiligten in den Phasen der Anforderungsspezifikation und des konzeptionellen Entwurfs.

Die große Vielfalt an Problemstellungen, insbesondere hinsichtlich sich stetig und rasch verändernder Marktsituationen, und verschiedenartigster Stakeholder, wie man sie besonders in großen Projekten zur Entwicklung verteilter, Web-basierter Systeme vorfindet, führt häufig zu Kommunikationsschwierigkeiten. In derart verteilten Szenarien, wie es z.B. bei EAI-Projekten der Fall ist, verfügen die Beteiligten über unterschiedlichste Hintergründe im Hinblick auf ihre Tätigkeitsschwerpunkte und ihre (akademische oder nicht-akademische) Ausbildung. Dies führt dazu, dass bei Gesprächen über Aspekte des zu lösenden Problems bzw. zu entwickelnden Systems jede Gruppe ihre eigene „Sprache“ spricht. So reichte beispielsweise bei der Spezifikation der Geschäftsprozesse im KIM Projekt die Bandbreite der gewünschten Sprachen von der geschriebenen natürlichen Sprache über einfache Flussdiagramme bis hin zu Petri Netzen (vgl. Petri 1962) und der Business Process Modeling Notation (BPMN).

Darüber hinaus stellten wir fest, dass die große Mehrheit der Beteiligten über keinerlei Erfahrung im Bereich der Software-Entwicklung bzw. Modellierung verfügte und aufgrund ihres Tagesgeschäfts nur schwer für Interviews zur Verfügung stand. Um eine möglichst weit reichende Zusammenarbeit mit allen Projektbeteiligten zu ermöglichen, war Einfachheit im Erlernen, Verstehen und Anwenden daher eine der Hauptanforderungen an die zur Modellierung zu verwendenden Sprachen.

In den letzten Jahren wurden viele Modellierungssprachen und -methodiken entwickelt mit dem Ziel, ihre jeweils zugehörige Problemdomäne so umfassend wie möglich abzudecken. Dies wurde meist durch die Integration einer Vielzahl von Konzepten und Notationen für möglichst alle Eventualitäten der Problemdomäne erreicht. Die dadurch erzielte Ausdruckskraft und Mächtigkeit dieser Ansätze ist beträchtlich, was sie zu einem guten Werkzeug und Kommunikationsmedium für den konzeptionellen und logischen Entwurf innerhalb des Entwicklungsteams macht. Eine überraschende Beobachtung war jedoch, dass diese Sprachen für die gemeinsame Anwendung mit Stakeholdern zu komplex waren. Der Aufwand und die Zeit, die mit dem Erlernen einer Projekt-weit vorgegebenen Menge an Modellierungssprachen für alle Beteiligten verbunden gewesen wäre, stellten sich als zu hoch heraus.

Demzufolge erkannten wir das Grundprinzip der Einfachheit als Schlüsselfaktor zur Erreichung von Anwendbarkeit und Effektivität. Wir beobachteten, dass der Einsatz kleiner und einfacher Sprachen, die ungefähr nur die wichtigsten 80% der Konzepte und Problemstellungen einer Domäne abdecken, im Gegenzug aber so einfach blieben, dass sie von 100% der Beteiligten verstanden und genutzt werden konnten, in den meisten Fällen die bessere Wahl darstellte. Diese 80%-ige Erfassung der Aspekte und Problemstellungen einer Domäne reichte meist völlig aus und führten zu einer deutlich höheren Anwendbarkeit und Akzeptanz als es mit existierenden „schwergewichtigen“ Ansätzen der Fall war.

Eine weitere Verbesserung der Erlern- und Anwendbarkeit dieser Sprachen konnte durch die Integration aus der Problemdomäne bekannter Abstraktionen und Konzepte sowie Stakeholder-Gruppen-spezifischer grafischer Notationen erlangt werden. Dadurch konnten Wiedererkennungseffekte bei den Beteiligten erzielt werden, so dass die Sprachen an Einfachheit und Intuitivität gewannen.

3 Das WebComposition DSL Framework

Ausgehend von unseren im vorangegangenen Kapitel beschriebenen Erfahrungen stellen Domänen-spezifische Sprachen (DSLs) eine ideale Alternative zu den existierenden, „schwergewichtigen“ Modellierungssprachen und -methodiken dar. Wir verwenden DSLs als kleine, einfache und hochgradig auf einen bestimmten Aspekt eines verteilten Web-basierten Systems zugeschnittene Spezifikations-Sprachen. Die Nutzung von Abstraktionen, Konzepten und grafischer Notationen aus der zugehörigen Problemdomäne zählen zu den weiteren Charakteristika von DSLs. Dadurch sind sie einfach zu erlernen, zu verstehen und zu verwenden, besonders für Domänen-Experten ohne Kenntnisse im Bereich der Software-Entwicklung.

Abb. 1 zeigt unser evolutionäres, auf dem WebComposition Ansatz (vgl. Gaedke und Turowski 2000, S. 117-138) basierendes Prozessmodell für die DSL-basierte Entwicklung von Web Anwendungen. Wir unterscheiden drei Phasen im Zuge einer kontinuierlichen Evolution: *Analyse*, *Entwicklung für Wiederverwendung* und *Entwicklung mit Wiederverwendung*.

