Universität Hannover
Fachgebiet Software Engineering
Institut für angewandte Systeme
Fachbereich Informatik

Vergleich von verschiedenen XML-GUI-Standards

Bachelorarbeit

im Studiengang Angewandte Informatik

von

Peng Xu
2090830

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Rainer Parchmann

Hannover, 16. Februar 2005

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

DANKSAGUNG

Ich möchte mich an dieser Stelle für die sehr gute Betreuung von Herrn Prof. Dr. Kurt Schneider und Herrn Dipl. Wirt. -Inf. Daniel Lübke und ebenfalls von Herrn Prof. Dr. Rainer Parchmann als Zweitprüfer recht herzlich bedanken.

Weiterhin möchte ich mich bei Herrn Sebastian Wittler für die Rechtschreibkorrektur bedanken. Für die moralische Unterstützung gilt mein Dank meinem Freund, meiner Familie und allen Freunden.

Kurzfassung

Bei traditionellen Entwicklungen von GUIs werden meistens imperative Programmiersprachen verwendet, dabei treten aber viele Probleme auf, z.B. Lokalisierung, Mischung von GUIs und Logik, schlechte Wartbarkeit usw. Aus diesem Grund erschienen einige XML-Dialekte, um solche Probleme zu beheben. Diese Arbeit beschäftigt sich zunächst mit einigen XML-GUI-Dialekten, nämlich XUL, XAML, XForms, UIML und der Qt-Designer. Danach werden die Eigenschaften der vorgestellten fünf Dialekte aus verschiedenen Sichten verglichen. Zur Veranschaulichung wird zum Schluss jeweils eine GUI in den fünf Dialekten erstellt.

Inhaltsverzeichnis

Seite

1 Einleitung

In diesem Kapitel werden die Problemstellung und Zielsetzung dieser Arbeit erläutert.

1.1 Problemstellung

Heutzutage spielen (graphische) Benutzeroberflächen für viele Systeme und Anwendungsprogramme eine wichtige Rolle. Sie können die Fähigkeit der Kommunikationen zwischen Rechner und Menschen verbessern. Einfache und schöne Benutzeroberflächen von Softwaren sind bei Kunden sehr beliebt.

Viele Firmen und Organisationen bemühen sich, Standards und Werkzeuge für Benutzeroberflächen zu entwickeln. Seit ein paar Jahren ist XML populär geworden. Vor allem wird XML für Datenmodellierung und –Austausch verwendet. Eine Reihe von XML-Dialekten wird für verschiedene Zwecke entwickelt. Wegen der vielen Vorteile, z.B. Klartext, gute Strukturierung usw. kommt XML bei der Entwicklung von Benutzeroberflächen in Frage.

Zur Beschreibung von Oberflächen sind z. Zt. am Markt viele XML-Dialekte erhältlich. Es soll nun evaluiert werden, welche Dialekte für welche Aufgaben gut oder weniger gut geeignet sind. Dazu ist ein Leistungskatalog zu erstellen, der als Grundlage für den Vergleich dient.

1.2 Zielsetzung

Die Aufgabe besteht darin, eine Übersicht von XML-Dialekten zur Beschreibung von Oberflächen zu erstellen und nach einem ebenfalls zu erstellenden Leistungskatalog zu vergleichen. Hierbei ist herauszuarbeiten, für welche Aufgaben sich welche Lösung am besten eignet. Zudem soll untersucht werden, ob es mit den Dialekten möglich ist, eine Oberfläche für einen Client zu beschreiben, der komplett von einem Server gesteuert wird.

1.3 Gliederung

Die nachfolgenden Kapitel dieser Arbeit gliedern sich wie folgt:

Kapitel 2 beschäftigt sich mit einigen Grundlagen von XML und davon verwandte Standards, dazu wird ein Beispiel gegeben. Anschließend werden die Grundlagen von XML-GUI-Dialekten kurz erwähnt.

Im Kapitel 3 werden fünf XML-GUI-Dialekte vorgestellt, nämlich XUL, XAML, XForms, UIML und der Qt-Designer.

Im Kapitel 4 wird ein Vergleich zwischen den fünf vorgestellten Dialekten anhand der Eigenschaften der Dialekte durchgeführt. Die Kriterien bei dem Vergleich werden nach verschiedenen Aspekten in 6 Tabellen abgelegt. Zum Schluss wird ein GUI-Beispiel gegeben, das in den fünf Dialekten geschrieben ist.

Abschließend befindet sich in Kapitel 5 eine Zusammenfassung.

2 Grundlagen von XML und XML-GUIs

Diese Arbeit beschäftigt sich mit XML-GUI-Standards. Um solche XML-GUI-Standards und deren Eigenschaften besser verstehen zu können, werden zunächst die Grundlagen von XML betrachtet. Das grundlegende Wissen von XML-GUIs wird danach behandelt.

2.1 Grundlagen von XML

XML ist innerhalb weniger Jahre zu einem der wichtigsten Datenformate und Präsentationsformate geworden. Um XML in der Praxis sinnvoll einsetzen zu können, gibt es mittlerweile eine Vielzahl begleitender Standards. Der Schwerpunkt dieser Arbeit ist die Benutzung von XML zur Beschreibung von Benutzeroberflächen. Deshalb gibt das Kapitel nur einen allgemeinen Überblick über die relevanten Standards von XML.

2.1.1 Herkunft und Motivation von XML

XML (*Extensible Markup Language*) ist eine erweiterbare Auszeichnungssprache (Markup-Sprache). Sie, sowie viele weitere Standards, wurden von dem W3C (*World Wide Web Consortium)* [1] entwickelt und als Standards herausgegeben. Das erste Ziel war es, mithilfe von XML nichtstatische Webseiten erzeugen zu können. XML ist eigentlich eine vereinfachte Form der SGML (*Standard Generalized Markup Language*), die seit Anfang der 80er Jahre als ein internationaler Dokumentations-Standard existiert.

HTML (*Hypertext Markup Language*) wird seit langem für den Entwurf von Webseiten verwendet, und XML sieht ein bisschen wie HTML aus. Aber im Gegensatz zu HTML ist XML eine Metasprache, mit der sich beliebige andere Auszeichnungssprachen definieren lassen. Solche Auszeichnungsprachen lassen sich problemorientiert definieren und einsetzen, d.h. XML ist „eine Menge von Regeln zur Schaffung von Auszeichnungssprachen“. Außerdem ist XML aber auch eine „Familie von Technologien, mit denen alles von der Formatierung von Dokumenten bis hin zur Filterung von Daten erledigt werden kann“ [2].

XML gehört zu den Technologien, die in den letzten Jahren schnell entwickelt und weltweit verwendet werden. Es lässt sich heutzutage fast überall XML entdecken. Insbesondere bei der Entwicklung web-basierter Anwendungen ist sie das beste Mittel, um die Kriterien der Plattform- und Herstellerunabhängigkeit zu erfüllen. Das W3C hat die folgenden zehn Punkte als plakative Kurzcharakterisierung von XML veröffentlicht [3]:

1. XML steht für strukturierte Daten.
2. XML sieht ein wenig wie HTML aus
3. XML ist Text, aber nicht zum Lesen.
4. XML ist vom Design her ausführlich.
5. XML ist eine Familie von Techniken.
6. XML ist neu, aber nicht so neu.
7. XML überführt HTML in XHTML.
8. XML ist modular.
9. XML ist die Basis für RDF (*Resource Description Framework*) und das Semantic Web.
10. XML ist lizenzfrei, plattform- und herstellerunabhängig, und gut unterstützt.

Außerdem hat XML folgende Charakteristika:

- Inhalt und Darstellung können getrennt sein.
- XML ist einfach zu interpretieren und zu verarbeiten.
- Es gibt strengere Fehlerüberprüfung für XML, d.h. XML ist einfach zu validieren.

2.1.2 Aufbau eines einfachen XML-Dokuments

Ein XML-Dokument besteht generell aus zwei Teilen: dem Dokumentprolog und dem Dokumentinhalt.

Der Dokumentprolog besteht aus einer oder mehreren Prozessinstruktionen und einer optionalen Deklaration der DTD (*Data Type Definition*), wobei die DTD als eine einfache XML-Grammatik betrachtet werden kann.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Telefonbuch SYSTEM " Beispiel.dtd">
```

Um dem Inhalt eines XML-Dokuments grammatikalisch reglementieren zu können, bietet der XML-Standard zwei unterschiedliche Ansätze an. Der erste ist die oben angesprochene DTD, die leider nur begrenzte Möglichkeiten bietet. Um komplexere Grammatiken erstellen zu können, muss der XML-Schema-Standard verwendet werden, der als eine komplizierte XML-Grammatik betrachtet werden kann.

Elemente sind die wichtigsten Bestandsteile der XML-Dokumente. Mit ihnen werden die Inhalte ausgezeichnet. Daher werden sie dem Dokumentinhalt zugeordnet.

```
<Telefonbuch name= "mein Telefonbuch">
   <Vorname>Peng</Vorname>
   <Nachname>Xu</Nachname>
   <Tel-Nr>0049-123-4567890</Tel-Nr>
</Telefonbuch>
```

2.1.3 XML und verwandte Standards

XML selber ist ein einfacher Standard, und wird von vielen anderen Standards begleitet. Viele Organisationen und Firmen haben einige von XML abgeleitete Standards entwickelt. Oft verwendeten Standards werden in Tabelle 2-1 dargestellt.

Präsentation und Interaktion	Datenschutz und -sicherheit	Protokolle und Services	Datenbanken und Wissen im Web
§ Multimedia: SMIL [4] § Grafik: SVG [5] § Publishing: (X)HTML [6], XSL [7], (CSS) [8], MathML [9] § Interaktion: XFORMS [10] § Mobilität und Geräteunabhängigkeit: CC/PP [11] VoiceXML [12]	§ Datenschutz: P3P [13] § Signatur: XML-Signature [14] § Verschlüsselung: XML-Encryption [15] § Schlüsselmanagement: XML Key Management [16]	§ XML Messaging und RPC: SOAP [17] § Description of Services: WSDL [18]	§ XML-Abfragesprache: XQuery [19], (XPath) [20] § Semantic Web [21]: RDF [22], OWL [23]
XML [24], Namensräume [25], XSLT [26], XPath, XLink [27], XPOINTER [28], XML BASE [29], XML Schema [30], DOM [31]			

Tabelle 2-1: XML und verwandte Standards [32]

Seit der Geburt von XML hat sich eine Vielzahl von Anwendungen entwickelt. Mögliche Anwendungsbereiche sind folgende:

- § Datenbankintegration,
- § Entwicklung für das Web,
- § XML-basiertes Messaging,
- § Metadaten erzeugen,
- § Serveranwendungen erstellen,
- § Einen Client erstellen.

Für die Erstellung von Benutzeroberflächen eignet sich XML sehr gut. Dies wird der Inhalt der folgenden Kapitel sein.

2.2 Grundlagen von XML-GUIs

Seit vielen Jahren werden verschiedene Beschreibungssprachen für Benutzer-oberfläche - UIDLs (User Interface Description Language) - entwickelt, um Mensch-Computer-Interaktion zu verbessern.

Solche UIDLs streben verschiedene Ziele an [33]:

- Plattformunabhängigkeit (plattform-neutral): Portierung der UI zwischen verschiedenen Plattformen.
- Wiederverwendbarkeit: Verbesserung der Wiederverwendbarkeit des UI-Designs.
- Geräteunabhängigkeit
- Erweiterbarkeit und Anpassungsfähigkeit
- Automatische Erzeugung des UI-Codes.

Es ist schwierig, alle Ziele zu erreichen, besonders wenn die Beschreibung von Benutzeroberflächen und Applikationslogik in fast jeder Entwicklungsumgebung (so genannt IDE, *Integrated Development Environment*) zusammen verarbeitet werden. Das bedeutet, dass die Entwicklung und die Wartung von Benutzeroberflächen sehr aufwändig sind und immer gute Programmierkenntnisse erfordern. Im optimalen Fall sollten grafische Designer die Arbeit für die Darstellung der modernen Benutzeroberflächen übernehmen, damit die Softwareentwickler sich auf Applikationslogik konzentriert können.

Außerdem gibt es immer dasselbe Problem: die Vielfältigkeit der Rechnerplattformen und die Vielfältigkeit der Entwicklungsumgebungen, d.h. auf verschiedenen Plattformen werden unterschiedliche Systemressourcen zur Verfügung gestellt und verschiedene Bibliotheken eingesetzt. Daraus folgt, dass es viel Arbeit benötigt, um eine Version eines Programms an allen Plattformen anzupassen. Es sollte eine Lösung gefunden werden, dass Benutzeroberflächen für alle Systeme einheitlich beschrieben werden könnten.

XML-GUI-Dialekte bieten die Möglichkeiten an, die beschriebenen Probleme zu lösen. XML-GUI-Dialekte sind Varianten von XML, mit denen Benutzeroberflächen generiert werden können. Sie streben an, dass die Struktur von Benutzeroberflächen und die Logik getrennt behandelt werden. Außerdem ist XML ein universales Datenaustauschformat, deswegen können die von XML-GUI-Dialekten beschreibenden Benutzeroberflächen problemlos auf verschiedenen Plattformen portiert werden. Der Vorteil von XML - Speicherung mit Unicode - ermöglicht außerdem die richtige Erstellung aller Zeichen in einem XML-Dokument.

3 XML-GUI-Dialekte

Dieses Kapitel beschäftigt sich mit verschiedenen XML-GUI-Dialekten. Zunächst wird XUL von Mozilla vorgestellt. Anschließend wird XAML behandelt, das von Microsoft entwickelt wird und als eine neue deklarative Sprache für die nächste Generation von Windows - „Longhorn“ - gedacht ist. Danach werden die Eigenschaften der XForms vom W3C betrachtet. Zum Schluss wird einen Überblick über andere XML-GUI-Dialekte (UIML und Qt Designer) gegeben.

3.1 XUL

In diesem Abschnitt werden die Geschichte und die Eigenschaften von XUL kurz erläutert. Anhand eines Beispiels wird dann der Aufbau einer XUL-Datei erläutert. Zwei Varianten, Mozilla-XUL und Lutor-XUL, werden zum Schluss detailliert betrachtet.

3.1.1 Herkunft von XUL

XUL steht für *XML User-Interface Language* [34] und ist eine von Mozilla entwickelte und auf XML basierende Auszeichnungssprache. Sie bietet die Möglichkeit, die plattformunabhängigen grafischen Benutzeroberflächen von Applikationen schnell und einfach zu erstellen und modifizieren.

XUL wurde ursprüngliche für den Open Source Webbrowser Mozilla (Netscape 6) im Rahmen des XPToolkit-Projekts (*Cross Platform Toolkit*) [35] entwickelt. Die Sprache entstand aus dem Grundgedanken, dass Mozilla nicht nur als Webbrowser sondern auch als Entwicklungsframework betrachtet werden kann [36]. Ab Netscape 6.x wird XPFE (*Cross Platform Front End*, wobei „Front End“ bedeutet „graphisches Anzeigen in Mozilla“) [37] verwendet, die eine auf XUL basierende Entwicklungsumgebung ist. XPFE besteht hauptsächlich aus drei Komponenten, nämlich: XUL, CSS (*Cascading Style Sheets*) und JavaScript. Dieser Gedanke ist ähnlich wie bei XHTML (*Extensible Hypertext Markup Language*), wobei XHTML aus HTML, CSS und JavaScript besteht.

XPFE hat folgende Struktur:

- Inhalte: XUL dient dazu, die Struktur von Benutzeroberflächen einer Applikation zu beschreiben.
- Layout: CSS wird verwendet, um die Themes (Look&Feel) darzustellen.
- Funktionalität: JavaScript verwirklicht den Applikationslogik.

Zwei Begriffe müssen dabei klar unterschieden werden:

XUL ist nicht verantwortlich für die Darstellung oder das Layout einer Benutzeroberfläche, diese Aufgabe übernimmt CSS. JavaScript behandelt Events und Benutzerinteraktionen.

3.1.2 Eigenschaften von XUL

Der Erfolg von XUL wird von dessen Eigenschaften garantiert. Vor allem hat XUL alle Eigenschaften von XML. Es wurde entwickelt, um plattformunabhängige Applikationen zu realisieren. Dazu stehen ausreichende GUI-Elemente zur Verfügung. Mit XUL können fast alle GUI-Elemente erstellt werden, unter anderem: Buttons, Labels, Bilde, Eingabe-Steuerelemente (z.B. Textboxen, Checkboxen), Werkzeugleisten, Menüs, Menüleisten, Popupmenüs, Dialoge, Bäume usw. In XUL können HTML, JavaScript, CSS und vieles mehr integriert werden.

Außerdem hat XUL einige Besonderheiten:

- XUL hat eigenen Namensraum: *http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul.*
- XUL hat kein Standard.
- XUL hat keine offizielle DTD oder ein Schema.

Hier werden ein paar Vorteile bzw. Nachteile von XUL in Tabelle 3-1 aufgelistet.

Vorteile	**Nachteile**
§ Plattformunabhängigkeit § Trennung von Aussehen und Struktur der Oberfläche § Schnelle Entwicklung § Freier Quellcode § Klartext (einfach zu verarbeiten) § Ausreichende Grundelemente § Auch für Entwicklung von Webapplikationen geeignet § Wiederverwendbarkeit § Basiert auf offenen Standards.	§ Noch nicht reif genug, muss noch weiter bearbeitet werden § Keine Standardisierung § Langsamere Verarbeitung als native Benutzeroberflächen § Umständliche Portierung des Mozilla-XUL-Toolkits § Es kann nur in Mozilla-Browsern verwendet werden.

Tabelle 3-1: Vorteile bzw. Nachteile von XUL

3.1.3 Aufbau einer XUL-Datei

Bevor die Struktur von XUL vorgestellt wird, wird zuerst eine kurze aber typische XUL-Datei als Beispiel gegeben.

```
1  <?xml version="1.0"?>
2  <?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
3  <!DOCTYPE window>
4  <window
5  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
6  <label value=" hello, world!" />
7  </window>
```

Nachdem diese XUL-Datei in Mozilla geladen wird, sieht es wie Abbildung 3-1 aus:

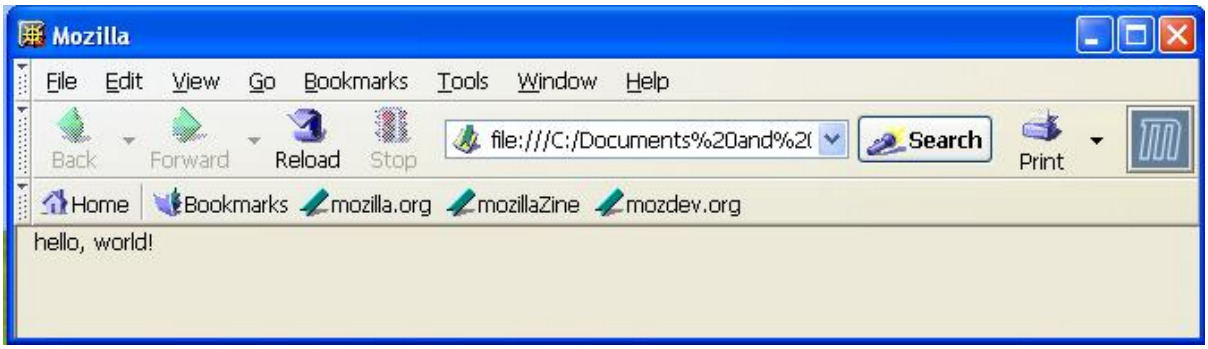

Abbildung 3-1: Beispiel „hello world “ in Mozilla.

Anhand der Zeilenummern wird diese XUL-Datei kurz erklärt.

1) XUL ist zuerst ein XML-Dokument, daher die XML Deklaration.
2) Die Deklaration des Stylesheets. Es wird festgelegt, welches Stylesheet für die Darstellung verwendet werden soll. Hier wird eine „chrome://“-URL verwendet, um die Datei *global.css* zu laden. (Mehr Informationen zu „chrome://“-URL sind in dem Abschnitt 3.1.4.3 zu finden.)
3) Deklaration des Dokumententyps.
4) Die XUL-Datei beschreibt ein Fenster. (Die Auszeichnung ist „window“)
5) Deklaration des Namensraums für XUL.
6) Hier wird ein *Label* deklariert. Die Basis jeder XUL-Applikation ist ein XUL Fenster. Zwischen <window> und </window> werden alle Kindelemente deklariert.
7) Abschließen von „ window “.

Abbildung 3-1 (Beispiel „hello world “ in Mozilla) hat eine einfache GUI gezeigt. Das Aussehen ist nicht so phantastisch wie gedacht, weil das nur eine sehr kurze XUL-Datei ist. *Firefox* [38] und *Thunderbird* [39] sind zwei Produkte von Mozilla, deren Benutzeroberflächen in XUL erstellt sind. Mit XUL und anderen Technologien zusammen hat Mozilla schöne Software geliefert.

3.1.4 Mozilla XUL

XUL wurde ursprünglich von Mozilla entwickelt. Der Abschnitt gibt einen Überblick über die Struktur von Mozilla-XUL und über einige Technologien.

3.1.4.1 Struktur von Mozilla-XUL

Die Philosophie von XUL ist die logische Trennung von vier Komponenten einer GUIs:

- Inhalt : Struktur und Beschreibung von UI-Elementen.(z.B. XUL, XHTML usw.)
- Aussehen: Skin oder Theme (*Look & Feel*) genannt. (z.B. CSS, Image)
- Verhalten: Applikationslogik wird durch Skriptsprache mit UI verbunden. (z.B. JavaScript)
- Lokalisierung: Dadurch können Applikationen einfach in verschiedenen Sprachen für unterschiedliche Regionen genutzt werden. (DTD und .properties-Datei)

Entsprechend besteht die Struktur einer XUL-Applikation aus 4 Arten von Dateien, nämlich der XUL-Datei (.xul), CSS-Datei (.css), Javascript-Datei (.js) und DTD-Datei (.dtd). (Natürlich gibt noch mehrere Dateien, um andere Technologien mit XUL zu kombinieren). Normalerweise werden XUL-Dateien und Javascript-Dateien in dem Unterverzeichnis „*content*“, CSS-Dateien in dem Unterverzeichnis „*skin*“ und DTD-Dateien in dem Unterverzeichnis „*locale*“ gespeichert. Außerdem gibt es unter „*locale*“ ein Unterverzeichnis für jede Code-Region, z.B. „*en-US*“ für die USA, „*de-CH*“ für die Schweiz.

3.1.4.2 Mozilla-XUL mit JavaScript und CSS

Wenn Mozilla gestartet wird, werden XUL-Dateien gelesen und interpretiert, d.h. es werden die GUI-Komponenten (Widgets) angezeigt und die entsprechenden Event-Handler initialisiert. Die Zusammenarbeit von XUL, JavaScript und CSS wird hier detailliert beschrieben.

XUL wird zur Beschreibung von Widgets verwendet. In einer XUL-Datei wird entweder ein Stylesheet direkt angegeben oder ein Verweis auf einer externe .css-Datei bekannt gegeben, um das Layout von Benutzeroberflächen darzustellen. Ähnlich ist es bei Javascript. In Mozilla spielen XUL, CSS und Javascript verschiedene Rollen, deswegen ist es besser, wenn die drei einzeln gespeichert werden. In diesem Fall ist das anstrebende Ziel „Wiederverwendbarkeit“ besser zu erreichen.

CSS:

CSS legt das Aussehen von Benutzeroberflächen fest und wird extern als eine *.css*-Datei gespeichert. Die .xul-Datei wird so codiert, dass sie mit einer CSS-Datei zusammenarbeiten kann, z.B.:

```
<? xml-stylesheet href="chrome://global/skin/" type="text/css" ?>
```

Das Stylesheet global.css, das bereits in Mozilla als Basis-Style vordefiniert ist, wird importiert.

```
<? xml-stylesheet href="chrome://navigator/skin" type="text/css" ?>
```

Ähnlich wie oben, aber nur das von Mozilla vordefinierte navigator.css wird importiert,

Natürlich kann eine eigene CSS-Datei verwendet werden. Dies wird durch Einsatz von folgendem Code realisiert:

```
<? xml-stylesheet href="chrome://my-style/skin" type="text/css" ?>
```

JavaScript:

Ohne Skriptsprache hat XUL wenig Sinn. Mit JavaScript können viele Funktionalitäten erweitert werden. Folgend wird ein Abschnitt einer XUL-Datei angezeigt. Es wird in dem Fall ein *beispiel.js* aufgerufen.

```
<script type="application/x-javascript"
src="chrome://beispiel/content/ beispiel.js" />
```

3.1.4.3 Verwendete Technologien

XUL kann wie HTML in Mozilla-Browser benutzt werden, d.h. sie kann von entfernten Webseiten oder vom lokalen Dateisystem geladen werden. Außerdem strebt Mozilla an, sich als ein Framework zu betrachten, deswegen bietet Mozilla eine externe Funktion an, dass alle gepackten Dateien über eine Chrome-URL verfügbar sind. In Mozilla haben ein Paket und eine Applikation die gleiche Bedeutung [40]. Ein Paket kann als Applikation installiert und in Mozilla registriert werden, dann kann es in einem eigenen Fenster wie eine Applikation ausgeführt werden. Hier werden einige Begriffe eingeleitet: Paket, Chrome, Chrome-URL, und Registrieren.

Paket

Alle Dateien, welche eine Mozilla-Applikation ausmachen, werden in eine JAR-Datei gepackt. Ein Paket kann drei Verzeichnisse enthalten, wie oben schon erwähnt wurde, nämlich „*content*“, „*skin*“, „*locale*“. Natürlich kann jedes Verzeichnis je nach Bedarf getrennt gepackt werden.

Chrome

Im installierten Verzeichnis von Mozilla gibt es ein Unterverzeichnis „chrome“. Unter diesem Verzeichnis werden alle von Mozilla vordefinierten Pakete gespeichert.

Außerdem werden alle selbst erstellten Pakete auch hier abgelegt. Durch so genannte Chrome-URLs können auf alle Ressourcen einfach zugegriffen werden. „chrome://“ ist ein Art von URL: wie die „http://“-URL eine Referenz auf eine Website ist, so wird die „chrome://“-URL immer als eine Referenz auf die eigenen Ressourcen (z.B. Paket) verwendet, Außerdem bietet eine „chrome://“-URL die Möglichkeit an, dass eine Mozilla-Applikation auf ein paar lokale Daten mit bestimmten Zugriffsrechten zugreifen kann. Im Gegensatz dazu haben normale Webpages oder Applets kein Recht, auf lokale Dateien zuzugreifen. Abbildung 3-2 zeigt den Inhalt eines Verzeichnisses „*chrome*“.

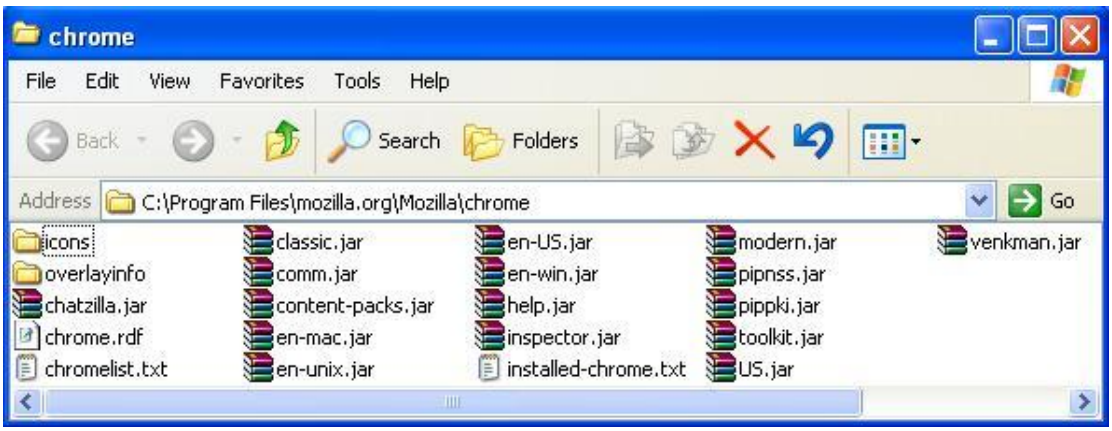

Abbildung 3-2: Fenster von dem Verzeichnis „*chrome*“

Ein Paket registrieren

XUL-Dateien können als eine Applikation behandelt und in einem eigenen Fenster ausgeführt werden, müssen davor aber zunächst in Mozilla registriert werden. Bei Mozilla ist ein Mechanismus XPInstall (*Cross Platform Install*) [41] vorhanden, durch das alle gebrauchten Dateien gepackt werden und auf jeder Plattform installiert werden können.

XPCOM /XPConnect

Ein noch zu lösendes Problem ist: wie komplizierte Applikationslogik mit einer XUL-GUI kommunizieren könnte. Mozilla verwendet ein Objektssystem namens XPCOM (*Cross-platform Component Object Model*) [42]. In C++ werden die XPCOM-Komponenten geschrieben, die bestimmte Logik realisieren können.

JavaScript kann von XUL direkt aufgerufen werden. XPConnect [43] ist eine Brücke zwischen JavaScript und XPCOM-Komponenten. Daher kann XUL durch den Einsatz von JavaScript und XPCOM/XPConnect mit anderen Programmiersprachen kombiniert werden.

Im Folgenden wird noch ein Beispiel von Mozilla-XUL gegeben, das Ergebnis wird in einem Mozilla-Browser in Abbildung 3-3 dargestellt, und als eine standalone Applikation in Abbildung 3-4 angezeigt.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<!DOCTYPE window>
<window title="Hello World!"
xmlns:html=http://www.w3.org/1999/xhtml
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
style="background-color: white;"
  width="300"
  height="150"
  onload="centerWindowOnScreen( )">
<script type="application/x-javascript"
  src="chrome://global/content/dialogOverlay.js" />

<vbox>
  <hbox align="left">
    <label value ="Hello, World from" />
    <image src="xul.jpg" />
  </hbox>
  <button label="click me!" oncommand="alert('Hello World');" />
</vbox>

</window>
```

Abbildung 3-3: Laden des Beispiels in Mozilla-Browser

Abbildung 3-4: Ausführung des Beispiels als eine standalone Applikation.

An dieser Stelle wurden die meisten von Mozilla-XUL verwendeten Technologien bekannt gegeben. Der nächste Abschnitt geht kurz darauf ein, wie weit das XUL von Luxor entwickelt ist.

3.1.5 Luxor XUL

Neben Mozilla-XUL gibt es noch andere Firmen, die sich mit XUL beschäftigen. Eine davon ist Luxor. Ihr XUL-Version ist nicht nur ein Klon von Mozilla-XUL, sondern besitzt auch eine Reihe von eigenen Funktionalitäten.

Ein paar Funktionen von Luxor-XUL sind mächtig, z.B. Mini Web-Server, der eine „peer-to-peer“ Funktion ermöglicht; Portal-Engine, Template-Engine und ein auf Python basierter Skript-Interpreter. Die Tools sind in reinem Java- programmiert und die generierten GUI basieren auf Java-Swing.

Tabelle 3-2 zeigt den Unterschied zwischen Mozilla- und Luxor-XUL [44].

Art	Luxor	Mozilla
Funktionalität	Anwendungssprache :Java Skriptsprache: Python	JavaScript, andere Sprache durch XPCOM realisierbar
Templates	Apache Velocity plus XSL/T	Auszeichnung: <template>
Web Server	Integriert	Nein
Portal Engine	Integriert	Nein
Installation	„Web Start“ von Sun	XPInstall von Mozilla
CSS-Unterstützung	Nur Intern	Intern und Extern
Start-Verzeichnis	„Startup“	„Content“
Registrierung nötig	Nein, Web Start, JNLP sorgt dafür.	Ja

Tabelle 3-2: Der Unterschied zwischen Luxor- und Mozilla-XUL

Wobei JNLP für *Java Network Launch Protocol* steht.

Im Moment ist Luxor-XUL noch nicht so reif wie Mozilla-XUL. Es fehlt an guter Dokumentation, im Moment sind nur ein paar Beispiele von Luxor-XUL vorhanden.

3.2 XAML

In diesem Teil wird *Longhorn* [45] - eine neue Generation des Windows-Systems von Microsoft - kurz erwährt, dann wird eine mit Longhorn gleichzeitig entstandene Auszeichnungssprache XAML (*Extensible Application Markup Language*) [46], die einer der zukünftigen XML-GUI-Dialekte ist, vorgestellt.

3.2.1 Herkunft von Avalon XAML

Um den Hintergrund von XAML besser verstehen zu können, wird zuerst Longhorn betrachtet.

Longhorn führt ein neues Applikationsmodell ein, durch das Microsoft anstrebt, die Unterschiede zwischen traditionalen Desktop-Applikationen und Webapplikationen zu eliminieren und die Vorteile der beiden auszunutzen.

Die Longhorn-Applikationen werden in einem so genannten „single unified program model“ [47] geschrieben, d.h. Desktop- und Webapplikationen haben dasselbe Programmiermodell. Um das Ziel zu erreichen, wird eine Trennung von Präsentation, Daten und Kommunikation in Longhorn realisiert, indem drei wichtige Komponenten bzw. Subsysteme von Longhorn, nämlich *Avalon*, *WinFS* und *Indigo*, eingesetzt werden. Die Architektur von Longhorn wird in Abbildung 3-5 dargestellt.

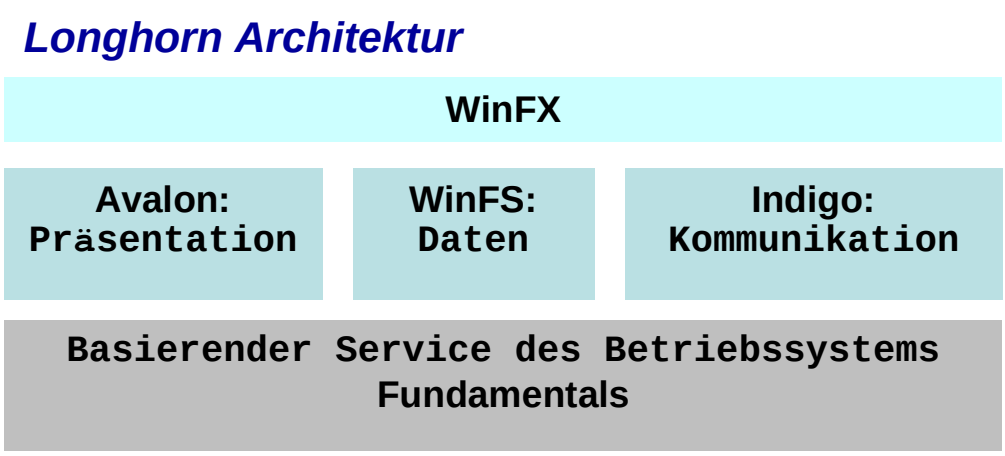

Abbildung 3-5: Architektur von Longhorn

Die Komponenten sind dabei:

- *Avalon* ist ein Präsentationssubsystem von Longhorn.
- *WinFS* ist eine Abkürzung von „Windows Future Storage“, die eine neue Technologie zur Datenspeicherung von Longhorn ist.
- *Indigo* ist das Kommunikationssubsystem in Longhorn.

WinFX (Windows Framework Extensions), die neueste Version von DotNET-Framework in Longhorn, enthält die drei oben genannten Subsysteme.

Als Präsentationsschicht von Longhorn liefert Avalon eine „Avalon-Engine“ und ein „Avalon-Framework“ [48]. Die Avalon-Engine ermöglicht eine einheitliche Behandlung von Dokumenten, Medien-Dateien und Benutzeroberflächen, während das Avalon-Framework ein GUI-Framework vom Windows-System repräsentiert. Durch das Framework kann das UI-Layout einfacher und besser dargestellt werden. Als Weiteres umfasst das Framework ein neues Programmiermodell, das eine objektorientierte Klassen-Bibliothek ist, die über ein Objektmodell verfügt. Daher definiert Avalon eine neue XML-basierte deklarative Sprache XAML. Jedes Element in XAML entspricht einer Klasse im Avalon-Framework. Bei einer Longhorn-Applikation werden das GUI-Layout und die Applikationslogik getrennt behandelt. Zur Darstellung von Benutzeroberflächen wird XAML eingesetzt, damit werden die Vorteile der deklarativen Programmierung ausgenutzt.