In der *Analyse*-Phase wird im Bestand der verfügbaren DSLs nach einer für den gegebenen (Problem-)Kontext geeigneten DSL recherchiert. Dabei können verschiedene Kontext-Parameter wie beispielsweise der zu spezifizierende Aspekt der Web Anwendung, die Charakteristika der mitarbeitenden Stakeholder-Gruppe oder die Art des zu konzipierenden Systems in die Suche mit einfließen. Bleibt die Suche erfolglos, d.h. wurde keine passende DSL gefunden, muss eine bestehende angepasst oder gar eine neue entwickelt werden. Ist dies der Fall, wird in enger Zusammenarbeit mit den am Projekt beteiligten Stakeholdern die Problemdomäne analysiert und geeignete Abstraktionen und Konzepte zur Beschreibung von Lösungen innerhalb der Domäne identifiziert. Das Ergebnis dieser Phase bildet das (gefundene oder angepasste bzw. neu entwickelte) *Domain-Specific Model (DSM)*. Dabei handelt es sich in der Regel um ein XML Schema Dokument, das das formale Schema für alle Lösungen, die mit der DSL innerhalb der Problemdomäne spezifiziert werden können, definiert.

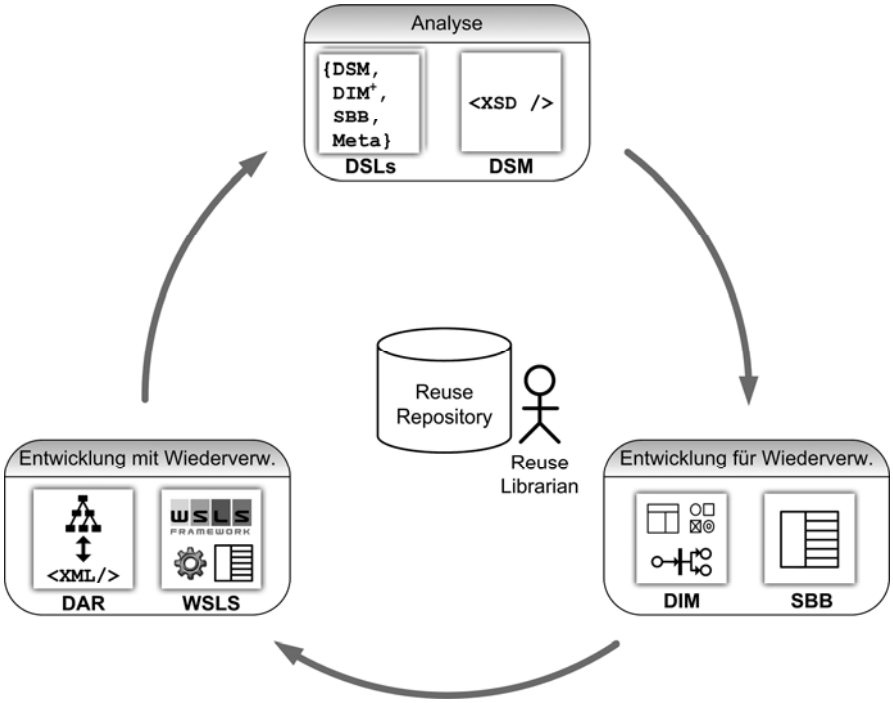

Abb. 1: Das evolutionäre WebComposition Prozessmodell für die DSL-basierte Entwicklung von Web Anwendungen

Die Phase „*Entwicklung für Wiederverwendung*" setzt auf der Analyse-Phase auf und wird im Falle von Änderungen bzw. Neuentwicklung einer DSL durchlaufen. Wurde hingegen in der vorhergehenden Phase eine geeignete existierende DSL gefunden, kann die Phase übersprungen werden.

Basierend auf dem DSM wird hier zunächst ein oder mehrere *Domain Interaction Model (DIM)* entwickelt. Ein DIM definiert zu den im DSM spezifizierten Abstraktionen und Konzepten eine dedizierte Notation, die für eine gegebene Stakeholder-Gruppe so intuitiv wie möglich sein sollte. Dabei muss das DIM nicht notwendigerweise alle Aspekte des DSM abdecken. In manchen Fällen ist für eine bestimmte Zielgruppe nur ein bestimmter Ausschnitt des DSM relevant, so dass ein DIM für diese Zielgruppe auch nur für diesen Ausschnitt Notationen bereitstellen muss. Mit Hilfe dieser Notationen können Stakeholder die DSL anwenden, d.h. auf dem DSM basierende DSL-Programme verstehen, validieren und sogar selbst entwickeln, ohne dabei mit kompliziertem Quellcode konfrontiert zu werden. Die Konzepte und Notationen eines DIM sollten deshalb in Zusammenarbeit mit den Stakeholdern aus der Problemdomäne abgeleitet werden, so dass sie für diese möglichst einfach zu erlernen und anzuwenden sind. Sofern möglich sollte dabei versucht werden, die Notationen so einfach wie möglich zu

gestalten, z.B. dass darauf basierende Modelle einfach mit Papier und Bleistift gezeichnet werden können. Dadurch kann im Vergleich zu Modellen, die aufgrund ihrer Komplexität eher für den Entwurf am Computer geeignet sind, eine höhere Akzeptanz und schnellere Erlernbarkeit realisiert werden (vgl. Nielsen 1993). Um den verschiedenen Anforderungen und Charakteristika verschiedener Stakeholder-Gruppen entsprechen zu können, kann innerhalb einer DSL für jede Gruppe ein eigenes DIM bereitgestellt werden. Durch begleitende, auf dem DIM basierende (grafische) Editoren kann deren Anwendbarkeit und Effektivität noch weiter erhöht werden.