Außerdem kann Avalon webähnliche Desktop-Applikationen erstellen. Microsoft will erreichen, dass eine traditionale Desktop-Applikation oder eine Webapplikation gleich erzeugt bzw. betrachtet werden können. Es entspricht dem von Longhorn eingeführten Programmiermodell.

3.2.2 Grundlagen von XAML

XAML kann als eine Kombination der Funktionalitäten von HTML, SVG (*Scalable Vector Graphics*) und CSS betrachtet werden. XAML-Dokumente sind in HTML-ähnlicher Syntax zu verfassen. Für die Präsentation von Benutzeroberflächen verwendet XAML kein CSS, das bereits von dem W3C als Layout-Stil standardisiert ist, sondern eigene Elemente. Auch für die Vektor-Darstellung verwendet XAML bzw. Avalon nicht den Vektorgrafik-Standard SVG, sondern eigene Methoden, die bislang als WVG (*Windows Vector Graphics*) bekannt waren. (Microsoft will jedoch den Namen nicht mehr benutzen!)

Es ist bekannt, dass XAML für das Layout von Longhorn-Applikationen verwendet wird, aber für die Realisierung der Applikationslogik und der Verarbeitung von Events wird entsprechend einer von DotNET mitgelieferte Programmiersprache, z.B. C# oder VB.NET (*Visual Basic.NET*) ausgewählt und eingesetzt.

In einer Longhorn-Applikation kann der Code entweder in einer XAML-Datei als Codeblöcken (so genannt „*Inline Code*“) eingebettet werden oder in einer separaten Datei (so genannt „*Code-behind*“) abgelegt werden. Bei der Änderung von Layouts (nicht UI-Struktur) ist es nicht erforderlich, den Code der Applikationslogik zu

verarbeiten. Alle durch XAML realisierten UI-Layouts können durch DotNET realisiert werden. Es gibt folgen Vorteile, wenn in XAML programmiert wird [49]:

- Bessere Sichtbarkeit von der Hierarchie der Steuerelemente
- Bessere Sichtbarkeit von der Vererbung der Attribute
- Einfache Verarbeitung und Interpretation der Auszeichnungssprache durch Tools
- Trennung von UI und Logik.

Wenn eine XAML-Datei keinen Code enthält, kann die Datei direkt wie eine Webseite in den IE von Longhorn geladen werden. Wenn der Code in einer XAML-Datei erfasst ist oder die Applikation aus XAML-Dateien und Code-Dateien besteht, muss die XAML-Datei kompiliert werden, weil der IE von Longhorn den Code in C# oder in VB.NET nicht interpretieren kann.

Eine Longhorn-Applikation kann entweder als eine standalone Applikation ausgeführt werden oder als eine Webapplikation in einen Browser geladen werden, d.h. eine Applikation kann entweder (wie eine traditionelle Windows-Applikation) auf Formen basieren, oder (wie eine Webapplikation) auf Seiten basieren. In diesen beiden Fällen muss die XAML-Datei nicht geändert werden, aber sie muss entsprechend noch mal kompiliert werden. Außerdem entspricht jede XAML-Datei einer Seite (*page*), d.h. wenn eine Applikation aus mehren Seiten (*pages*) besteht, soll sie als navigierte Applikation kompiliert werden können, wobei eine navigierte Applikation eine Hauptseite (main page) hat, in der Buttons für Navigation (z.B. „*next*“, „*back*“) vorhanden sind.

3.2.3 Aufbau einer XAML-Datei

Jede XAML-Datei hat ein Wurzel-Tag, das aber nicht eindeutig festgelegt ist. Normalerweise ist ein Wurzel-Tag ein Container von Elementen. Jedes in XAML enthaltenes Element entspricht einer (Unter)Klasse vom Avalon-Framework, und ein Attribut stimmt mit einer Property oder einem Event einer Klasse überein. Die Einstellungen der Attribute in einer XAML-Datei können durch die Einstellungen der Property in der entsprechenden Code-Datei durchgeführt werden.

In einer XAML-Datei werden die Namensräume mehrfach deklariert, aber normalerweise werden sie in einem Container-Element bekannt gegeben. Jeder Namensraum bietet einen URI (*Uniform Resource Indicator*) an, der auf eine Datei hingewiesen. In dieser Datei werden alle Namensräume von DotNET aufgelistet, in den alle Klassen von DotNET deklariert sind [50].

An dieser Stelle wird eine XAML-Datei (ohne Logikcode) als Beispiel gegeben [51].

```
<Canvas
    xmlns="http://schemas.microsoft.com/2003/xaml"
    Background="LightCyan" Width="100%" Height="100%">
    <Image Source="lh.bmp" Canvas.Left="5" Canvas.Top="5" />
    <Text Canvas.Left="90" Canvas.Top="20" FontSize="36">
      Hello, Longhorn!
    </Text>
</Canvas>
```

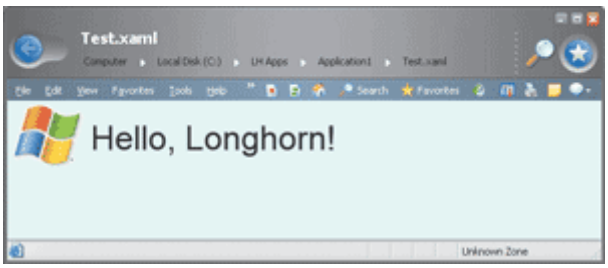

Abbildung 3-6: Laden die XAML-Datei in Internet Explorer Browser von Longhorn

In diesem Beispiel wird ein Wurzel-Tag „Canvas“ zunächst als ein Container definiert. Dann kommt die Deklaration der Namensräume von XAML. Anschließend werden ein Image- und ein Text-Element definiert und einige zugehörige Attribute gesetzt. Diese XAML-Datei enthält keinen Code für Applikationslogik, deswegen kann sie direkt in den IE von Longhorn geladen und angezeigt werden. Das Ergebnis des Beispiels wird in Abbildung 3-6 dargestellt.

Wenn ein XAML-Element definiert wird, wird eine Instanz einer Klasse im Avalon-Framework während der Laufzeit erzeugt und nach einer interpretierten Hierarchie auf dem Bildschirm angezeigt. Dieser Verlauf ist eine dynamische Erstellung der UI und geschieht nur, wenn eine XAML-Datei keinen Code für Applikationslogik enthält.

Wenn eine XAML-Datei einen Codeblock enthält oder es eine „Code-behind“-Datei gibt, dann muss zuerst die XAML-Datei kompiliert werden. Zurzeit ist ein Programm „MSBuild.exe“ vorhanden, um eine XAML-Datei zu kompilieren. Nach dem Erscheinen der neuen „Visual Studio.NET“ mit Codename „*Whidbey*“ [52] (voraussichtlich im Jahr 2005) wird dies vereinfacht.

Nach dem Kompilieren wird für jede XAML-Datei eine Quellcode-Datei von DotNET erzeugt, die „.cs“ (in C#) oder „.vb“ (in VB.NET) als Endung hat. In der Tat hat diese Kompilation die Aufgabe, eine Abbildung einer XAML-Datei in den entsprechenden Quellcode durchzuführen, d.h. diese XAML-Datei wird in einer Quellcode-Datei von DotNET übersetzt. Im Quellcode ist eine Definition einer Klasse zu finden, diese ist eine von dem Wurzel-Tag der XAML-Datei vererbte Unterklasse, z.B. eine XAML-Datei heißt „MyCanvas.xaml“, und deren Wurzel-Tag ist „Canvas“. Der entsprechende Quellcode sieht wie folgendes aus [50]:

```
public partial class MyCanvas :
MSAvalon.Windows.Controls.Canvnas { ... }
```

Das Schlüsselwort „partial“ bedeutet, dass der Quellcode eine Übersetzung einer XAML-Datei ist.

Nach der Kompilation wird die XAML-Datei nicht mehr verwendet, sondern der Quellcode.

3.2.4 Elemente von XAML

XAML besitzt ausreichende Elemente nicht nur für UI-Layout, sondern auch für Animation, Text-Format usw. Die Elemente werden in vier Kategorien eingeteilt, nämlich „*Panels*“, „*Controls*“, „*Text Formatting*“, und „*Shapes*“ [49].

Die *„Panels“*-Elemente werden verwendet, um Layouts von Seiten zu kontrollieren. Außerdem können „Panels“-Elemente als Container für andere Elemente verwendet werden. Dazu gehören sechs Elemente: „Canvas“, „DockPanel“, „FlowPanel“, „TextPanel“, „GridPanel” und „FixedPanel“. Jedes Element davon stellt ein besonderes Layout dar.

Die Steuerelemente aus der Kategorie „Controls“ dienen zur Verarbeitung von Benutzersinteraktionen. In Abbildung 3-7 werden ein paar Steuerelemente angezeigt.

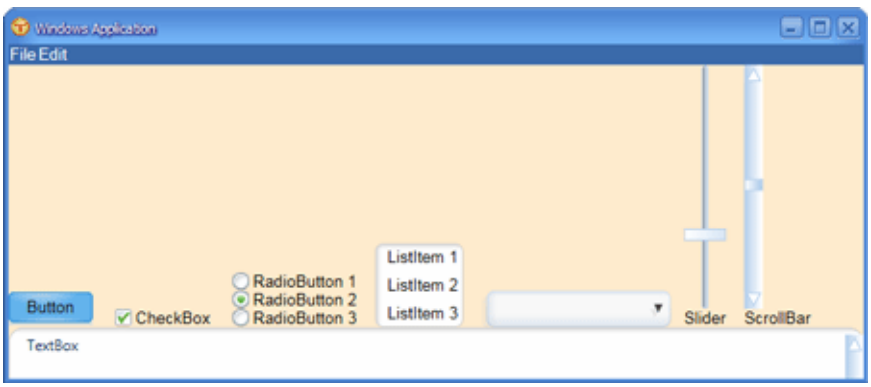

Abbildung 3-7: Steuerelemente von XAML in Longhorn [49]

Die Elemente aus der Kategorie „Text Formatting“ beschäftigen sich mit dem Formatieren von Text und mit der Struktur von Dateien.

Avalon basiert auf Vektorgrafik. Die „Shaps“-Elemente von XAML sind dafür zuständig, Vektorgrafik zu zeichnen. Die Elemente bestehen aus Ellipse (*Ellipse*), Linie (*Line*), Path (*Path*), Rechteck (*Rectangle*), Polygon (*Polygon*) und Polylinie

(*Polyline*). Des Weiteren können mit XAML einige Transformations-Effekte realisiert werden. Außerdem ermöglicht XAML Animationen.

Einige besondere Funktionalitäten von XAML zur Verarbeitung von Dokumenten sind erwähnenswert, z.B. das so genannte „Dokument Service“. XAML ermöglicht es, Online-Dokumente besser darzustellen und zu lesen. Es gibt zwei wichtige Services: „*PageViewer Control*“ und „*Document Layout Services*“.

„PageViewer Control“ ist zuständig für das Anzeigen von Online-Dokumenten, während „*Document Layout Services*“ die Funktion anbietet, das Layout von Dokumenten besser darzustellen. Die ähnlichen Funktionalitäten und Eigenschaften von PDF (*Portable Document Format*) werden durch XAML mit der Hilfe von „*Document Layout Services*“ realisiert.

Aus den Kenntnissen aus dem vorangehenden Abschnitten folgt, dass XAML nicht nur zur Erstellung von Benutzeroberflächen entwickelt wurde, sondern auch versucht, die Vorteile von PDF und Flash zu nutzen.

3.3 XForms

HTML-Formulare bestimmen derzeit viele Webpräsenzen, aber was die nächste Generation, die XML-basierten XForms (*Extentional Forms*), zu bieten hat, könnte das Herz vieler Webautoren höher schlagen lassen.

3.3.1 Hintergrund und Eigenschaften von XForms

Für Webanwendungen spielt HTML-Form immer eine wichtige Rolle. Formulare sind Benutzerschnittstellen von Webseiten. Durch ein Formular bietet HTML die Möglichkeit an, Interaktionen zwischen Benutzer und entferntem Server zu realisieren, indem Informationen von Benutzer gesammelt und zu einem Server verschickt werden.

In den letzten Jahren haben sich Technologien für Webapplikationen, z.B. E-Business weltweit schnell entwickelt und sind immer verbessert worden. XML und XHTML sind zwei häufig eingesetzte Technologien dafür. HTML-Form hat wegen ihrer vielen Beschränkungen keine ausreichenden Fähigkeiten, die heutigen Bedürfnisse von Webanwendungen zu erfüllen, z.B. ist HTML-Form von Geräten abhängig, da die Bildschirme von PDAs und Handys niedrigere Auflösungen als PCs haben. Es ist deshalb schwierig, HTML-Form richtig und effizient einzusetzen. In dieser Situation hat das W3C die nächste Generation der Webformulare XForms entwickelt, das den Status einer W3C-Recommendation erreicht hat.

Als Nachfolger der HTML-Form ist XForms eine Anwendung von XML, die die nächste Generation von Formularen für das Web präsentiert. XForms hat alle Funktionalitäten von HTML-Form, außerdem hat XForms die Nachteile von HTML-Form durch die Integration der XML-Technologien und die Ergänzung der Interaktionen beseitigt. XForms besitzt ausreichende Eigenschaften, um die Bedürfnisse von Unternehmern, Verbraucher und Anwendungen zu erfüllen. Ein Vergleich von XForms und HTML-Form wird in der Tabelle 3-3 erstellt.

HTML-Form [53]	**XForms**
Keine Fähigkeit von lokalen Datentypen-überprüfungen.	Lokale Datentypenüberprüfung während der Eingabe von Daten
Mischung von Präsentationen, Daten und Funktionen.	Trennung von Präsentationen, Daten und Funktionen
Funktionen durch Skripte realisiert.	Kein Erfordernis von Skripten
Schwer zu integrieren mit anderen Datentypen (z.B. XML).	Einfache Integration mit SVG, XHTML und anderen XML-Standards

Geräteabhängigkeit	Geräteunabhängigkeit
Keine eingebettete Logik	Formulare mit Logik
Anpassung an einfache Formulare	Unterstützung von komplizierten Formularen
Schwierigkeit beim vorherigen Datenladen in einem Formular.	Fähigkeit von vorherigen Datenladen
Schwer zu verwalten	Einfach zu verwalten
Webbrowser erforderlich zum Anzeigen von HTML-Form	Webbrowser nicht erforderlich zum Anzeigen von XForms
Schwierigkeit beim Internationalisieren von Zeichencode	Unterstützung des Internationalisieren von Zeichencode
Zugänglichkeitsprobleme	Unterstützung von Zugänglichkeit

Tabelle 3-3: Vergleich von HTML-Form und XForms

Neben den Vorteilen gegenüber HTML-Form hat XForms weitere nennenswerte Eigenschaften, wie z.B.

(a) Trennung der Daten und der Logik von Präsentationen

XForms wird in keiner eigenen Datei, sondern wird in einer so genannten „Host-Datei“ z.B. XHTML, SVG integriert. Das ursprüngliche Ziel von XForms war Trennung der Daten und der Logik von Präsentationen, indem eine Definition von XForms in einer Host-Datei in drei Teile („XForms-Modell“, „Instance“-Daten und „Benutzeroberflächen“) eingeteilt werden kann. Genauere Informationen dazu sind im Abschnitt 3.3.2 zu finden

(b) Integration von XML

XForms ist richtig mit XML bzw. anderen XML-Technologien integriert.

§ XForms basiert auf XML.
§ Formulare können durch lokale oder externe XML-Dateien initialisiert werden, d.h. Daten können vorher in Formularen geladen sein.
§ Die in Formular gesammelten Daten (durch „Submit“-Operation) werden als XML behandelt, d.h. Daten werden in Unicode entweder durch das Internet zum Server transportiert oder in XML-Dateien lokal gespeichert. Des Weiteren ist es möglich:

  - Submitten eines Formulars zu mehreren Servern.
  - Die übermittelten Daten als Eingabe von anderen Formularen

- Die übermittelten Daten sind XML-Dateien in Unicode. Das bedeutet, dass die Internationalisierung einfach implementiert werden kann.
- Kombination mit existierenden XML-Technologien, z.B. mit XPath, XML-Schema usw. XPath wird verwendet, um Daten zu lokalisieren und kalkulieren, und existierende XML-Schemas werden eingesetzt, um die Datentypen ohne Skripte zu definieren und zu validieren.

(c) Vollständigkeit der Datentypenüberprüfung

XForms hat die Fähigkeit, bestimmte Eigenschaften einer eingegebenen Datei durch XForm-fähige Browser lokal überprüfen zu lassen, z.B. die Typen von eingegebenen Daten, Eingabe eines erforderlichen Feldes oder die Reihfolge von Eingaben. Alle Daten werden ohne Unterstützung von Skripten lokal validiert, d.h. die Daten mussten nicht mehr hin und zurück zum Server transportiert werden, um die Gültigkeit von Daten zu überprüfen. XML-Schema ermöglicht dies, weil XML-Schema die Fähigkeit hat, Typen von Daten festzulegen und das Format von Daten zu überprüfen. Daher wird die Fähigkeit der Interaktion zwischen Benutzer und Server verbessert.

(d) Weniger Verwendung von Skripten

Durch Einsatz von XPath und XML-Schema werden Skripte selten verwendet. Außerdem behandelt XForms Events durch so genannte XML-Events [54], die die meisten Events umfassen. Wegen der Eigenschaften von XML-Events können die meisten Events in XForms ohne Skripte behandelt werden. In XForms können folgende Aktionen von entsprechenden XML-Events realisiert werden [55]:

- Setzen des Fokus von Steuerelementen
- Anzeigen von Messages
- Navigierung zu einer neuen URI (im selben Fenster oder im neuen Fenster)
- Veränderung von relevanten Daten bei irgendeiner Änderung in Felder.
- Aktualisierung des Bildschirms bei Rekalkulationen oder Revalidierungen
- „Submitten“ oder Rücksetzen von ganzen oder partiellen „instance“-Daten.