Neben der Entwicklung eines oder mehrerer DIMs wird in der Phase „Entwicklung für Wiederverwendung“ ein sog. *Solution Building Block (SBB)* für die DSL entwickelt. Ein SBB ist eine Fachkomponente, die für die Ausführung der mit einer DSL erstellten Programme zuständig ist. Dabei fungiert der SBB nicht als Code-Generator, sondern als Komponente, deren Verhalten mittels eines DSL-Programms, in der Regel ein auf dem DSM basierendes XML Dokument, konfiguriert werden kann. Das *WebComposition Service Linking System (WSLS)* (vgl. Gaedke et al. 2005, S. 26-37) fungiert dabei als technische Plattform für die SBBs. WSLS setzt auf den Grundprinzipien „Evolution und Wiederverwendung“ des WebComposition Ansatzes auf und ermöglicht die systematische Evolution von Web Anwendungen durch Wiederverwendung hochgradig konfigurierbarer Software-Komponenten. Die Realisierung des Prinzips „Konfigurieren statt Programmieren“ stellt dabei eines der Hauptziele von WSLS dar. Im Hinblick auf die SBBs erleichtert das WSLS Framework deren systematische Komposition und Konfiguration zu vollständigen Web Anwendungen.

In der Phase “*Entwicklung mit Wiederverwendung*” wird mit Hilfe der gefundenen oder angepassten bzw. neu entwickelten DSL Komponenten (dem Domain-Specific Model, den Domain Interaction Models und dem Solution Building Block) ein DSL-Programm zur Spezifikation eines Aspekts einer Web Anwendung entwickelt. Wir bezeichnen ein solches auf dem DSM basierendes und mit Hilfe der Notationen aus einem oder mehrerer DIMs entwickeltes DSL-Programm auch *Domain Abstract Representation (DAR)*. Ein DAR stellt die Spezifikation einer konkreten Lösung innerhalb der Problemdomäne der DSL dar. Da ein DSM in der Regel in Form eines XML Schema Dokuments vorliegt, wird dementsprechend ein DAR in einem darauf basierenden XML Dokument serialisiert und gespeichert.

Das entwickelte DAR wird anschließend in seiner XML Repräsentation an den Solution Building Block der DSL übergeben. Dieser adaptiert sein Verhalten entsprechend dem DSL-Programm und bringt es dadurch (indirekt) zur Ausführung. Die Entwicklung von Web Anwendungen kann somit evolutionär durch die Komposition von SBBs und deren Konfiguration mit DARs durchgeführt werden.

Sowohl die entstehende Vielzahl an DSLs für die unterschiedlichen Aspekte verteilter Web-basierter Systeme sowie die DSLs selbst unterliegen einer ständigen Evolution durch Variation und Selektion. So können die Erfahrungen aus dem Einsatz von DSLs mit unterschiedlichen Stakeholder-Gruppen zu Verbesserungen bzw. Änderungen, Neuentwicklungen oder gar dem Verwerfen von DSLs führen. Um den Einsatz und die Verwaltung der DSLs in einem solch evolutionären Umfeld möglichst systematisch und effizient zu gestalten, sieht unser Ansatz die Einrichtung eines zentralen *Reuse Repository* (vgl. Gaedke et al. 1999) sowie einer Team-Rolle namens „*Reuse Librarian*“ vor.

Das *Reuse Repository* dient als zentraler Ort zur Speicherung, Verwaltung und Suche von DSLs und zugehöriger Meta-Information. Während des gesamten Prozesses werden neu entwickelte oder angepasste Artefakte sowie Meta-Information wie zum Beispiel Erfahrungen im Umgang mit DSLs systematisch im Repository abgelegt. Darüber hinaus werden beispielsweise in der Analyse-Phase bestehende DSLs abhängig von durch den Anwendungs- und Projektkontext vorgegebenen Kriterien im Repository gesucht. Versionierungs-Funktionalitäten, ein geeignete Klassifikationsschema sowie Suchfunktionalitäten sind daher für das Reuse Repository unabdingbar.

Die Team-Rolle des *Reuse Librarian* ist für die systematische Verwaltung und den optimalen Einsatz der DSLs zuständig. Sie begleitet das Projekt-Team durch den gesamten Entwicklungsprozess und achtet bei der Entwicklung und der Auswahl von DSLs auf effiziente Wiederverwendung. Darüber hinaus steht sie bei der Suche und Anwendung von DSLs dem Team beratend zur Seite und ist für die Administration des Reuse Repository sowie die Konservierung und Bereitstellung gewonnenen Wissens verantwortlich.

4 DSL Katalog

In diesem Kapitel stellen wir drei DSLs zur Spezifikation wichtiger Aspekte bei der Entwicklung von Web Anwendungen vor: Navigationselemente, Web-basierte Navigationsprozesse und formularbasierte Interaktion. Die Beschreibung jeder DSL gliedert sich in eine kurze Einführung in die Problemdomäne, die Beschreibung des Domain-Specific Model und die Präsentation eines Domain Interaction Model. Während die ersten beiden DSLs nur kurz umrissen werden, wird die DSL zur Spezifikation formularbasierter Interaktion ausführlich beschrieben.

4.1 Link List

Problemdomäne: Die Spezifikation von Link-Strukturen wie zum Beispiel Menüs oder Landmarks durch die Verbindung von Anwendungsdomänen und deren untergeordneter Informationen stellt eine zentrale Aufgabe bei der Entwicklung von Web Anwendungen dar.

Domain-Specific Model: Das Link List DSM basiert auf XLink und bietet Konzepte zur Gliederung von Links und zur Festlegung des Verhaltens der Quelldomäne beim Übergang zum Ziel eines Links (Einbettung oder Austausch).

Domain Interaction Model: Bis jetzt haben wir ein DIM entwickelt, das Symbole für ein sog. „Level“, einen „Internal Link“, einen „External Link“ und einen „Connector“ definiert. Ein „Level“ gruppiert eine Menge von Links, die beiden Link-Typen spezifizieren das Verknüpfungsziel und das Übergangsverhalten und der „Connector“ realisiert die Aggregation von Links und die Verschachtelung der Levels.

4.2 Web-basierte Navigationsprozesse

Problemdomäne: In modernen Web Anwendungen spielen Navigationsprozesse, d.h. die dynamische Navigation durch eine Menge von Anwendungsdomänen abhängig von durch den Benutzer ausgelösten Ereignissen, eine wichtige Rolle.

Domain-Specific Model: Die aus dem Bereich der Endlichen Automaten bekannten Konzepte bilden die theoretische Grundlage unseres DSM. Die Realisierung des DSM basiert auf XLink und beinhaltet Elemente zur Spezifikation von Zuständen (gleichzusetzen mit Anwendungsdomänen), Übergängen und Ereignissen.

Domain Interaction Model: Aufgrund der Anforderungen und Charakteristika im KIM Projekt haben wir ein DIM entwickelt, das auf einfachen und intuitiven Petri Netz-Konstrukten basiert.

4.3 PetrIX Dialog Modellierung

Problemdomäne: Terry R. Schuster verkündetet auf der Max World Konferenz in San Francisco im Jahr 1998: „Interaction is not animation. It's not audio. It's not video. It's user control and dynamic experience.” Gerade im Umfeld des Webs, das überwiegend durch grafische Dialoge geprägt ist, muss sich der Entwurf von Benutzerschnittstellen mit dem bedeutungsvollen Verhalten einzelner grafischer Elemente beschäftigen. Daher wird im Folgenden eine DSL vorgestellt, die den

umfassenden Entwurf von Dialogschnittstellen ermöglicht und so einen wichtigen Beitrag zur Förderung dynamischer Benutzerinteraktion leistet.

Domain-Specific Model: Das DSM wird auf der Grundlage von XForms (vgl. Dubinko et al. 2003) realisiert, da es eine Empfehlung des W3C darstellt und ein integraler Bestandteil von XHTML 2.0 ist. Obwohl herkömmliche Browser in aller Regel XForms noch nicht unterstützen, haben die meisten Hersteller Unterstützung für die XForms Empfehlung signalisiert. Erste Ansätze existieren bereits wie beispielsweise für Mozilla (vgl. Beaufour et al. 2005) oder ein Plug-In für den Internet Explorer. Ein weiterer Vorteil von XForms bildet die Interoperabilität zu anderen Empfehlungen wie SMIL, SVG oder WML.

XForms unterscheidet drei wesentliche Teile, um die Darstellung von den Daten und dem Verhalten zu trennen: das *XForms model*, *instance data* und *user interface*. Das *XForms model* beschreibt lediglich die logischen Elemente eines Formulars. Es besteht aus einer x*ml instance*, die den Wertebereich eines Formulars beschreibt, das von einem Benutzer bearbeitet wird. Darüber hinaus verfügt das Model auch über die nötigen Informationen zum Übertragen dieser Daten, wie bspw. die Adresse des Servers oder die Kodierungsrichtlinien. Im Gegensatz dazu beschreibt das *user interface* die Anordnung und das Erscheinungsbild der einzelnen Dialogelemente, die wiederum an Teile der XML Instanz gebunden werden. Diese Datenbindung erfolgt über XPath Ausdrücke.

Das Verhalten dedizierter Interaktionen einzelner Dialogelemente wird in XForms mittels XML Events realisiert. XML Events bildet eine Abstraktion und damit Vereinheitlichung herkömmlicher Scripting Technologien wie JavaScript oder JScript, die sehr browserspezifisch sind (vgl. McCarron et al. 2003). Dadurch entstehen positive Effekte bezüglich der Entwicklungszeit, als auch eine Verbesserung hinsichtlich der Zugänglichkeit der entstehenden Dialoge.

Abb. 2 zeigt einen schematischen Überblick über die Dialogelemente der XForms Empfehlung. Zusätzlich zu den reinen Dialogelementen existieren auch Elemente zur Gruppierung („group“) und Manipulation des Verhaltens („repeat“, „switch/case“). Ein *Interaktionsträger* stellt das Bindeglied zwischen der Dateninstanz und dem Dialogelement dar. Er stellt damit quasi einen Repräsentanten für einen interaktiven Vorgang dar. Eine solche Bindung wird mittels XPath Ausdrücken direkt, oder über vordefinierte (wieder verwendbare) XForms-basierte Ausdrücke (*xforms:bind*), realisiert. Außerdem werden zusätzliche (Inter-)Aktionen mittels des XForms Event Modells im Interaktionsträger vereinigt und erlauben so eine Anpassung an ein jeweils gewünschtes Verhalten.