3.3.2 Struktur von XForms

XForms hat das traditionale HTML-Form in drei Komponenten eingeteilt: XForms-Modell, „instance“-Daten, und Benutzeroberflächen. Es gibt drei entsprechende logische Komponenten, nämlich Logik, Daten und Präsentation.

(a) XForms-Modell

Im „XForms-Modell“ sind der Inhalt bzw. der Zweck eines Formulars enthalten, z.B. Sammlung von Daten, Such-Engine usw. Es ist unabhängig von der Präsentation, d.h. wenn ein Formular auf unterschiedenen Geräten oder Ansichten betrachtet wird, bleibt das Modell erhalten. Ein Modell besteht aus zwei Teilen: der Struktur der Daten und der Logik. In einem Modell wird die Struktur von zu verarbeiteten Daten definiert, und die Logik definiert das Verhalten von Formularen.

Zwischen den Tags <*instance*> und <*/instance*> werden Datenstrukturen definiert, genauer gesagt werden alle Datenelemente, die den Präsentationen entsprechen, deklariert, z.B.:

```
<xforms:model…>
    <xforms:instance />
       <Addresse>
          <Strasse/>
          <PLZ/>
          <Ort/>
       </Addresse>
    <xforms:instance />
……
</xforms:model>
```

XForms unterstützt einige Kalkulationen, dafür werden alle betroffenen Variablen in dem „instance“-Element definiert. D.h. hier können einige Daten-Items als Konstante oder „temporary“-Variable definiert werden. In diesem Beispiel werden „instance“-Daten in dem Modell „lokal“ definiert, alternativ können sie auch eine entfernte XML-Datei mit Hilfe einer URI referenzieren, z.B.:

```
<xforms:instance href="http://www.my-
xforms.com/Beispiel.xml"/>
```

Der andere Teil des XForms-Modells ist die Definition der Logik. Das Verhalten von Formularen ist durch ein Element „*submission*“ definiert. Darin wird das Verhalten eines Formulars nach der Sammlung der Daten festgelegt. Außerdem können mehrere „*submission*“-Elemente durch eine eigene ID für unterschiedliche Zwecke definiert werden, z.B.:

```
<xforms:submission id="act01" action="http://www.my-
xforms.com/" method="post"/>

<xforms:submission id="act02" method="put"
action=file:///temp/instance-daten-export.xml />
```

In “act01” werden die Daten zum Server verschickt, während die Daten in „act02“ in einer lokalen XML-Datei gespeichert werden, d.h. XForms kann auf Dateien entweder auf dem Server oder auf der lokalen Maschine zugreifen.

Ein anderes wichtiges Element in XForms-Modell ist das Element „bind“, durch das die Daten entweder von Benutzer eingegeben werden können, oder von anderen Daten abhängen. In „bind“ werden die Beschränkungen von Daten angegeben. Es wird besonders für die Elemente verwendet, die als „temporary“-Variable definiert sind und nicht in Benutzoberflächen präsentiert werden. Durch die Attribute von „bind“ werden durch solche „temporary“-Variablen die Verbindungen mit anderen Daten geschaffen.

(b) Benutzeroberflächen

Die Benutzeroberflächen in XForms ermöglichen Interaktionen zwischen Benutzern und Servern. XForms besitzt ein paar atomare Steuerelemente für die Darstellung von Formularen, z.B. „input“, und ein paar Elemente zum Anordnen von atomaren Steuerelementen, z.B. „repeat“. Alle Elemente sind abstrakt definiert, d.h. die Elemente können bei verschiedenen Geräten bzw. Plattformen problemlos nach eigenem Darstellungsstil implementiert werden. Eine typische Anwendung ist das „*select1*“-Element, das im Browser als „drop-down“-Menü dargestellt ist, während es im PDA (*Personal Digital Assistant*) als Radio-Button präsentiert ist [56]. Trotz der verschiedenen Darstellungen wird die gleiche Funktion (nämlich ein einziges Item auswählen) implementiert.

Abbildung 3-8: „select1“ in Form eines „drop-down“-Menüs [107].

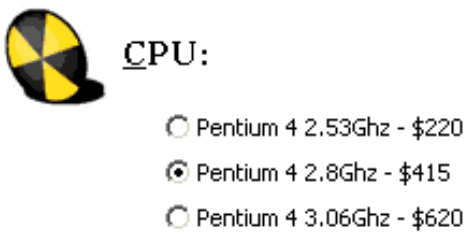

Abbildung 3-9: „select1“ in Form von Radio-Buttons [78].

(c) Verbindung des XForms-Modells mit Benutzeroberflächen

Nach dem Kennenlernen des XForms-Models und der Präsentation wird eine Frage sehr oft gestellt, nämlich wie die beiden verknüpft werden sollen. Eine XML-Technologie namens XPath wird dafür eingesetzt. Es ist bekannt, dass eine XML-Datei ein hierarchischer Baum ist. XPath greift auf die Daten in XML durch einen hierarchischen Pfad zu.

Durch die Verwendung von XPath wird die Struktur der „instance“-Daten (in dem XForms-Modell) mit den Steuerelementen (in Benutzeroberflächen) verbunden. Eine der wichtigen Funktionen von XForms ist die Überwachung der Änderungen von Steuerelementen. Eine andere ist die Kalkulation. In HTML-Form werden solche Funktionen durch Skriptsprachen realisiert. In XForms werden diese Aufgaben durch XPath-basierte Attribute von Elementen erfüllt.

Die Zusammenarbeit vom XForms-Modell, Benutzeroberflächen und „instance“-Daten werden in Abbildung 3-10 dargestellt. Hierbei wird das „XForms Submit Protocol“ für die Sendung und den Empfang von Daten eingesetzt [57].

Zusammengefasst ist es ersichtlich, dass XForms ganz mit XML integriert ist. Der Zugriff von Daten wird durch XPath realisiert, XML-Schema legt die Datentypen fest und die Events werden durch XML-Events behandelt.

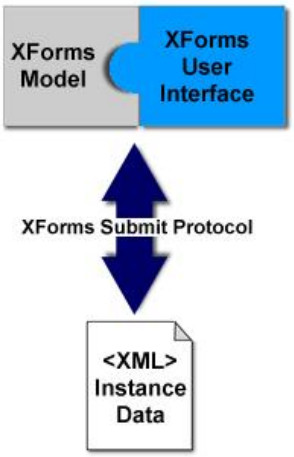

Abbildung 3-10: Zusammenarbeit von XForms-Modell, Benutzeroberflächen und „instance“-Daten

3.3.3 Aufbau eines Formulars in XHTML

Im Folgenden wird ein Skelett einer XHTML-Datei mit XForms gegeben:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xf="http://www.w3.org/2002/xforms">
<head>
    <style type="text/css">
       <!-- Definition von CSS, optimal-->
    </style>

    <xf:model >
      <xf:instance>
       <!--definition von instance-Daten-->
      </xf:instance>
```

```
<!-- Behandlungen von Events, Aktionen, Daten-Bindungen usw.-->

      <xf:submission />
      <xf:bind/>   <!--"bind"-Element ist optimale-->
    </xf:model >

</head>
    <body>
      <!-- UI von XForms und anderen Deklarationen von XHTML-->
   </body>
</html>
```

Dieses Skelett zeigt, dass ein Stylesheet in der XHTML-Datei vorhanden ist. Das Stylesheet kann auch von Präsentationen weiterverwendet werden.

An dieser Stelle wird ein kleines Beispiel gegeben [56].

```
<!--Beispiel Hello World-->
<html xmlns= "http://www.w3.org/1999/xhtml"
xmlns:xforms="http://www.w3.org/2002/01/xforms"
xmlns:my="http://www.example.com/my-data">
       <head>
         <xforms:model>
            <xforms:instance>
               <my:data>Hello World</my:data>
            </xforms:instance>
         </xforms:model>
       </head>
       <body>
          <xforms:group>
              <xforms:output ref="/my:data">
                  <xforms:label>Output Control Example
               </xforms:label>
              </xforms:output>
          </xforms:group>
       </body>
</html>
```

Bei dem Beispiel werden zunächst die Deklarationen von Namensräumen durchgeführt. Der Restteil passt zu dem oben gegebenen Skelett. Ein Attribut „**ref**“ dient dazu, die Struktur von „instance“-Daten (in dem Modell) mit den Streuerelementen zu verbinden. Bei dem Beispiel wird ein „*output*“-Element, dessen „instance“-Daten „Hello World“ sind, in den Namensraum „*my*“ abgebildet. *"/my:data"* ist ein XPath-Ausdruck. Das Ergebnis sieht wie Abbildung 3-11 aus:

Output Control Example
Hello World

Abbildung 3-11: „Hello World” in XForms

3.4 UIML

UIML (*User Interface Markup Language*) [58] wird in diesem Abschnitt als einen weiteren XML-GUI-Dialekt vorgestellt.

3.4.1 Hintergrund und Grundkenntnisse von UIML

In den letzten Jahren werden zahlreiche mobile Geräte, wie z.B. PDA und Handy, auf den Market gebracht. Es war immer die Zielsetzung, eine Applikation auf verschiedenen Geräten und/oder auf unterschiedlichen Plattformen auszuführen. Ein Bespiel dafür: ein elektronischer Kalender auf PC, PDA und Handy.

Um das Ziel zu erreichen, sind manche Schwierigkeiten zu überwinden. Verschiedene Geräte haben verschiedene Darstellungsfähigkeiten, z.B.: die Größe des Bildschirms, Methoden für Ein- und Ausgabe usw. Trotz gleicher Funktionalität muss eine traditionale Applikation für unterschiedliche Geräte bzw. Plattformen in verschiedenen Programmiersprachen mehrmals entwickelt werden. Wegen diesem Umstand entwickelte der Firma *Harmonia* [59] UIML, um auf allen Geräte bzw. Plattformen passende Benutzeroberflächen darzustellen.

Die Geräteunabhängigkeit und die Plattformunabhängigkeit von UIML werden durch das so genannte „generic vocabulary“ realisiert, d.h. ein allgemeines Vokabular von UI-Elementen wird eingesetzt. Damit können die UI-Elemente von allen Geräten in einer Familie möglichst einfach identifiziert werden.

Da fast jedes Gerät bzw. jede Plattform eine eigene Entwicklungssprache für Applikationen hat, z.B. HTML für Webbrowser, WML (*Wireless Markup Language*) [60] für Handys und PDAs， wurden für UIML einige Renderer entwickelt, durch die eine allgemeine UIML-Datei in eine Zielsprache konvertiert und interpretiert wird. Mit Hilfe einer solchen Transformation ist die Geräte- bzw. Plattformunabhängigkeit dann möglich. Die momentan von UIML unterstützten Zielsprachen sind unter anderem Java, HTML, WML und VoiceXML. Daher wird sich UIML wohl als eine Allzwecksprache für Benutzeroberflächen etablieren.

Mit „generic vocabulary“ können Benutzeroberflächen nur sehr beschränkt beschrieben werden. Deswegen bietet UIML für jede Zielsprache einen Renderer und ein entsprechendes Spezialvokabular zur Verbreitung der Darstellungs-fähigkeiten an.

Eine Benutzeroberfläche in UIML besteht aus einigen „part“-Elementen. Jedes Element hat ein Klasse- und ein ID-Attribut. Jedes Klasse-Attribut mit gleichem

Namen wird bei Transformationen in eine Zielsprache gleichartig behandelt. Jedes „part“-Element hat jedoch eigene Eigenschaften.

Die Namen der Klassen- und Eigenschafts-Attribute sind kein Bestandteile von UIML, sondern ein Vokabular, das je nach der Zielsprache unterschiedlich ist. UIML hat nur wenige Elemente [61], mit denen abstrakte Beschreibungen der Benutzeroberflächen ermöglicht werden. Ausreichende Darstellungen von Benutzeroberflächen werden durch Vokabulare erweitertet.

UIML ist auch unabhängig von Benutzeroberflächenmetaphern, z.B. graphische Benutzeroberflächen oder „Sprachausgabe“, d.h. UIML ist nicht nur ein GUI-Dialekt, sondern auch geeignet für nicht-graphische Benutzeroberflächen, z.B. VoiceXML. Außerdem kann UIML entweder interpretiert werden, z.B. in einem Webbrowser als HTML-Seite, oder in Zielsprachen kompiliert werden, z.B. in WML oder in Java.

3.4.2 Struktur von UIML

UIML besitzt drei Funktionen zur Beschreibung der Benutzeroberflächen, nämlich Aussehen, Benutzerinteraktion und Verbindung mit der Applikationslogik. UIML beschreibt Benutzeroberflächen abstrakt, indem die Beschreibung in sechs Bereiche (Struktur, Stil, Inhalt, Verhalten, Präsentation und Logik) geteilt wird. Die Zusammenarbeit der eben genannten Bereiche wird in Abbildung 3-12 gezeigt.

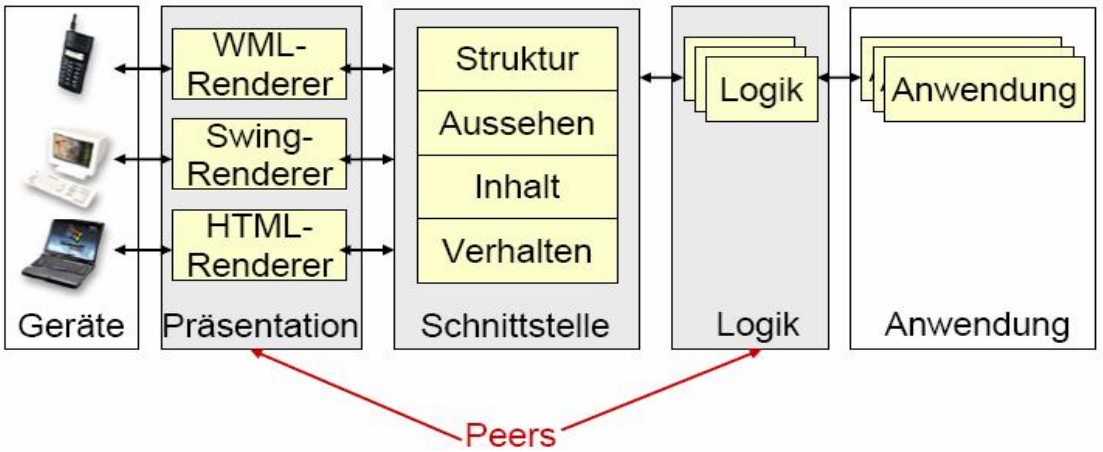

Abbildung 3-12: Struktur von UIML [62]

Ein Skelett einer UIML-Datei sieht wie folgt aus:

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 3.0 Draft//EN"
"http://uiml.org/dtds/UIML3_0a.dtd">
<uiml xmlns='http://uiml.org/dtds/UIML3_0a.dtd'>
  <head> ... </head>
```

```
  <template> ... </template>
  <interface>
     <structure> ... </structure>
     <style> ... </style>
     <content> ... </content>
     <behavior> ... </behavior>
  </interface>
  <peers>
     <presentation> ... </presentation>
     <logic> ... </logic>
  </peers>
</uiml>
```

Es ist zu sehen, dass das „uiml“-Element die allgemeine Beschreibung in UIML, die ganzen Definitionen von UIML (außer einem XML-Prolog) umfasst.

Zuerst kommt das „head“-Element, das Metadaten für UIML enthält, z.B. Daten vom Autor. Durch das „template“-Element ist es dann möglich, einige Blocks von UI-Darstellungen wieder zu verwenden. Die beiden Elemente sind optional. Weiterhin sind wichtige Teile von UIML:

(a) „Interface“-Element : Schnittstelle

- Struktur-Beschreibung (<structure>): Logische Komponenten der Oberfläche und ihre Anordnung.
- Styl-Beschreibung (<style>): Darstellung der Komponenten, z.B. Farbe, Zeichensatz usw.
- Inhalt-Beschreibung (<content>): darzustellende Daten, z.B. Texte, Bilder.
- Verhalten-Beschreibung (<behavior>): Verhalten bei aktiven Events.

(b) „peer“-Element: Beschreibung der Abbildung in einer entsprechenden Zielsprache.

- Präsentation (<presentation>): Darstellungen auf den unterstützten Geräten. Vokabulare werden in diesem Teil eingesetzt. Für unterschiedliche Geräte, Plattformen bzw. Zielsprachen kann das „presentation“-Element mehrmals definiert werden.
- Logik (<logic>): Anbindung an die Anwendungen (Geschäftslogik, Datenquellen usw.).

An dieser Stelle wird ein Beispiel „Hello World“ gezeigt [61]:

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 3.0 Draft//EN"
"http://uiml.org/dtds/UIML3_0a.dtd">
<uiml>
```

```
    <interface>
      <structure>
        <part id="TopHello">
           <part id="hello" class="helloC"/>
        </part>
      </structure>
      <style>
        <property part-name="TopHello"
          name="rendering">Container
        </property>
        <property part-name="TopHello" name="content">Hello
        </property>
        <property part-class="helloC" name="rendering">String
        </property>
        <property part-name="hello" name="content">Hello World!
        </property>
      </style>
    </interface>

    <peers>
      <presentation name="WML">
        <component name="Container" maps-to="wml:card">
           <attribute name="content" maps-to="wml:card.title"/>
        </component>
        <component name="String" maps-to="wml:p">
           <attribute name="content" maps-to="PCDATA"/>
        </component>
      </presentation>
    </peers>
</uiml>
```

Die WML-Datei, die von dem obigen UIML-Code generiert wird, sieht wie folgt aus:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.0//EN"
"http://www.wapforum.org/DTD/wml.xml">
<wml>
  <card title="Hello">
     <p>Hello World!</p>
  </card>
</wml>
```

Dieses Beispiel hat gezeigt, wie eine UIML-Datei durch Abbildung des „peer“-Elementes in eine entsprechende Zieldatei konvertiert wird. Da die UIML-Datei nur abstrakt Benutzeroberflächen beschreibt, sieht eine UIML-Datei komplizierter als eine WML-Datei aus. Außerdem muss für jede Zielsprache ein eigenes Vokabular eingesetzt werden und ein entsprechendes „peer“-Element angegeben werden. Falls mehrere Zielsprachen erfordert sind, muss für jede Zielsprache eine eigene Beschreibung angeboten werden, die sehr kompliziert und aufwändig ist.