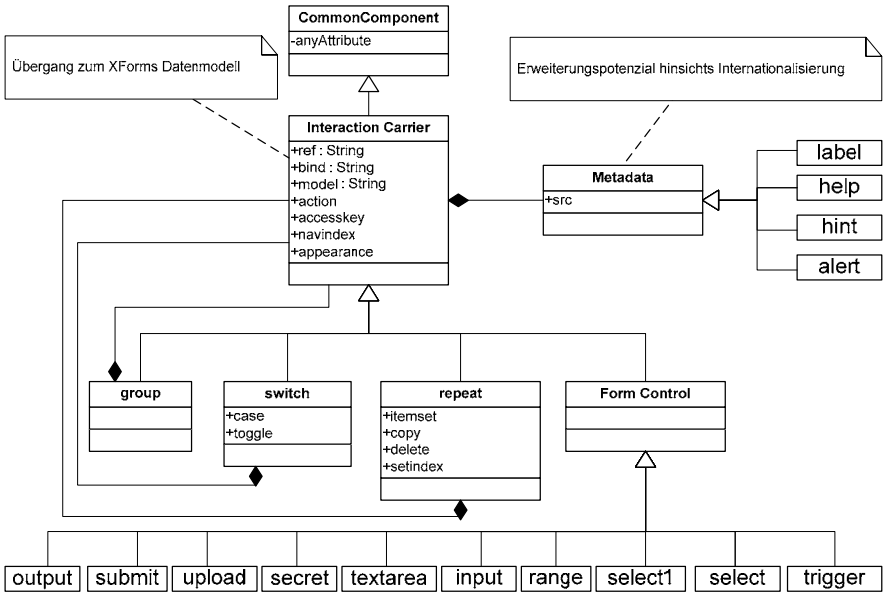

Abb. 2: Die Elemente der XForms Benutzerschnittstelle als Abstraktion mittels eines UML Klassendiagramms

Domain Interaction Model: Um die unterschiedlichen Stakeholder und Benutzerschnittstellendesigner adäquat zu unterstützen, wurde das Werkzeug Visio der Firma Microsoft angepasst. Diese Erweiterung bietet Unterstützung für die Modellierung und Visualisierung von PetrIX Diagrammen. PetrIX kombiniert die guten Modelleigenschaften von Petri Netzen (vgl. Petri 1962) für Interaktionsstrukturen wie zum Beispiel Workflows mit der Abstraktion und Erweiterbarkeit von XForms. Mittels geeigneter Transformationen auf Basis von Termersetzungen können die Petri Netze in XForms Dokumente überführt werden. Dazu unterscheidet PetrIX zwei logische Einheiten: Partitionen und Interaktionsstrukturen.

Eine *Paritionsstelle* fasst eine Menge von Interaktionsträgern (Stellen im Petri Netz) zusammen, gruppiert und reichert sie um Layoutinformationen an. Diese Layoutinformationen beinhalten dabei bspw. die Anwendung von Cascading Style Sheets (CSS) oder weiteren Ausprägungshinweisen (*appearance*) für die Präsentation auf einem konkreten Client. Partitionen können verschachtelt werden und sind somit in der Lage wiederum Partitionen zu enthalten.

Eine *Interaktionsstruktur* bildet einen Kontext innerhalb dessen Interaktion auf Basis von Interaktionsträgern gebildet wird. Sie definiert ferner das Verhalten zwischen den partizipierenden Stellen und kontrolliert, basierend auf der Markierung einer Stelle, die Sichtbarkeit und damit die dem Benutzer dargestellte Einheit des Dialogs. Dies gilt im Besonderen, wenn Anhängigkeiten zwischen

einzelnen Dialogeinheiten existieren oder die Komplexität des Gesamtdialogs durch einen geordneten Ablauf reduziert werden soll. Abb. 3 zeigt zwei für Web Anwendungen typische Interaktionsstrukturen: *Vor-Zurück* und *Auswahl*. Ihr Verhalten wird durch Petri Netz Transitionen ausgedrückt, die durch dedizierte Dialogelemente geschaltet werden können.

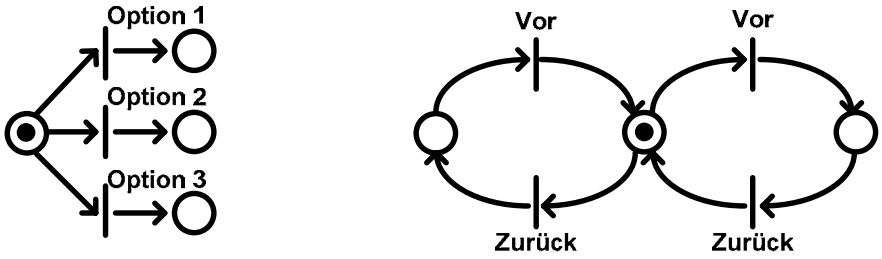

Abb. 3: Die Interaktionsstrukturen „Auswahl“ (links) und “Vor-Zurück” (rechts) sind gängige Muster für Dialoge in Web Anwendungen, hier dargestellt als Petri Netze.

Nachdem die Benutzerschnittstelle mit Hilfe von Visio modelliert wurde, werden die resultierenden Petri Netze mittels XSL(T) in XForms Dokumente transformiert. Diese Dokumente bilden dann die aus Kapitel 2 bekannte *Domain Abstract Representation (DAR)*, welche durch dedizierte Komponenten, den *Solution Building Blocks (SBB)*, konsumiert und weiterverarbeitet werden kann. Die Regeln für die Transformationen basieren auf der Zerlegung der Petri Netze durch ein Termersetzungssystem. In einer früheren Arbeit wird anhand einer Interaktionsstruktur exemplarisch aufgezeigt, wie eine solche Termersetzung ausgehend von einem Petri Netz durchgeführt wird (vgl. Gaedke und Nußbaumer 2002, S. 45-55). Durch die konsequente Übertragung unterschiedlicher Interaktionsstrukturen lassen sich so nach und nach weitere Regeln definieren, was die Wiederverwendung solcher Strukturen erhöht, die Verwendung von Mustern ermöglicht und letztendlich die leichtere Wiedererkennung von Strukturen fördert.

5 PetrIX - vom Modell zum Dialog

Im Folgenden stellen wir die PetrIX DSL im Rahmen eines konkreten Beispiels aus der Praxis vor. Ziel des Projektes war die Optimierung von Geschäftsprozessen durch ein Mitarbeiter-Selbstbedienungs-Portal. Dabei haben wir zunächst die interne Portal-Struktur mit Hilfe des WSLS Frameworks realisiert. Die hierbei entstandenen Anwendungsdomänen wurden mit dedizierten

Komponenten, den Solution Building Blocks assoziiert und mit den entsprechenden DSLs konfiguriert.

Abb. 4 stellt das Domain Interaction Model aus der PetrIX DSL dar, das in Abschnitt 4.3 eingeführt wurde. Dabei erleichtert der grafische Editor Visio durch die Kombination vordefinierter Interaktionsträger und -strukturen (vgl. Abb. 4: 1a, 1b) mit den visuell unterstützenden Funktionen, wie z.B. Drag & Drop, das Erstellen von Benutzerschnittstellen erheblich. Außerdem lassen sich die vordefinierten Dialogelemente mit zusätzlichen Attributen annotieren. Die entsprechenden Annotationen (vgl. Abb. 4: 3a, 3b) sind dem Domain-Specific Model XForms der PetrIX DSL nachempfunden und beschreiben das Verhalten und die Bindung an das Datenmodell. Visio sieht solche Erweiterungen als sog. *Stencils* vor. Für die PetrIX DSL wurde ein solcher Stencil entwickelt, um die gewünschte Unterstützung durch den grafischen Editor zu realisieren. Das dargestellte Diagramm (vgl. Abb. 4: 2) enthält die nötigen Partitionen, welche, durch Transitionen verbunden, die gewünschte Interaktionsstruktur realisieren. In der Abbildung wird gezeigt, wie die Interaktionsstruktur “Auswahl” mit dem Dialogelement `xforms:select1` verknüpft wird. Das jeweilige Auswahlkriterium findet sich an den Transitionen zu den verbundenen Partitionen. Das im Editor aktivierte Dialogelement `xforms:input` kann durch entsprechende Annotationen (vgl. Abb. 4: 3b) wie beispielsweise CSS oder Datenbindungen weiter angepasst werden.

Im Folgenden wird beschrieben, wie die mit PetrIX modellierten Benutzerschnittstellen auf XForms Code Fragmente abgebildet werden können. Dazu soll der in Abb. 4 dargestellte Auszug aus dem Szenario Reisekostenabrechung verwendet werden, in dem ein Mitarbeiter das Transportmittel auswählen soll; im vorliegenden Beispiel sind dies *car*, *train* oder *plane*. Abhängig von der Auswahl werden dem Mitarbeiter dann die relevanten Teile des Formulars (die Partitionen) dynamisch eingeblendet, um nur die dafür benötigten Informationen abzufragen.

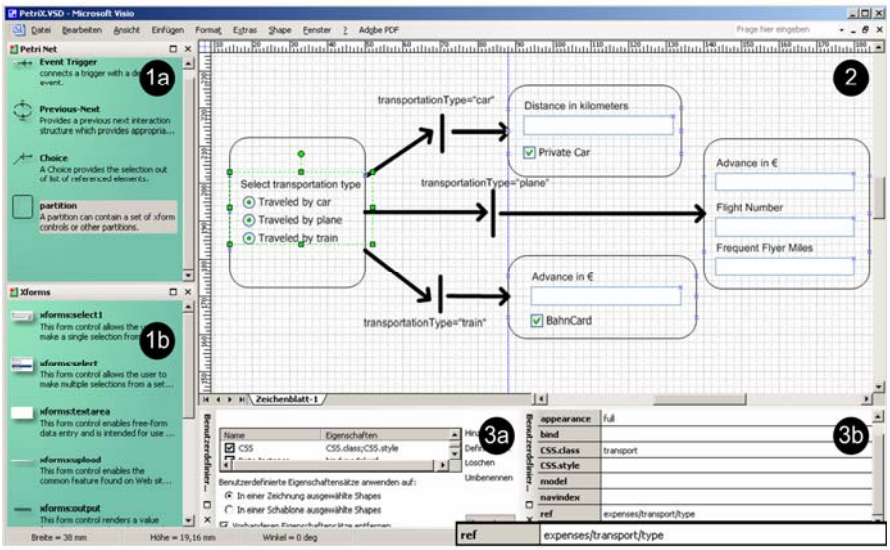

Abb. 4: Das Domain Interaction Model im Microsoft Visio Editor unterstützt Benutzerschnittstellen Designer bei der Erstellung von Dialogen basierend auf der PetrIX DSL.