3.5 Qt-Designer

Der Qt-Designer wurde von der norwegischen Firma *Trolltech* entwickelt, die auch für die Qt-Grafikbibliothek zuständig ist. Der wohl bekannteste „Kunde“ von Trolltech ist das Open Source Projekt KDE (*K Desktop Environment*) [63], welches komplett auf der Qt-Bibliothek aufgebaut wurde. Mit dem Qt-Designer ist es möglich, ähnlich wie in Visual Basic eine GUI zu erstellen. Der Qt-Designer 3.3 ist momentan die neuste Version, er ähnelt stark der Entwicklungsumgebung Delphi bzw. C++-Builder. Im Folgenden soll er behandelt werden.

3.5.1 Technologien rund um den QT-Designer

Um den QT-Designer besser zu verstehen, sollten zunächst folgende Begriffe kurz erklärt werden.

Qt ist ein C++-Toolkit der Firma Trolltech für die Entwicklung von GUI-Applikationen, das sich in letzter Zeit zunehmender Beliebtheit erfreut. Grund ist vor allem die einfach zu benutzende und plattformunabhängige API.

KDE ist eine moderne grafische Arbeitsumgebung für auf Linux, Unix und anderen Betriebssystemen basierende Arbeitsstationen, und vor allem ist KDE benutzerfreundlich, komfortabel und intuitiv.

C++ ist eine auf C basierende Programmiersprache für allgemeine Anwendungen und stellt Sprachmittel für abstrakte Datentypen sowie modulare, generische, objektorientierte und strukturierte Programmierung zur Verfügung.

Linux ist ein frei verfügbares Multitasking- und Multiuser-Betriebssystem, das von Linus Torvalds und vielen freien Entwicklern weltweit entwickelt wird. Linux bietet mittlerweile all die Funktionalität, die man von modernen Betriebssystemen erwartet. Bekannte Distributionen von Linux sind Suse, Red Hat, Mandrake und Debian GNU.

3.5.2 Grundlagen von Qt

Qt ist hauptsächlich ein objektorientiertes GUI-Toolkit für C++. Darüber hinaus bietet es viele Utility-Klassen an, z.B. Ein- / Ausgabe, XML, lokalisiertes String-Handling (Unicode) und Zugriff auf SQL-Datenbanken. Das Qt-Framework umfasst ca. 420 Klassen.

Im Vergleich zu MFC (*Microsoft Foundation Classes*), DotNet (für Windows), Gtk [64] und Motif [65] (für X Windows) läuft Qt auf sehr vielen Plattformen (Unix, Mac OS X, Windows (ab 95)), ohne den Code für jedes System neu schreiben zu müssen (solange der Quellcode keine betriebssystemspezifischen Systemaufrufe enthält,

lassen sich Qt-Applikation auf allen unterstützten Plattformen ohne Änderung übersetzen.)

3.5.3 Signale und Slots

Um die Kommunikation zwischen Qt-Objekten zu realisieren wird im Qt ein so genannter „Signal/Slot“-Mechanismus eingesetzt, der ähnlich dem von „Callbacks“ in anderen GUI-Werkzeugen ist. In Qt wird ein Signal von einem Widget ausgesendet, wenn ein bestimmtest Ereignis eintritt. Ein Slot ist einfach eine Memberfunktion, die als Antwort auf das Signal aufgerufen wird [66].

3.5.4 Entwurf und Implementation von Qt-GUIs im Qt-Designer

Qt bietet dem Entwickler viele verschiedene Möglichkeiten, ansprechende graphische Benutzungsoberflächen zu erstellen. Da diese Arbeit normalerweise sehr zeitaufwendig und langweilig sein kann, wurden GUI-Builder realisiert, die dem Entwickler diese Aufgabe erleichtern sollen. Qt enthält zu diesem Zweck den Qt-Designer, mit dem graphische Oberflächen erstellt und in vorhandene Projekte eingebunden werden können. Daher ist der Qt-Designer in der Tat ein RAD-Tool (*Rapid Application Development*).

Ähnlich wie in Visual Basic werden im Qt-Designer die Oberflächen im grafischen Modus per Drag & Drop erstellt. Dabei sind verschiedene Standardelemente der QT-Bibliothek wie z.B. Buttons oder Treeviews vorhanden. Der Qt-Designer kreiert aus dem „zusammengeklickten“ Widget eine XML-Datei, aus der durch einen Parser Quellcode generiert wird.

Der Qt-Designer kann als Tool des XML-GUI-Dialekts betrachtet werden, weil das native Datenformat des Designers (*.ui) wohlgeformtes XML ist. Die von dem Qt-Designer erstellte .ui-Datei enthält die XML-Beschreibung der GUI und kann mit *uic* (user interface compiler) in C++-Header (*.h-Datei) und C++-Quellcode (*.cpp-Datei) umgewandelt werden. Der Qt-Designer beschreibt die GUI nicht direkt mit C++-Quellcode, sondern durch .ui-Dateien als Brücke. Ansonsten sind die Änderungen der GUIs im Qt-Designer komplizierter.

Der mögliche Verlauf der Entwicklungen mit dem Qt-Designer kann beispielsweise so sein:

1. Es wird ein Dialog, Widget oder MainWidget mit dem Designer erstellt, eine so genanntes Form: MyForm.ui im XML-Format.
2. Der *uic* erzeugt daraus C++-Dateien: MyForm.cpp und MyForm.h (der Befehl „*uic*“ wird zwei Mal aufgerufen).

3. *moc* (meta object compiler) kümmert sich um alle C++-Dateien und baut sie zusammen.
4. Kompilierung der von *moc* generierten Dateien, alle Objekt-Dateien werden zusammen gelinkt.
5. Es steht nun eine Klasse MyForm zur Verfügung, die weiter in C++-Projekten verwendbar ist.

3.5.5 Das .ui-Dateiformat

Qt-Designer speichert „Forms“ in .ui-Dateien. Diese Dateien benutzen das XML-Format, um Formelemente und ihre Merkmale darzustellen.

Abbildung 3-13: Ein Währungsumrechner [67]

Der folgende Codeblock aus der .ui-Datei entspricht der Abbildung 3-13

```
<!DOCTYPE UI>
<UI version="3.2" stdsetdef="1">
<class>umrechner</class>

<widget class="QDialog">
    <property name="name">
        <cstring>umrechner</cstring>
    </property>
    <property name="geometry">
        <rect>
            <x>0</x>
            <y>0</y>
            <width>361</width>
            <height>246</height>
        </rect>
    </property>
......
</widget>
<includes>
......
```

```
</includes>
<layoutdefaults spacing="6" margin="11"/>

</UI>
```

Das Wurzelelement ist ein "UI", der den ganzen Inhalt einschließt.

```
<UI version="3.2" stdsetdef="1">
...
</UI>
```

Hier wird ein Dialogbox „*umrechner*“ deklariert. Weiterhin werden einige kleine Widgets durch ihren Namen, Klasse und entsprechende Attribute (<*property*>) beschrieben. Außer den Deklarationen von „Widget“-Elementen können weitere Elemente innerhalb des UI-Elementes enthalten sein, z.B.: connections, functions, includes, layoutdefaults, signals, slots und variables.

In einem „connections“-Element können mehre „connection“-Elemente deklariert werden. In „signals“-, und „slots“-Elementen werden alle Signale und Slots definiert. Durch die Kombinationen von „connections“-, „signals“- und „slots“-Elementen wird der Signal/Slot Mechanismus in Qt eingesetzt.

```
<connections>
    <connection>
        <sender>......</sender>
        <signal>......</signal>
        <receiver>......</receiver>
        <slot>......</slot>
    </connection>
  ......
</connections>
......
<slots>
    <slot>......</slot>
</slots>
< signals>
    <signal>......</ signal>
</signals>
```

An dieser Stelle wird nicht jedes Element betrachtet. Ausführliche Informationen dazu sind online bei [68] zu finden.

3.6 Andere XML-GUI Dialekte

Außer den vorgestellten XML-GUI-Dialekten gibt es noch andere Dialekte. Hier werden nur einige davon kurz aufgelistet.

(a) eNode [69]:
Das Ziel von eNode ist Erstellung dynamischer Benutzeroberflächen und Behandlung des Webs und des Desktops in der gleichen Weise. eNode ist ein auf Java-basierter XML-GUI-Dialekt.

(b) XUI *(eXtensible User Interface)* [70]:
XUI ist ein Framework von „Java und XML“, es dient dazu, Applikationen für „rich Client“, Desktop und Mobile zu erstellen.

(c) AUIML *(Abstract User Interface Markup Language)* [71]:
Die vom AUIML-Toolkit erzeugten Benutzeroberflächen können für Swing von Java oder für HTML-Servlets verwendet werden. Dazu gibt es den „Interface-Builder“, der mit dem Eclipse Visual Editor zusammenarbeiten kann.

(d) Glade [72]:
KDE wurde mit Hilfe von Qt in C++ entwickelt, Während Gnome, ein Konkurrent von KDE, mit Hilfe von Gtk in C entwickelt wurde. In Qt ist der Qt-Designer das so genannte RAD-Tool, entsprechend in Gtk Glade. Mit Glade können Benutzer-oberflächen im XML-Format gespeichert und in C Code konvertiert werden.

(e) Flex und MXML [73]:
Flex von Macromedia ist ein Präsentationsserver, um RIAs (*Rich Internet Applications*) zu entwickeln. Dazu gibt es MXML, eine Markup-Sprache von Flex.

4 Vergleich der XML-GUI Dialekte

Zum Vergleich der vorgestellten Dialekte werden insgesamt sechs Tabellen aufgestellt. In jeder Tabelle werden einige Eigenschaften angegeben und verglichen. Anmerkungen zu jeder Eigenschaft sind hinter der entsprechenden Tabelle zu finden. Zum Schluss wird ein GUI-Beispiel gegeben, das in allen Dialekten geschrieben ist.

4.1 Allgemeine Eigenschaften

	XUL	XAML	XForms	UIML	Qt-Designer
A1 Firma / Organisation	Mozilla	Microsoft	W3C	Harmonia / OASIS	Trolltech
A2 Entwicklungsziel	GUIs für Mozilla Suits	GUIs für Longhorn	Nachfolger von HTML-Form	portierbare GUIs	schnelle GUI-Entwicklung in Qt
A3 Standardisierung	noch keine	keine	offene	offene	keine
A4 Open Source	ja	nein	ja	teilweise	ja
A5 Kosten	kostenlos	kommerziell	kostenlos	kostenlos/ kommerziell	kostenlos/ kommerziell

Tabelle 4-1: Allgemeine Eigenschaften

A1 Firma / Organisation

Die Firma Mozilla hat XUL ursprünglich entwickelt. Es existieren einige große Projekte zur Weiterentwicklung von XUL, z.B. Luxor XUL mit dem Ziel, auf Java-basierte Tools und auf Java-Swing basierte GUIs zu genieren. XAML wurde von der Firma Microsoft für das neue Windows-System Longhorn entwickelt. In Bezug auf XAML ist ein Open Source Projekt myxaml [74] zu finden. UIML wurde von der Firma Harmonia Inc. mit der Unterstützung der Organisation OASIS (*Organization for the Advancement of Structured Information Standards*) [75] entwickelt.

A2 Entwicklungsziel

Am Anfang wurde XUL zur Darstellung der GUIs mit dem neuen Netscape Browser (Mozilla) entwickelt, weiterhin wurde XUL eingesetzt, um das Ziel „building cross-platform user interfaces as easy as building web pages“ zu erreichen.

XAML wurde als eine deklarative Auszeichnungssprache für Avalon (die GUI-Klassen bzw. das GUI-Framework von Longhorn) vorgestellt, um Desktop- und Webapplikationen zu entwickeln, Webseiten darzustellen und druckbare Dateien zu erzeugen.

XForms wurde vom W3C als Nachfolger von HTML-Form entwickelt und empfohlen, um Interaktionen zwischen Benutzern und Servern zu verbessern und Funktionalitäten wie z.B. E-Commerce zu ermöglichen.

Das Entwicklungsziel von UIML ist die Erzeugung von UIs für verschiedene Geräte bzw. Plattformen in einer einzigen Sprache.

Das Ziel von dem Qt-Designer ist ein RAD-Tool zu entwickeln, um die Erstellung von Anwendungen auf der Basis der Qt-Bibliothek schnell und einfach zu ermöglichen.

A3 Standardisierung

Ein Standard von XUL existiert gegenwärtig noch nicht, aber er wurde als Standard beim W3C vorgeschlagen. XAML ist eine Spezifikation für Microsoft selbst. XForms wurde im Oktober 2003 vom W3C standardisiert. Die aktuelle Version ist 1,1. UIML ist ein Standard von der Organisation OASIS und wurde als Standard beim W3C vorgeschlagen. Die neueste Version ist 3,0.

A4 Open Source

Es geht hier um „Open Source“ der Produkte, der Sprachen oder des Programms. Die meisten Browser von XUL und XForms sind Open Source. Manche Renderer von UIML sind Open Source, manche aber nicht.

A5 Kosten

Es geht hier um die Kosten der entsprechenden Produkte, die für Anzeigen und Weiterverarbeitung nötig sind. Die Browser von XUL sind kostenlos. XAML ist ein eigenes Produkt von Microsoft für den kommerziellen Zweck. Für XForms gibt es kommerzielle und kostenlose Browser. UIML ist eine offene Spezifikation. Die Renderer von UIML sind aber kommerziell. Für Linux und Mac ist der Qt-Designer kostenlos. Im Windows-System ist er kostenlos nur für nicht-kommerzielle Applikationen.

4.2 Standards und Technologien

		XUL	XAML	XForms	UIML	Qt-Designer
B1	Namens-räume	ja	ja	ja	ja	keiner
B2	Schema / DTD	keins	Schema, noch nicht validiert	Schema	DTD	keins
B3	Event-Handling	JavaScript	DotNET	XML-Events	UIML	Signal/Slot
B4	Stylesheet	CSS	Attribute von Elementen	CSS und XSL	„style“-Element	eigene Elemente
B5	(X)HTML	kombinierbar	alternative	Host-Sprache	konvertierbar	kein Zusam.
B6	UI-Elemente	ausreichend	ausreichend	genügend	je nach Zielsprache	Qt-Bibliothek
B7	Grafik-Rendering	SVG (in Arbeit)	WVG	SVG	SVG möglich	OpenGL(3D) Canvas (2D)
B8	Animation	nein	unterstützt	SMIL	SMIL möglich	C++
B9	Web-Services: SOAP	ja	ja	noch nicht standardi-siert	ja	durch Plugins
B10	Integration mit anderen Techno-logien	HTML,CSS, DOM, XSLT JavaScript, XLink, RDF, SOAP, XHTML, , und WSDL. (MathML, SVG in Arbeit)	die von Longhorn unterstützten Techno-logien	SVG, XHTML, XPath, XLink, XML-Schema, XML-Events, und CSS.	Java, HTML, WML, VoiceXML DotNET, ASP und JSP. [61]	die von Qt unterstützten Techno-logien

Tabelle 4-2: Standards und Technologien

B1 Namensraum

Außer dem Qt-Designer haben auch andere Dialekte eigene Namensräume:

- § XUL: http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul.
- § XAML: http://schemas.microsoft.com/2003/xaml
- § XForms: http://www.w3.org/2002/xforms
- § UIML: http://uiml.org/dtds/UIML3_0a.dtd

B2 Schema / DTD

XUL hat kein offizielles Schema oder DTD. XAML hat ein eigenes Schema, das bis jetzt noch nicht ganz validiert ist. Wenn das Longhorn SDK installiert ist, wird per default die Schema-Datei xaml.xsd unter dem Verzeichnis *c:\program files\microsoft visual studio .net whidbey\common7\packages\schema\xml* abgelegt. Sowohl XForms als auch XML-Schema sind beide vom W3C entwickelt und standardisiert, und Formulare von XForms werden nach einem bestimmten Schema geschrieben. UIML verwendet eine DTD-Datei, die auch der Namensraum ist. Im Qt-Designer sind keine Deklarationen von Namensräumen, Schema oder DTD zu finden.

B3 Event-Handling

Event-Handling spielt eine wichtige Rolle bei der Verarbeitung von Benutzeroberflächen. Jeder Dialekt verwendet einen eigenen Mechanismus für Event-Handling.

In XUL kommt JavaScript zum Einsatz, weil JavaScript plattformunabhängig und Kompilierung nicht erforderlich ist. In XAML können alle Sprachen, die XAML unterstützen, verwendet werden, um die Events zu behandeln (z.B. C#, VB.NET).

XForms benutzt XML-Events, die eine Menge von Ereignissen sind und bereits vorher bei dem W3C definiert werden. Die XML-Events bieten XML die Möglichkeit an, Event-Listener uniform zu integrieren und Event-Listener mit dem Event-Handler, DOM (*Dokument Object Model*) Level 2 Event-Interfaces (*DOM2EVENTS*) [76], zu verbinden. In XML-Events werden viele allgemeine Ereignisse umfasst. Die restlichen Events werden mit JavaScript behandelt.

UIML wurde entwickelt, um alle Funktionalitäten von Benutzeroberflächen mit einer einzigen Sprache zu realisieren, deswegen werden die Events durch UIML selbst deklarativ behandelt. Qt verwendet ein besonderes Konzept „Signal/Slot“ zur Kommunikation zwischen Objekten. Das Konzept dient auch dazu, Events zu behandeln.