Im Fall des `xforms:select1` Dialogelements wird der ausgewählte Wert mittels eines XPath Ausdrucks (expenses/transport/type) an die Dateninstanz des XForms Modells gebunden (1).

```
<xforms:model>                                                    (1)
 <xforms:instance>
  <expenses xmlns=”urn:forms:expenses”>
   ...
   <transportation>
     <type />
     <distance />
     <privateCar />
     <flightnumber />
     <frequentFlyerMiles/>
     <bahncard />
   </transportation>
   ...
  </expensesForm>
 </xforms:instance>
</xforms:model>
```

Nach seiner Fertigstellung wird das PetrIX Diagramm als XML Dokument aus Visio exportiert. Ein auf XSL(T)-basierendes Termersetzungssystem erzeugt danach das resultierende XForms Dokument. Im dargestellten Beispiel werden die

mit der Auswahl verbundenen Partitionen *car*, *plane* und *train* durch das XForms Modul *switch* wie folgt realisiert: Das `xforms:select1` Dialogelement wird an die zugehörige Dateninstanz gebunden und speichert den aktuell selektierten Wert, z.B. *car*. Das zugehörige XForms Code Fragment wird basierend auf den annotierten Werten, wie zum Beispiel die XPath Ausdrücke, durch entsprechende Termersetzungsregeln konstruiert. (2)

```
<select1 ref="expenses/transport/type">                (2)
<label>Select transportation type</label>
  <item>
    <label>Traveled by car</label>
    <value>car</value>
  </item>
  <item>
    <label>Traveled by train</label>
    <value>train</value>
  </item>
  <item>
    <label>Traveled by plane</label>
    <value>plane</value>
  </item>
</select1>
```

Die Interaktionsstruktur „Auswahl“ wird durch das *switch* Modul von XForms realisiert. Jede verbundene Transition wird durch ein `xforms:case` dargestellt. Basierend auf dem im switch Modul referenzierten Wert können die relevanten Teile der Benutzerschnittstelle zur Laufzeit dynamisch ein- und ausgeblendet werden. Dieser Wert wiederum spiegelt die aktuell gewählte Option aus dem Dialogelement `xforms:select1` wider.

```
<switch ref="expenses/transport/type">                 (3)
 <case id="car">
  Termersetzung der Partition "car"
 </case>
 <case id="plane">
  Termersetzung der Partition "plane"
 </case>
 <case id="train">
  Termersetzung der Partition "train"
 </case>
</switch>
```

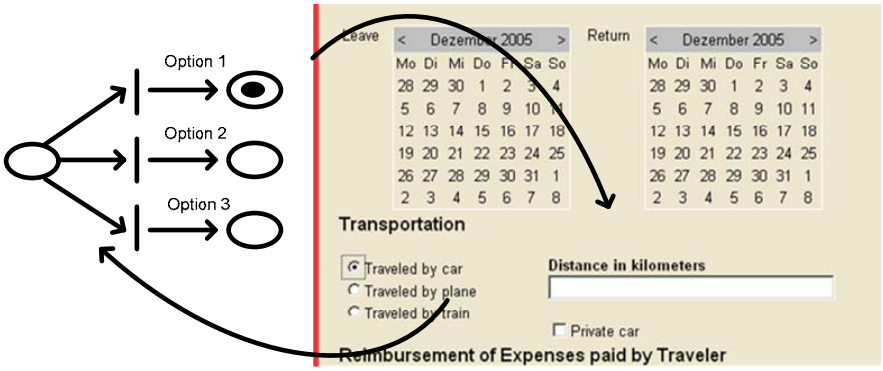

Abb. 5: Die Interaktionsstruktur “Auswahl” und der zugehörige Teil des Formulars.

Abb. 5 stellt schliesslich den zum Beispiel korrespondierenden Teil des Formulars innerhalb des Mitarbeiter-Selbstbedienungs-Portals dar. Die dargestellte Interaktionsstruktur befindet sich dabei gerade in dem Zustand, nachdem ein Benutzer die Option „Auto“ gewählt hat. Dementsprechend stellt das Formular auch nur die für diesen Zustand relevanten Teile dar.

6 Zusammenfassung & Ausblick

In diesem Beitrag stellten wir unseren Ansatz zur Entwicklung verteilter Web-basierter Systeme mit Domänen-spezifischen Sprachen vor. Wir berichteten von unseren Erfahrungen im Bereich der Kommunikation zwischen Kunden, Anwendern und Entwicklungsteams aus mehreren Praxis-Projekten. Darauf aufbauend arbeiteten wir die Ursachen heraus, die häufig zu Missverständnissen zwischen den Projektbeteiligten und dadurch zu ungenügend oder überhaupt nicht erfüllten Anforderungen führen.

Existierende Modellierungstechniken und -modelle beinhalten in der Regel eine äußerst umfangreiche Menge an Konzepten und Notationen, um dadurch die zugehörige Problemdomäne so erschöpfend wie möglich behandeln zu können. Im Hinblick auf die Zusammenarbeit mit Stakeholdern, d.h. Kunden und Anwendern, die meist über keinerlei Erfahrung im Bereich der Software-Entwicklung verfügen, erfordern diese jedoch einen zu hohen Lernaufwand.