B4 Stylesheet

XUL benutzt CSS (ein Standard vom W3C), um den Präsentationsstil festzulegen. Viele Produkte von Microsoft verwenden immer eigene Modelle oder Spezifikationen. XAML ist auch keine Ausnahme. In XAML werden Attribute von Elementen eingestellt, um den Präsentationsstil der GUIs darzustellen. In XForms kommen CSS und XSL (*Extensible Stylesheet Language*) für die Präsentation zum Einsatz.

UIML hat ein Element <*style*>, in dem die Stile der Benutzeroberflächen eingestellt werden können. UIML 3.0 verwendet kein CSS, weil CSS ein bestimmtes Vokabular

benutzt, d.h. es ist schwierig zu erweitern. Außerdem kann UIML-Code in Code von einigen imperativen Programmiersprachen umgewandelt werden (z.B. in Java, C). CSS spielt aber keine Rolle für eine imperative Programmiersprache.

Der Qt-Designer benutzt eigene Elemente für die Stile der Benutzeroberflächen (z.B.: <font>, <pointsize>, <italic>…)

B5 (X)HTML

XHTML spielt eine wichtige Rolle bei Webseiten, deswegen wird XHTML als ein Kriterium betrachtet:

XUL und XHTML sind in einer Webseite kombinierbar. XAML versucht alle Funktionen von XHTML zu ersetzen. XForms selbst ist auf keinen Fall eigenständig. Die Formulare von XForms werden immer in einer Host-Datei definiert. Meistens wird XHTML als Host-Sprache verwendet. (X)HTML ist eine Zielsprache, in die UIML konvertiert werden kann. Dafür sind ein Vokabular und ein HTML-Renderer erforderlich. Der Qt-Designer hat keinen direkten Zusammenhang mit XHTML.

B6 UI-Elemente

XUL und XAML haben das gleiche Ziel, traditionelle Desktop- und Webapplikationen identisch zu behandeln. Sie liefern beide ausreichende GUI-Elemente. Als Nachfolger von HTML-Form hat XForms den Schwerpunkt Interaktionen für Webseiten. Die UI-Elemente dafür sind ausreichend. In UIML werden UI-Elemente abstrakt definiert und UIML verwendet für jede Zielsprache ein entsprechendes Vokabular. Der Stil und die Art von Elementen sind je nach Zielsprache festzulegen. Der Qt-Designer hat die GUI-Elemente, die die Qt-Bibliothek liefert.

B7 Grafik-Rendering

Eine wichtige Funktionalität von GUIs ist die graphische Darstellungsfähigkeit. Eine bekannte 2D-Vektorgrafik-Sprache ist SVG.

Ein XUL-Projekt XUL ist noch in Arbeit mit dem Ziel, SVG zu unterstützen. XAML benutzt eine SVG-ähnliche Technologie namens WVG (aber mit eigener Spezifikation von Microsoft). SVG ist eine Host-Sprache von XForms, d.h. XForms kann in eine SVG-Datei integriert werden. SVG ist auch ein XML-Dialekt. Mit Hilfe von UIML soll ein Vokabular von SVG entwickelt und ein Renderer für SVG angeboten werden können [61]. Der Qt-Designer besitzt zwei Module „3D OpenGL“ und „Canvas 2D“ aus den Qt-Bibliotheken für grafische Darstellung.

B8 Animation

Momentan hat XUL keine Animationsfähigkeit, während Animationen in XAML durch den Einsatz der entsprechenden „animation“-Attribute jedes Elements möglich sind. Dazu haben viele Elemente noch eigene Animations-Kollektionen.

W3C empfiehlt einen Standard für Multimedia namens SMIL (*Synchronized Multimedia Integration Language*), das Animationsfähigkeiten hat und auch ein XML-Dialekt ist. XForms und SMIL können beide in einer XHTML-Datei oder SVG-Datei integriert werden.

In der Dokumentation von UIML3.0 wird der Zusammenhang zwischen UIML und SMIL nicht deutlich gemacht [77]. UIML ist so entwickelt, dass es in eine beliebige Zielsprache umwandelbar ist. Theoretisch kann XSLT (*XSL Transformations*) für UIML und SMIL eingesetzt werden, damit die beiden ineinander konvertiert werden können.

B9: Web-Services: SOAP

Unterstützung von Web-Services, insbesondere SOAP (*Simple Object Access Protocol*), ist ein wichtiges Kriterium für XML-Anwendungen, wobei SOAP ein auf XML-basiertes Protokoll für Web-Services ist, ein einfacher Mechanismus zum Austausch strukturierter und typisierter Information zwischen den Rechnern in einer verteilten dezentralisierten Umgebung.

XUL unterstützt SOAP durch JavaScript. Momentan hat XForms keine Unterstützung von Web-Services (kein Standard) [78], aber viele Firmen und Organisation versuchen, XForms und Web-Services zu verbinden. Ein beispielhafter Versuch ist unter [79] zu finden.

UIML hat ein „call“-Element, das eine direkte Verbindung mit SOAP setzt [61]. Und Qt hat eine neue Qt-Komponente: „SOAP“. Durch diese hat Qt die Fähigkeit, mit SOAP zusammenzuarbeiten.

4.3 Eigenschaften der abgeleiteten Anwendungen.

	XUL	XAML	XForms	UIML	Qt-Designer
C1 Installation/ Verteilung	XPInstall von Mozilla	Click-Once	nicht erforderlich	nicht erforderlich	erforderlich
C2 Browser	Mozilla	IE in Longhorn (ohne Code)	Plugins für XForms, zukünftiger Browser	nicht erforderlich	nicht erforderlich
C3 Standalone	möglich	möglich	nein	ja	ja
C4 Ausführungs-umgebung	XRE und GRE	Longhorn SDK	keine	je nach der Zielsprache	keine

Tabelle 4-3: Eigenschaften der abgeleiteten Anwendungen

C1 Installation / Verteilung

Hier wird betrachtet, wie die von den Sprachen oder Programmen erstellten Anwendungen installiert werden können.

Bei XUL gibt es zwei Methoden, um die auf einem entfernten Server abgelegte XUL-Applikationen zuzugreifen bzw. diese zu verteilen. Die erste Methode ist, entfernte XUL-Dateien durch eine Webverknüpfung wie eine HTML-Seite lokal in Mozilla anzeigen zu lassen. Die zweite Methode ist, komprimierte XUL-Dateien mit einem Tool von Mozilla namens „XPInstall“ (ein entferntes Installationssystem) zunächst herunterzuladen und auf einem lokalen so genannten Chrome-Verzeichnis installieren zu lassen. Die komprimierte Datei enthält alle Dateien der XUL-Applikation und die für die Installation erforderliche Dateien. Ein Skript *install.js* ist zuständig für die Installation. XUL-Applikationen können nach der Installation standalone ausgeführt werden.

XAML verwendet eine Verteilungs-Technologie namens „Click-Once“. Sie soll mit der nächsten Version von Visual Studio „Whidbey“ auf den Markt kommen. Damit sollen sich Applikationen ganz einfach von einer Webseiten installieren lassen [80]**.** Sie hat viele Möglichkeiten für Verteilungen, wie z.B. Software unter „*Program Files*“ zu installieren, Versioning und „*Side-By-Side*“-Installation [49].

Um die in UIML geschriebene UIs einzusetzen ist normalerweise ein Konvertierungsverfahren erforderlich. Zur Darstellung von UIs spielt die ursprüngliche UIML-Datei dann keine Rolle mehr.

Der Qt-Designer ist eine GUI-Entwicklungsumgebung. Die in dem Qt-Designer erstellten Anwendungen sind eingepackte Dateien, die installiert werden müssen.

C2 Browser

Hier wird betrachtet, ob ein Browser für das Anzeigen der Anwendungen erforderlich ist.

Eine XUL-Datei kann einfach als eine Webseite im Browser von Mozilla angezeigt werden. Wenn eine XAML-Datei keinen Code enthält (d.h. weder Inline-Code noch Code-behind), kann sie in den IE von Longhorn eingeladen und angezeigt werden. Ein Browser mit Plugins für XForms (z.B. formsPlayer für IE6.0 [81]) kann die Dateien, die die Formulare von XForms enthalten, richtig anzeigen. Außerdem würde Mozilla in *Mozilla 1.8* und *Firefox 1.1* XForms unterstützt. Momentan verfügen die Beide nur über Betaversionen.

C3 Standalone

Hier wird betrachtet, ob eine abgeleitete Anwendung als eine traditionelle Desktop-Applikation behandelt werden kann.

XUL-Applikationen können ohne Browser als Desktop-Applikationen ausgeführt werden. Die Voraussetzung dafür ist, dass die XUL-Applikationen im Chrome-Verzeichnis registriert sind. Die Registrierung erfolgt durch das Werkzeug „XPInstall“ oder durch die Erstellung der Register-Datei per Hand, was jedoch schwieriger ist. XAML-Applikationen können standalone ausgeführt werden. Der Unterschied zwischen Webapplikationen und Desktopapplikationen ist, dass verschiedene Argumente an den Compiler beim Kompilieren gegeben werden müssen. XForms wird in den meisten Fällen nicht als eine Sprache für standalone Applikationen betrachtet.

C4 Ausführungsumgebung

Hier wird betrachtet, was für eine Umgebung zur Ausführung der erstellten Anwendungen erforderlich ist.

Wenn XUL-Dateien als Applikationen betracht werden, dann müssen XRE (*XUL Runtime Environment*) [82] oder/und GRE (*Gecko Runtime Environment*) [83] auf der Plattform installiert werden, damit die XUL-Dateien richtig interpretiert werden können.

Für XAML-Applikationen muss zuerst der Longhorn SDK (*Software Development Kit*) [84] installiert werden, dann können die XAML-Applikationen richtig ausgeführt werden. Außer dem Longhorn Windows-Systems wird der Longhorn SDK nur für XP und Windows Server 2003 geliefert.

Für die Qt-Applikationen ist keine Ausführungsumgebung erforderlich, daher existieren keine Konflikte bei den verschiedenen Versionen.

4.4 Eigenschaften der Dialekte bzgl. der Programmierung

	XUL	XAML	XForms	UIML	Qt-Designer
D1 Sprachen /Skripte	JavaScript	C#, VB.NET	JavaScript	theoretisch mit allen Sprachen	C++
D2 Code	innen /außen	innen / außen	außen	kein	innen / außen
D3 Kompilierung (Kom.) / Interpretation (Int.)	Int.	Komp. mit Code erforderlich	Int.	nicht erforderlich	Kom. mit Code erforderlich
D4 Kombination mit anderen Programmier sprachen	C++ (Durch XPConncet und XPCOM), Java	C#, VB.NET, JScript.NET	PHP, Servlets, ASP und ColdFusion	in andere Sprachen konvertierbar	Perl, Phyton, OpenGL, SQL
D5 Bibliotheken	XPCOM	Avalon-Framework	keine	keine	Qt
D6 Anbindung	XUL-Templete und RDF	Data Source	XPath	keine	C++
D7 Objekt-orientierung	nein	ja	ja	nein	ja
D8 Bedeutung eines Elements	nicht eindeutig	eine Klasse	abstrakt	sehr abstrakt	eine Klasse oder ein Attribut
D9 Eigene (Custom) Komponenten	durch XBL	Visual Tree	fremde Elemente	Vokabular	Custom-widgets oder Plugins

Tabelle 4-4: Eigenschaften der Dialekte bzgl. der Programmierung

D1: Sprachen /Skripte, D2: Code, D3: Kompilierung/Interpretation

Bei B3 wurde das Event-Handling bereits erwähnt. Bei D1 wird betrachtet, welche Sprachen bzw. Skripte für die Logik der Anwendungen verwendet werden können. Bei D2 wird betrachtet, wo der Code von der Logik zu finden ist. Bei D3 wird betrachtet, ob Kompilierung erforderlich ist.

XUL unterstützt ECMAScript [85], das die standardisierte Variante von JavaScript ist. JavaScript kann entweder mit XUL zusammen in eine Datei geschrieben werden

oder als eine externe Datei gespeichert und von XUL-Dateien aufgerufen werden. Denn JavaScript ist interpretierbar, daher ist Kompilierung nicht erforderlich.

In Longhorn wird dies durch so genannten „managed Code“ realisiert. In C# oder VB.NET kann managed Code geschrieben werden. Die Logik von XAML-Applikationen kann in C# oder VB.NET realisiert werden. Ähnlich wie XUL kann die Logik von XAML durch Inline-Code oder Code-behind realisiert werden. Falls XAML mit Code kombiniert wird, dann ist eine Kompilierung erforderlich. Dafür gibt es eine Build-Engine von DotNET namens *MSBuild.exe*.

Die wichtigste Anwendung von XForms ist die Webanwendung. Manchmal sind XML-Events bzw. Aktionen von XForms nicht in der Lage, eine besondere Business-Logik zu behandeln. In diesem Fall wird JavaScript eingesetzt. Aber XForms selbst enthält keine Elemente für Skripts, diese sind in einer Host-Sprache enthalten, z.B. in XHTML.

Die von UIML beschriebenen UIs können theoretisch in beliebige Sprache konvertiert werden, d.h. es ist möglich mit beliebigen Sprachen die Applikationslogik zu realisieren.

Für die vom Qt-Designer erzeugten .ui-Dateien, die keinen Code enthalten, sind zwei Möglichkeiten zu verwenden: Kompilierung oder dynamische Interpretation und Laden. Bei der dynamischen Interpretation und Laden darf kein Code enthalten sein; eigene Slots werden in diesem Fall durch eine Unterklasse von „QQbject“[68] verwirklicht.

D4: Kombination mit anderen Programmsprachen

Mozilla bietet eine Technologie namens XPCOM, durch die XUL mit C++ kombinierbar ist. Die Oberschicht von XPCOM ist XPConnect. XUL verwendet meistens JavaScript, um sich mit XPConnect zu verbinden (Andere Skriptsprachen kommen auch in Frage, z.B. Python). Dann kommuniziert XPConnect weiter mit XPCOM, und in XPCOM sind viele C++-Komponenten vorhanden. XPCOM bietet XUL die Möglichkeit an, externe APIs zu verwenden. Durch XPCOM kann XUL mit C++, Python, Ruby und Perl zusammenarbeiten. Außerdem gibt es einige Open Source Projekte, um XUL in die Java-Umgebung zu integrieren, z.B.: jXUL [86], Luxor XUL.

XAML unterstützt keine interpretierte Sprache wie z.B. JavaScript 3.0.

Für UIML werden einige Vokabulare erzeugt, durch die UIs in vielen Sprachen darstellbar sind. Z.B.: Java, C++ für QT-Widgets in Linux, C++ für eingebettetes System, C für PalmOS, VoiceXML, WML, HTML, CSS und Visual Basic [61].

D5: Bibliotheken

Durch XPConnect kann XUL verschiedene XPCOM-Bibliotheken von Mozilla verwenden. XAML benutzt das Avalon-Framework. Das Qt-Framework stellt dem Qt-Designer eine API mit über 400 Klassen zur Verfügung.

D6: Anbindung

An dieser Stelle wird die Fähigkeit für Datenzugriffe betrachtet, z.B. Anbindung mit XML, SQL usw. Traditionelle Daten-Anbindungen verbinden Daten und UI-Elemente miteinander. Die Daten liefern die Informationen und die UI-Elemente stellen die Informationen im richtigen Format dar. Die Daten-Anbindungen können in beide Richtungen erfolgen und können statisch oder dynamisch sein [49].

XUL kann durch den Einsatz von Skripten auf einigen Typen von Datenquellen zugreifen (z.B. einfache XML-Dateien, SOAP Web-Services usw.). Ohne Skripte kann XUL nur auf RDF-Dateien mit Hilfe von XUL-Templates zugreifen. Trotz dieser Beschränkung ist diese Methode schneller als die Verwendung von Skripten.

XAML hat gute Fähigkeiten für Daten-Anbindungen. Eine Eigenschaft des UI-Elements kann entweder mit der Eigenschaft eines CLR-Objektes (*Common Language Runtime*) oder mit einem Attribut von XML-Elementen verbunden werden. Die Datenquellen können eine Datenmenge von SQL-Query, eine XML-Datei oder eine eigene Datenquelle sein. Solche Anbindungen sind mit Auszeichnungen realisierbar.

Ein wichtiger Unterschied zwischen XForms und HTML-Form ist, dass die Daten von XForms-Formularen in XML definiert sind. Das W3C definierte eine Sprache namens XPath, um strukturierte Daten zu referenzieren. XForms verwendet XPath zur Daten-Anbindung. XAML benutzt auch XPath bei der Anbindung von XML. Außerdem kommt bei XForms auch XLinks zum Einsatz. Aber XLinks ist nicht so flexibel.

Im Qt-Designer werden Daten-Anbindungen durch C++ verwirklicht.

D7: Objektorientierung

XUL, JavaScript und CSS zusammen bilden eine Mozilla-Applikation. Diese drei sind interpretierte, keine objektorientierte Sprachen [87]. XAML und XForms haben eigene Schemas, die objektorientierte Beschreibung von XML ermöglichen. Außerdem kann XAML nach einer Kompilierung in eine objektorientierte Sprache

(z.B. C#) umgewandelt werden. UIML ist eine Meta-Sprache, sie kann in OO-Sprachen wie z.B. Java konvertiert werden.

D8: Bedeutung eines Elements

In XUL präsentiert ein Element ein Widget, ein Layout-Element oder ein Template usw. Ein Element in XAML ist einer DotNET-Klasse zugeordnet. Die Attribute von Elementen in XAML sind die Eigenschaften oder Events von Klassen. Elemente in XForms sind relative abstrakt definiert, währende Elemente in UIML sehr abstrakt definiert sind. Im Qt-Designer bildet ein Element entweder eine Klasse oder eine Eigenschaft einer Klasse ab.

D9: Eigene Komponenten

Hier wird betrachtet, ob die Dialekte dem Benutzer die Möglichkeit bieten, eigene Komponenten zu erstellen.