Zur Reduktion der anfangs diskutierten Schwierigkeiten im Bereich der Kommunikation strebt unser Ansatz eine sehr intensive Einbindung der Stakeholder in den Entwicklungsprozess an. Dabei sollen diese eigenständig

Spezifikationen von Aspekten einer Web Anwendung verstehen, validieren, ändern und sogar entwickeln können. Um dies zu erreichen, stellt das Prinzip der Einfachheit eine Schlüsselrolle dar, so dass Domänen-spezifische Sprachen (DSLs) eine ideale Alternative zu existierenden schwergewichtigen Ansätzen darstellen. Wir verwenden DSLs als einfache, hochgradig fokussierte Sprachen zur Spezifikation kleiner und klar abgegrenzter Aspekte eines verteilten Web-basierten Systems. Dazu beinhalten sie Konzepte und Notationen, die in Zusammenarbeit mit Stakeholdern direkt aus der Problemdomäne abgeleitet werden.

Da die Menge an DSLs einer ständigen Evolution durch Variation und Selektion unterliegt, präsentierten wir einen evolutionären, wiederverwendungsorientierten Ansatz zur systematischen Entwicklung, Verwaltung und Anwendung von DSLs. Die technische Plattform unseres Ansatzes basiert auf sog. Solution Building Blocks, die mit Hilfe des WebComposition Service Linking Systems zu Web Anwendungen zusammengefügt und mittels DSL-Programmen konfiguriert werden können.

Anschließend stellten wir einen Auszug aus unserem DSL Katalog vor und beschrieben ausführlich eine DSL zur Spezifikation formularbasierter Benutzerinteraktion. Die DSL beinhaltet Konzepte aus XForms und Petri Netzen und verfügt über eine einfach zu verwendende grafische Notation, die durch einen in Microsoft Visio integrierten Editor unterstützt wird. Schließlich demonstrierten wir die Anwendung der DSL anhand eines Praxis-Beispiels, der Entwicklung eines Web Formulars zur Abrechnung von Reisekosten.

Für die Zukunft planen wir, bestehende DSLs anhand der Erfahrungen in aktuellen und zukünftigen Projekten weiter zu verbessern und unseren DSL Katalog um DSLs zur Spezifikation weiterer Aspekte verteilter Web-basierter Systeme zu erweitern. Im Hinblick auf die in diesem Beitrag vorgestellte PetriX DSL streben wir beispielsweise die Integration weiterer dynamischer Interaktionsstrukturen an, um so noch dynamischere Formulare realisieren zu können. Auch im Hinblick auf die Spezifikation der Präsentationsaspekte eines Formulars sind noch Ergänzungen denkbar – jedoch immer unter Beibehaltung einer möglichst einfachen und intuitiven Notation.

Darüber hinaus arbeiten wir am Ausbau unserer technischen Infrastruktur, wie zum Beispiel ein auf Ontologien basierendes Klassifikationsschema für das Repository und einer integrierten Entwicklungsumgebung für DSL-basiertes Web Engineering.

7 Literatur

Beaufour, A., Olsen, M., Pettay, O. and Reed, A.: Mozilla XForms Project. http://www.mozilla.org/projects/xforms/ (2005).

Bentley, J.L.: Programming pearls: Little languages, Communications of the ACM, 29. Jg. (1986), Nr. 8, S. 711-721.

Ceri, S., Fraternali, P. and Bongio, A.: Web Modeling Language (WebML): A Modeling Language for Designing Web Sites, in Proceedings of the 9th International World Wide Web Conference (WWW); Amsterdam, Nethderlands; S. 137-157, 2000.

Deursen, A.v., Klint, P. and Visser, J.: Domain-Specific Languages: An Annotated Bibliography, ACM SIGPLAN Notices, 35. Jg. (2000), Nr. 6, S. 26-36.

Dubinko, M., Leigh L. Klotz, J., Merrick, R. and Raman, T.V.: XForms 1.0, W3C. http://www.w3.org/TR/2003/REC-xforms-20031014/ (2003).

Gaedke, M. and Nußbaumer, M.: Formularbasierte Benutzerinteraktion mit Fachkomponenten, in Proceedings of the 4. Workshop Komponentenorientierte betriebliche Anwendungssysteme (WKBA 4); Augsburg, Germany; Universität Augsburg; S. 45-55, 2002.

Gaedke, M., Nussbaumer, M. and Meinecke, J.: WSLS: An Agile System Facilitating the Production of Service-Oriented Web Applications, in: S.C. M. Maristella: Engineering Advanced Web Applications; 2005, S. 26-37.

Gaedke, M., Rehse, J. and Graef, G.: A Repository to facilitate Reuse in Component-Based Web Engineering, in Proceedings of the International Workshop on Web Engineering at the 8th International World-Wide Web Conference (WWW8); Toronto, Ontario, Canada; S., 1999.

Gaedke, M. and Turowski, K.: Integrating Web-based E-Commerce Applications with Business Application Systems, Netnomics Journal, Baltzer Science Publishers, 2. Jg. (2000), Nr. 2000, S. 117-138.

Juling, W.: KIM Project Homepage, University of Karlsruhe. http://www.kim.uni-karlsruhe.de/ (2005).

McCarron, S., Pemberton, S. and Raman, T.V.: XML Events, W3C. http://www.w3.org/TR/xml-events/ (2003).

Nielsen, J.: Usability Engineering; Morgan Kaufmann; San Francisco 1993.

Petri, C.A. (1962). Kommunikation mit Automaten. Dissertation, Darmstadt: Technischen Universität Darmstadt.

Schwabe, D., Rossi, G. and Barbosa, S.: Systematic Hypermedia Design with OOHDM, in Proceedings of the ACM International Conference on Hypertext' 96; Washington, USA; S., 1996.

The Standish Group International: CHAOS Research. http://www.standishgroup.com (1994-2005).