XUL ist nicht in der Lage, die Funktionalität von Elementen zu ändern. XBL (*eXtensible Bindings Language*) [88], die auch von Mozilla kreiert wird, übernimmt diese Aufgabe. XBL kann das Verhalten eines Widgets beschreiben. Eine XBL-Datei hat einige „binding“-Elemente, die Inhalte (*Content*), Eigenschaften (*Properties*), Methoden (*Methods*), Events und Style für Elemente deklarieren können. Durch den Einsatz von XBL können eigene Komponente für XUL spezifiziert oder personalisiert werden. Folgender Code ist ein Skelett einer XBL-Datei [89]:

```
<?xml version="1.0"?>
<bindings xmlns="http://www.mozilla.org/xbl">
  <binding id="binding1">
    <!-- content, property, method and event descriptions -->
  </binding>
  <binding id="binding2">
    <!-- content, property, method and event descriptions -->
  </binding>
</bindings>
```

In XAML gibt es auch einen ähnlichen Begriff wie XBL namens „visual tree“. Durch „visual tree“ können eigene Komponenten in XAML erzeugt werden. Hier wird ein Beispiel von „visual tree“ gegeben [90].

```
<Style def:Name="MyButtonStyle">
     <Button />
     <Style.VisualTree>
         <!-- inner element -->
     </Style.VisualTree>
  </Style>
```

In XForms können auch fremde Elemente verwendet werden. Das Problem ist, dass manche Renderer für XForms die fremden Elemente nicht richtig erkennen können.

Normalerweise werden in diesem Fall solche Elemente ignoriert. In manchen Fällen ist es aber wichtig, solche Elemente anzeigen zu lassen, z.B. eine elektronische Unter-schriftenformkontrolle. Dafür hat XForms ein *„mustUnderstand“*-Attribut eingesetzt, das ein boolescher Datentyp ist [78].

In UIML können durch den Einsatz von neuen Vokabularen eigene Komponenten erstellt werden.

Der Qt-Designer lässt sich durch Widget-Plugins („QWidgetPlugin“) um eigene Widgets erweitern. Alternativ kann ein eigenes Widget in einem „customwidget“-Element einer .ui-Datei enthalten sein. Die Verwendung von Plugins ist die neuere Methode.

4.5 Nicht-funktionale Anforderungen

	XUL	XAML	XForms	UIML	Qt-Designer
E1 Plattform-unabhängigkeit	ja	nein	ja	ja	ja
E2 Geräte-unabhängigkeit	ja	nein	ja	ja	ja
E3 Wartbarkeit	gut	mittel	gut	schlecht	gut
E4 Wieder-verwendbarkeit	gut	gut	sehr gut	möglich	gut
E5 Erweiterbarkeit	XPCOM und Overlays	Element-Komposition ,Objekt-orientierung	mehr Möglich-keiten	durch Umwandlung	gut
E6 Internationa-lisierung / Lokalisierung	DTD und .properties -Datei	unterstützt	unterstützt	unterstützt	C++

Tabelle 4-5: Nicht-funktionale Anforderungen

E1: Plattformunabhängigkeit

An dieser Stelle wird betrachtet, ob die von den Dialekten abgeleiteten Anwendungen von Plattformen bzw. Betriebssystemen abhängig sind.

XUL beschreibt die GUIs von Mozilla-Applikationen. XUL ist plattformneutral, d.h. ohne Änderung von XUL können die Applikationen auf allen Betriebssystemen, auf denen Mozilla installiert ist, ausführt werden, indem Mozilla als Interpretationsschicht zwischen der Applikation und dem Betriebssystem fungiert. Es folgt „write once, run anywhere“. Mozilla unterstützt z.B. Linux (Unix), Win32, und Mac [36].

XAML ist nur auf Windows-Betriebssystemen benutzbar.

XForms kommt zum Einsatz bei Webanwendungen und ist plattformunabhängig. Das Modell von XForms ist geeignet für verschiedene Benutzeroberflächen, z.B. XHTML oder unterschiedliche Architekturen von Servern, sogar VoiceXML.

Die ursprüngliche Motivation von UIML war es, eine plattform- und geräteunabhängige Metasprache zu entwickeln. Mit einem entsprechenden Renderer kann eine UIML-Datei unabhängig von Plattformen theoretisch in eine beliebige Sprache umwandelt werden.

Qt besitzt plattformunabhängige APIs. Die Quellcodes von Qt-Applikationen können einmal geschrieben werden und je nach Bedarf für jede Plattform kompiliert werden, und dann werden mit entsprechenden Bibliotheken der Plattform verknüpft (*link*). Qt-Applikationen können auf folgenden Plattformen laufen [91]:

- Qt/Windows: Microsoft Windows XP, 2000, NT 4, Me/98/95;
- Qt/X11: Linux, Solaris, HP-UX, IRIX, AIX, und viele Unix Varianten;
- Qt/Mac: Mac OS X
- Qt/Embedded: Embedded-Linux

Wobei Embedded-Linux für die Entwicklung auf eingebetteten Systemen verwendet wird, die unter Linux laufen und mit relativ wenig Speicher auskommen.

E2: Geräteunabhängigkeit

Durch die Trennung der Daten von Präsentationen ermöglicht XForms die Geräte-unabhängigkeit. Falls ein Datenmodell für alle Geräte zu verwenden ist, musste nur noch die Präsentation der Formulare an das entsprechenden Gerät angepasst werden. Daher kann XForms für Handys, PDAs, sogar für die Geräte für Blinde eingesetzt werden. Weiterhin ist XForms geräteunabhängig, dann können die Elemente von XForms direkt mit anderen XML-Dialekten, z.B. VoiceXML (*speaking web data*), WML, und SVG integriert werden.

UIML kann auf folgenden Geräten verwendet werden: PC, Handheld-PC, Palm, Handy, Voice-Phone und sogar noch nicht veröffentlichte Geräte [92].

Qt bzw. die eingebettete Bibliothek von Qt enthält ein komplettes Windows-System. Daher kann Qt einfach an jedes Displaygerät bzw. Inputgerät angepasst werden [93]. Qt läuft deshalb auf PDAs, Handys und Tablet PCs.

E3: Wartbarkeit *(Maintainability)*

Die vorgestellten XML-GUI-Dialekte sind deklarativ umformuliert. Der deklarative Teil und der imperative Teil einer Applikation sind getrennt behandelt. Im Vergleich zu dem imperativen Teil ist der deklarative Teil relativ einfach zu warten.

XUL kann externe JavaScript- oder CSS-Dateien aufrufen. Alle Funktionalitäten werden getrennt betracht, deswegen wird die Lesbarkeit verbessert, das bedeutet auch bessere Wartbarkeit.

Microsoft strebt an, die Funktionalitäten von XUL, PDF, CSS, SVG usw. alle in XAML zu integrieren. Trotz der Unterstützung der Technologie „Code-behind“ sieht eine XAML-Datei immer noch kompliziert aus. Im Gegensatz dazu behandelt XForms die Events durch XML-Events, daher wird die Einführung von den relativ komplexen Skripts vermieden. Das führt zur besseren Wartbarkeit.

Die Elemente von UIML sind abstrakt definiert, und für jede Zielsprache muss ein entsprechendes Vokabular eingesetzt werden. Die Informationen zu möglichst vielen Zielsprachen werden berücksichtigt und zusammengeschrieben. Das bedeutet, dass UIML-Dateien schwer zu lesen und warten sind.

Eine vom Qt-Designer erzeugte .ui-Datei (im XML-Format) kann zu jeder Zeit von dem Qt-Designer eingelesen und daraus die entsprechenden GUIs erstellt werden. Durch „drag and drop“ können die GUIs anschließend einfach bearbeitet werden. Es ist nur erforderlich, die .ui-Datei noch mal in C++-Code umzuwandern und zu kompilieren. Die Voraussetzung ist, dass der Qt-Designer vorhanden ist.

E4: Wiederverwendbarkeit (*Reusability*)
Hier wird betrachtet, in wie weit die von XML-GUI-Dialekten erzeugten Codes wieder verwendet werden können.

XUL hat einen Mechanismus namens „Overlays“, durch den es möglich ist, die in XUL geschriebenen Komponenten wieder zu verwenden. Ein Overlay ist eine separate Datei, die grundsätzlich auch eine XUL-Datei ist und zur Laufzeit eingebunden wird. Ein weiterer Grund ist die Modularisierung der Codebasis, was einer erhöhten Wiederverwendbarkeit gleichkommt [36]. Typischerweise werden Overlays verwendet, wenn ein Teil einer XUL-Datei mehrmals in einigen Applikationen aufgetaucht, z.B. eine Menüleiste.

XAML erhöht ihre Wiederverwendbarkeit durch das Konzept „Element- Komposition“. Außerdem ist XAML eine objektorientierte Sprache, und es ist bekannt, dass objektorientierte Programmierung eine bessere Wiederverwendbarkeit besitzt.

XForms erhöht ihre Wiederverwendbarkeit durch das Konzept der Trennung der Daten und der Logik von Präsentationen. Außerdem hat die geringe Benutzung von Skripten auch diese Eigenschaft verstärket.

UIML hat ein Element „template“, in dem ein Block eines UIML-Codes schon vordefiniert ist. Dieser Block kann als eine Komponente von anderen UIML-Dateien wieder verwendet werden [61].

Da Qt modular aufgebaut und in C++ geschrieben ist, besitzt es gute Wiederverwendbarkeit gemäß des objektorientierten Ansatzes. Weiterhin bietet Qt die Unterstützung von dynamischen Bibliotheken und Plugins, die ebenso die Wiederverwendbarkeit verbessert.

E5: Erweiterbarkeit (Extensiblility)

In D9 wurde das Kriterium „Eigene Komponente“ bereits diskutiert. Es gehört zur Erweiterbarkeit eines Dialekts. Hier gibt es noch mehr dazu:

Einige Methoden können eingesetzt werden, um XUL zu erweitern. Z.B. kann ein neues Objekt hinzugefügt werden, sogar ein XPCOM-Objekt. Oder ein neues XML-Element und ein neues Thema können ergänzt werden. Außerdem ist Mozilla Open Source, daher ist es möglich, Mozilla zu personalisieren [40]. Weiterhin können durch den Einsatz von „Overlays” fremde Komponente benutzt werden. Ein Produkt von Mozilla namens „Firefox“ bietet z.B. fast hundert Extensionen an.

Das Konzept „Element-Komposition“ bietet einen zentralen Mechanismus zur Erweiterung von XAML [94].

XForms bietet viele Möglichkeiten zur Erweiterung: mit Skripten, mit neuen Datentypen und Bibliotheken, mit Extensionsfunktions in Xpath, mit neuen Steuerelementen, mit XForms-Aktionen usw. [78]. Außerdem ist XForms keine standalone Datei, es ist nur ein Block, der in andere Host-Sprachen eingebettet werden muss, z.B. es gibt bereits die Kombinationen von XForms und PHP, Servlets, ASP oder ColdFusion [95].

UIML ist theoretisch in eine beliebige Sprache umwandelbar. Dies gehört auch zur Erweiterbarkeit. UIML kann sogar für die noch nicht veröffentlichte Geräte verwenden lassen.

E6: Internationalisierung /Lokalisierung

Alle vorgestellten XML-GUI-Dialekte (inklusive Qt) unterstützen Unicode.

XUL, XAML und XForms verwenden ein ähnliches Konzept, um Daten bzw. Stile zu lokalisieren, z.B. sind die angezeigten Namen von Elementen, Labels, Hilfe-, Hinweis-, und Warnungsnachrichten abhängig von Sprachen. Solche Texte werden nicht direkt im XUL-, XAML-, oder XForms-Code angegeben, sondern in einer

externen (XML-)Datei abgelegt. Der Zugriff ist durch eine Referenz des Elementsnamen des Texts möglich. XUL z.B. verwendet DTD- und .properties-Dateien, um lokalisierte Informationen abzulegen.

UIML hat die Beschreibung eines UIs in sechs Blöcke aufgeteilt. Im Block „*content*“ können Text, Images usw. enthalten sein. Deshalb können UIs lokalisiert werden. Weiterhin kann die Lokalisierung von Stilen durch die Einstellungen im Block „*style*“ realisiert werden.

Die Qt-Applikationen haben die Fähigkeit, in der Laufzeit Sprachen zu wechseln.

4.6 Handhabung und Entwicklungsprozess

	XUL	XAML	XForms	UIML	Qt-Designer
F1 Funktionalität	Siehe Bemerkungen				
F2 Tools					
F3 Entwicklungszeit	lang	kurz mit Visual Studio	lang	lang	kurz
F4 Dokumentation	gut	mittel	gut	veraltet	gut

Tabelle 4-6: Handhabung und Entwicklungsprozess

F1: Funktionalität

Einer der wichtigen Anwendungsschwerpunkte von XUL sind Webapplikationen. XUL ist deshalb geeignet für Webanwendungen. Durch XUL können so genannte „Rich Internet Clients“ realisiert werden. Die auf der Server-Seite abgelegte Webapplikationen können einfach durch einen Browser auf der Client-Seite verteilt werden. XUL hat ausreichende GUI-Elemente und kann anhand XPCOM über tausend Mozilla-Componenten verwenden.

XAML strebt an, die folgten Funktionen zu integrieren:

- Präsentation der GUI
- Dateiservice wie PDF
- Vektordarstellung wie SVG
- Stylesheet wie CSS
- Animation wie Flash.

XForms bietet genügend UI-Elemente an. Durch den Einsatz von XML-Schema ermöglicht XForms Datentypen von eingegebenen Daten lokal zu überprüfen, was den Komfort für den Benutzer erhöht. XForms speichert und transportiert Daten in das XML-Format, das ist natürlich faszinierend.

UIML ist einzusetzen, wenn Applikationen nicht nur für eine einzige Art von Geräten oder Plattformen entwickelt werden sollen.

Qt stellt eine API mit über 400 C++-Klassen zur Verfügung, die weit über bloße GUI-Programmierung hinausgehen: template-basierte Collections, Serialisierung, Netzwerke, XML, Datenbanken, Internationalisierung etc.

F2: Tools

Außer dem Qt-Designer gibt es momentan noch keine ausgereiften IDE-Toolkits für die anderen vorgestellten Dialekte. Hier werden einige Browser oder Player bzw. Renderer bekannt gegeben.

Einige Produkte von Mozilla wurden entwickelt unter der Voraussetzung, dass XUL als Präsentationssprache verwendet werden sollte, z.B.

- Firefox 1.0, ein leichter Webbrowser,
- Thunderbird 1.0, ein neue E-Mail-Client von Mozilla
- Mozilla Suite [96], ein komplettes Set mit einem Webbrowser, einem Client für E-Mail und Newsgroup, einem Client für IRC (Internet Relay Chatting), und einem HTML-Editor erhält.

Außerdem gibt es noch andere Ansätze, um XUL in eine andere Entwicklungsumgebung zu bringen. z.B. Blackwood Project [97] von Mozilla, Luxor, jXUL und XULUX [98], die Java als Programmierumgebung verwenden. Außerdem ist „Embed Gecko XUL“ [99] auf einigen Geräten wie Handy, PDA einsatzbar.

Vor dem Erscheinen von Visual Studio 2005, „Whidbey“ haben sich bereits einige Firmen mit XAML-Tools beschäftigt wie z.B.: Xamlon [100], MobiForm [101]

Tools zur Darstellung von XForms sind u.a.:

- Chiba [102]
- SVGViewPlus von MobiForm [103]
- Mozquito DENG [104]
- X-Smiles von Universität Helsinki [105]
- FormsPlayer von x-port

Die Firma Harmonia hat einige Renderer für UIML entwickelt. Es gibt ein Produkt-Paket LiquidUI, das UIML Server, Java Renderer, HTML Renderer, WML Renderer, VoiceXML Renderer und PalmOS Renderer enthält, wobei der PalmOS Renderer UIML in C (mit API für PalmOS) konvertiert. Die Firma VirginiaTech beschäftigt sich auch mit Renderern für UIML.

Außer dem Qt-Designer bietet Qt auch andere Werkzeuge oder „command line“-Tools, z.B. *Qt-Linguist, Qt-Assistant und moc*, ein Compiler für Meta-Objekte, und *uic*, ein Compiler für Benutzeroberflächen.

4.7 Ein GUI-Beispiel in den fünf Dialekten

Hier wird ein Beispiel in den fünf vorgestellten Dialekten geschrieben. In diesem Beispiel wird eine Benutzeroberfläche für eine Namenseingabe erstellt. Darunter werden die Screenshots der GUIs in den entsprechenden Dialekten angezeigt. Alle Quellcodes sind im Anhang zu finden.

XUL:

Eine XUL-Datei kann entweder direkt in Mozilla eingeladen werden, wie Abbildung 4-1 zeigt, oder nachdem sie im Chrome-Verzeichnis registriert wurde, als eine standalone Applikation ausgeführt werden, wie es auf Abbildung 4-2 zu sehen ist. Der Befehl zur Ausführung war

```
"C:\Program Files\mozilla.org\Mozilla\mozilla.exe" -chrome
chrome://meinbeispiel/content/
```

Wobei die XUL-Datei mit dem Name „*meinbeispiel.xul*“ schon im Chrome-Verzeichnis abgelegt war.

Abbildung 4-2: XUL-Datei als Webseite in Mozilla

Abbildung 4-1: GUI in XUL im standalone Mode

XUL unterstützt das so genannte „native Look & Feel“, d.h. die Darstellung von GUIs kann von Betriebssystemen spezifiziert werden.

XAML:

Momentan gibt es Visual Studio 2005, *Whidbey* nur als Demo-Version, daher wird hier eine XAML-Datei durch Xamlon, eine Software für XAML, in Abbildung 4-3 angezeigt. Nach der Erscheinung von *Whidbey* soll die Entwicklung von GUIs in

XAML vereinfacht werden, sonst ist es sehr schwierig, per Hand GUIs in XAML zu erstellen.

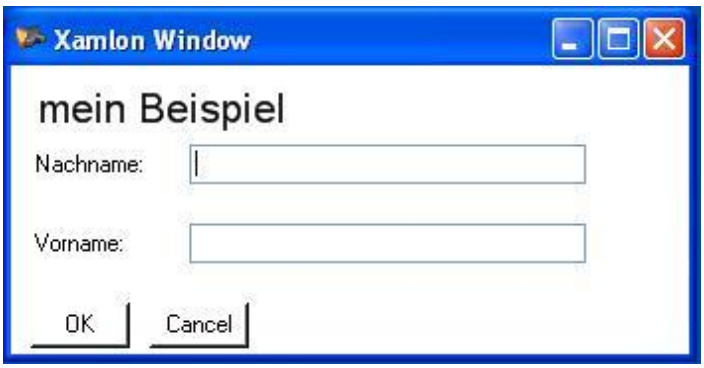

Abbildung 4-3: GUI in XAML angezeigt in dem Xamlon-Browser.

XForms:

Hier wird eine XHTML-Datei als Host-Sprache für XForms eingesetzt. Die Bearbeitung und Darstellung von XForms sind durch ein Plugin für IE6.0 namens „Formsplayer“ durchführbar. Der Screenshot davon wird in Abbildung 4-4 dargestellt.

Abbildung 4-4: XForms in IE6.0 mit dem Plugin „Formsplayer“

UIML:

Hier wird eine UIML-Datei mit Unterstützung von LiquidUI [106] in Java (Abbildung 4-5) und in eine HTML-Datei (Abbildung 4-6) konvertiert.

Abbildung 4-5: Umwandlung von UIML in JAVA-AWT

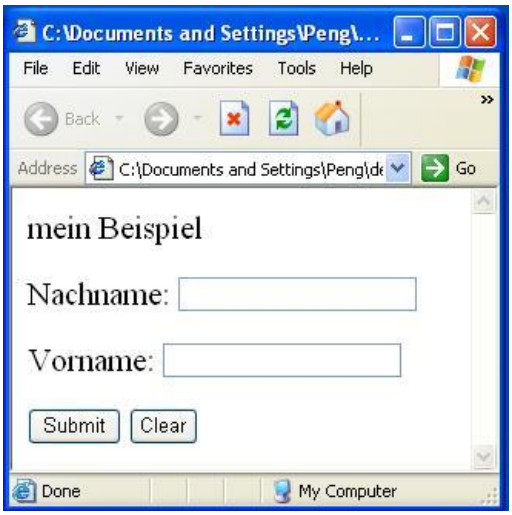

Abbildung 4-6: Umwandlung von UIML in HTML

Qt-Designer:

Der Qt-Designer unterstützt auch „natives Look & Feel“, die folgte Abbildung 4-7 ist auf Linux erstellt.

Abbildung 4-7: GUI im Qt-Designer

4.8 Bewertung

Vor der Bewertung aller Dialekte wird noch ein Punkt hier kurz betrachtet: ob es mit den Dialekten möglich ist, eine Oberfläche für einen Client zu beschreiben, der komplett von einem Server gesteuert wird.

XUL hat eine Installationsstrategie, XPInstall, durch das eine in XUL geschriebene Oberfläche von einem Server gesteuert wird. Aber die Durchführung von XPInstall ist umständlich.

XAML hat eine ähnliche Installationsstrategie wie XUL, Click-Once. Microsoft behauptet, dass Click-Once mächtig und bequem ist. Aber bis jetzt wurden noch keine reifen Produkte auf den Markt gebracht.

XForms wird meistens in XHTML integriert und auf einem Server abgelegt, d.h. XForms kann einfach von einem Server gesteuert werden. Außerdem gibt es einen Prozessor „Chiba“, das XForms von der Serverseite steuern kann: Die Serverseite übernimmt die Interpretation vom Formular aus XForms. Die Steuerung anhand des Prozessors kostet viel Zeit und braucht großes Volumen zur Datenübertragung.

Die Firma Harmonia hat die UIML-Toolkits „*LiquidUI*“ entwickelt, die ein UIML-Server zur Verfügung stellt. Der UIML-Server erzeugt aus einer UIML-Datei die UIs für den Webbrowser oder Handy. Falls der UIML-Server von einem Client eine Nachfrage in Form einer URL bekommt, wird er ein UI nach der entsprechenden Nachfrage erzeugen und zum Client zurückschicken, indem er entsprechende Renderer von *LiquidUI* aufruft.

Der Qt Designer ist geeignet für Desktopapplikationen. Bei der Entwicklung wurde die Kommunikation zwischen Servern und Clients wenig betrachtet.

An dieser Stelle wird das geeignete Anwendungsbereich von jedem vorgestellten Dialekt kurz bewertet.

XUL und ihre verwandte Technologien sind geeignet für Webanwendungen. Insbesondere hat Mozilla über tausend XPCOM-Komponenten geliefert, die meistens für das Web und Netz eingesetzt sind. Aber bei XUL fehlt es noch an einer guten Entwicklungsumgebung bzw. guten RAD-Tools.

XAML und ihre Technologien streben an, ein neues Programmiermodell zu geben. Sie sind in der Lage, eine Anwendung mit der Webfähigkeit und der Desktopfähigkeit zu entwickeln. XAML ist plattformenabhängig, weil sie vom Avalon-Framework (oder

DotNET-Framework 1.1) abhängig ist. Außerdem sind die Programmiersprachen für die Applikationslogik, die mit XAML zusammenarbeitet sind, auch plattformabhängig.

XForms ist zur Sammlung und Verwaltung von Informationen in XML-Format zuständig. XForms kann insbesondere die Daten von Benutzern im Internet besser sammeln. Außerdem hat XForms die starke lokale Datentypenüberprüfungs-fähigkeit, durch die XForms in der Lage ist, bessere und bequemere Kommunikation zwischen Rechnern und Menschen zu realisieren. XForms, SVG, SMIL sind Standards vom W3C. Durch die Kombination der drei Dialekte sind Entwicklungen von Graphiken und Multimediaanwendungen fürs Web einfach und effizient.

Trotz der Plattform- und Geräteunabhängigkeit ist UIML relative unbequem zu verwenden. Das Entwicklungsziel von UIML war, Codes für Benutzeroberflächen nur in einer einzigen Sprache einmalig zu schreiben (Den Rest erledigt die Sprache selbst, z.B. Kompilierung und Übersetzung des Codes für ein bestimmtes Gerät). In der Tat müssen jedoch einige Kenntnisse von allen getroffenen Zielsprachen beherrscht werden, damit die entsprechenden UIs erstellt werden können.

Qt ist ein reifes GUI-Framework, trotzdem ist es nur eignet für erfahrende C++-Programmierer. Die vom Qt-Designer erzeugte .ui-Datei spielt wenige Rolle für Benutzer, Sie dient nur zur Verbesserung der Fähigkeit des RAD-Tools. Benutzer bemerken sie in den meisten Fällen nicht.

5 Zusammenfassung

XML ist eine wichtige Technologie für das Internet und das World Wide Web der Zukunft. Seit Jahren wird XML wegen seiner Strukturierung oder anderen Eigenschaften in vielen Bereichen eingesetzt. Nach der Einführung von GUI-basierten Systemen und Applikationen ist die Computerwelt einfacher und schöner geworden. Wegen der Vielfalt von Plattformen und Geräten sind dann XML-basierte GUIs entstanden. In dieser Arbeit wurden fünf XML-basierte GUI-Standards als Schwerpunkte betrachtet.

Viele XML-basierte GUIs sind immer noch in Entwicklungs- bzw. Verbesserungsphase. Manche Nachteile von solchen GUIs sollen behoben werden, z.B. Standardisierung und unkomfortable Bedienung. Bei der Entwicklung der verschiedenen Dialekte wurden verschiedene Ziele gesetzt. Daher ist normalerweise ein Dialekt nur für ein oder einige Zwecke geeignet. XUL und ihre verwandte Technologien sind geeignet für Webanwendungen. XAML sind in der Lage, eine Anwendung mit der Webfähigkeit und der Desktopfähigkeit zu entwickeln. XForms ist zur Sammlung und Verwaltung von Informationen in XML-Format zuständig. Trotz der Plattform- und Geräteunabhängigkeit ist UIML relative unbequem zu verwenden. Qt ist ein reifes GUI-Framework, trotzdem ist es nur eignet für erfahrende C++-Programmierer.

Für unterschiedliche Dialekte sind unterschiedliche Tools für das Anzeigen, die Entwicklung und die Weiterbearbeitung zu haben. Es ist ein Entscheidungsproblem, welcher Dialekt in Frage und zum Einsatz kommt, um eine bestimmte Aufgabe zu lösen

Es ist aber eine Tendenz erkennbar, dass XML-basierte GUIs in der Zukunft in vielen Bereichen zu finden sind. Insbesondere haben einige Webprogrammier- bzw. Skriptensprache (z.B. PHP, ASP, JSP, Perl, …) neue Versionen, die eine Volle Unterstützung von XML-Technologien haben, auf den Markt gebracht.

Anhang

A Literatur

1. W3C, 2005, www.w3c.org
2. Erik T. Ray, ***Einführung in XML***, 2001, 1. Auflage, O'Reilly
3. W3C, ***XML in 10 Punkten***, 2001, http://www.w3.org/XML/1999/XML-in-10-points
4. W3C, SMIL2.0, 2005, http://www.w3.org/AudioVideo/
5. W3C, SVG1.1, 2003, http://www.w3.org/Graphics/SVG/
6. W3C, (X)HTML2.0, 2004, http://www.w3.org/MarkUp/
7. W3C, XSL,2005, http://www.w3.org/Style/XSL/
8. W3C, CSS level 2, 2004, http://www.w3.org/Style/CSS/
9. W3C, MathML2.0, 2001, http://www.w3.org/Math/
10. W3C, XFORMS1.1, 2004, http://www.w3.org/MarkUp/Forms/
11. W3C, CC/PP, 2004, http://www.w3.org/Mobile/CCPP/
12. W3C, VoiceXML2.0, 2004, http://www.w3.org/Voice/
13. W3C, P3P, 2004, http://www.w3.org/P3P/
14. W3C, XML-Signature, 2002, http://www.w3.org/Signature/
15. W3C, XML-Encryption, 2002, http://www.w3.org/Encryption/2001/
16. W3C, XML Key Management (XKMS 2.0), 2004, http://www.w3.org/2001/XKMS/
17. W3C, SOAP1.2 2003, http://www.w3.org/2000/xp/Group/
18. W3C, WSDL2.0, 2004, http://www.w3.org/2002/ws/
19. W3C, XQuery1.0, 2004, http://www.w3.org/XML/Query
20. W3C, XPath1.0, 1999, http://www.w3.org/TR/xpath
21. W3C, Semantic Web, 2005, http://www.w3.org/2001/sw/
22. W3C, RDF, 2004, http://www.w3.org/RDF/
23. W3C, OWL, 2004, http://www.w3.org/2001/sw/WebOnt/
24. W3C, XML1.1, 2004, http://www.w3.org/XML/
25. W3C, Namensräume, 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114/
26. W3C, XSLT: http://www.w3.org/Style/XSL/
27. W3C, XLink1.0, 2001, http://www.w3.org/XML/Linking
28. W3C, XPOINTER, 2002, http://www.w3.org/TR/xptr/
29. W3C, XML BASE,2001, http://www.w3.org/TR/xmlbase/
30. W3C, XML Schema, 2004, http://www.w3.org/XML/Schema
31. W3C, DOM, 2004, http://www.w3.org/DOM/
32. Klaus Birkenbihl, *XML-Architektur*, 2003 http://www.w3c.de/PubPraes/XML%20-%20Architektur%20Tutorial%20-%20gen.html
33. Workshop on Developing User Interfaces with XML**,** Advances on User Interface Description Languages, 2004, http://www.edm.luc.ac.be/uixml2004/index.php?selected=cfp:
34. xulplanet, XUL, 1999-2005, http://www.xulplanet.com/ ; Mozilla, xul, 2001, http://www.mozilla.org/projects/xul/
35. Mozilla, XPToolkit, 1999, http://www.mozilla.org/xpfe/
36. David Boswell, Brian King, Ian Oeschger, Pete Collins & Eric Murphy, ***Creating Applications with Mozilla***, 2002, O'Reilly, 1. Auflage
37. Mozilla, XPFE, 1999, http://www.mozilla.org/xpfe/
38. Mozilla, Firefox, 2005, http://www.mozilla.org/products/firefox/
39. Mozilla, Thunderbird, 2005, http://www.mozilla.org/products/thunderbird
40. Nigel McFarlane, ***Rapid application development with Mozilla***, 2004, Pearson Education
41. Mozilla, XPInstall,2003, http://www.mozilla.org/projects/xpinstall/

42. Mozilla, XPCOM, 2003, http://www.mozilla.org/projects/xpcom/
43. Mozilla, XPConnect, 2000, http://www.mozilla.org/scriptable/
44. Luxor, ***XML UI Language (XUL) Toolkit***, 2005, http://luxor-xul.sourceforge.net/
45. Microsoft, Longhorn, 2005, *http://msdn.microsoft.com/longhorn/*
46. Microsoft, xaml, *http://longhorn.msdn.microsoft.com/?//longhorn.msdn.microsoft.com/lhsdk/core/overviews/about%20xaml.aspx*
47. Aaron Skonnard**, *XML Report from the Microsoft PDC 2003***, 2004 http://msdn.microsoft.com/msdnmag/issues/04/02/XMLFiles/default.aspx
48. Karsten Januszewski, Tim Sneath, and Arik Cohen, Microsoft Corporation November 2004 ***Avalon November 2004 Community Technology Preview***, 2004 http://msdn.microsoft.com/Longhorn/understanding/pillars/avalon/avnov04ctp/default.aspx
49. Brent Rector, ***Introducing Longhorn for Developers***, 10. 2003 http://msdn.microsoft.com/Longhorn/understanding/books/rector/default.aspx
50. Ian Griffiths, co-author of Mastering Visual Studio .NET, ***Inside XAML***, 01.19.2004 http://www.ondotnet.com/pub/a/dotnet/2004/01/19/longhorn.html
51. Dino Esposito, ***A First Look at Writing and Deploying Apps in the Next Generation of Windows,*** January 2004, http://msdn.microsoft.com/msdnmag/issues/04/01/DevelopingAppsforLonghorn/
52. Microsoft, Whidbey, 2005, http://msdn.microsoft.com/vstudio/whidbey/default.aspx
53. W3C Forms Working Group, ***XForms 1.0 Frequently Asked Questions,*** 2003 http://www.w3.org/MarkUp/Forms/2003/xforms-faq.html
54. W3C, XML-Events, 2003 http://www.w3.org/TR/2003/REC-xml-events-20031014/
55. Micah Dubinko ***What Are XForms?*****,** 11. 9. 2002, http://www.xml.com/pub/a/2001/09/05/xforms.html
56. Joel Rivera, Len Taing, ***Get ready for XForms,*** 01. 09. 2002 http://www-106.ibm.com/developerworks/xml/library/x-xforms/?Open&ca=daw-xml-dr
57. W3C, ***XForms - The Next Generation of Web Forms,*** 2005, http://www.w3.org/MarkUp/Forms/
58. uiml.org, uiml, 1999-2004, *http://www.uiml.org/*
59. Harmonia, Inc, harmonia, 1997-2005 *http://www.harmonia.com/*
60. Openmobilealliance, wml, 2001 *http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html*
61. uiml.org, ***Draft specification for language version 3.0***, 12. 02. 2002 http://www.uiml.org/specs/docs/uiml30-revised-02-12-02.pdf
62. Prof. Dr.-Ing. Holger Vogelsang, ***Benutzeroberflächen Deklarative Beschreibung***, 14.10.2004 http://www.fbi-lkt.fh-karlsruhe.de/~voho0001/Benutzeroberflachen/Vorlesung/FB14-Deklarative_Beschreibung.pdf
63. kde.org, KDE, 2005, www.kde.org
64. gtk,org, Gtk, 2005, www.gtk.org/
65. Motif, 2005, www.opengroup.org/motif/
66. Anne-Marie Mahfouf , German Translation : Andreas Nicolai, 2003, ***Qt Designer und KDevelop-3.0 für Beginner,*** http://women.kde.org/articles/tutorials/kdevelop3/de/
67. Prof.Dr. Grisbert Dittrich, ***Ein "richtiges" Programm***, 2004 http://mediasrv.cs.uni-dortmund.de/Lehre/WS2003_04/EINI_WS2003_04/ppt/EINI%20Kap22.ppt
68. Trolltech AS, ***Qt Reference Documentation,*** 2004, http://doc.trolltech.com/3.3/index.html
69. eNode, Inc, eNode, 2002, http://www.enode.com/
70. XUI, 2005, http://xui.sourceforge.net/
71. IBM, AUIML, 2004, www.alphaworks.ibm.com/tech/auiml
72. Gnome.org, Glade2.6.8, 2004, http://glade.gnome.org/
73. Macromedia, Flex und MXML, 2005, http://www.macromedia.com/devnet/flex/
74. myxaml, 2004, www.myxaml.com/
75. OASIS, 1993-2005, www.oasis-open.org/

76. W3C, DOM2EVENTS, 2000, http://www.w3.org/TR/DOM-Level-2-Events/
77. OASIS, **The Relationship of the UIML 3.1 Spec. to Other Standards/Working Groups**, 2004 http://www.oasis-open.org/committees/download.php/8256/The%20Relationship%20of%20the%20UIML%203%20v01.04.doc
78. Micah Dubinko, ***XForms Essentials***, 2003, O'Reilly
79. Nicholas Chase, ***Send-part-of-an-XForms-instance-to-a-Web-service***, 13. 08. 2004 http://www-106.ibm.com/developerworks/xml/library/x-tipxf3/
80. ***Avalon zum Anschauen,*** 17. 01 2005, http://www.wcm.at/story.php?id=7548
81. x-port.net, FormsPlayer1.3, 2005, www.formsplayer.com/
82. Mozilla, XRE,2003, www.mozilla.org/projects/xul/xre.html
83. Mozilla ,GRE: Chak Nanga, Doug Turner, 2005, http://www.mozilla.org/projects/embedding/GRE.html
84. Microsoft, Longhorn SDK, 2005, http://longhorn.msdn.microsoft.com/
85. Ecma International, ECMAScript, 1999, http://www.ecma-international.org/publications/standards/Ecma-262.htm
86. jXUL project, ,jXUL, 2001, http://jxul.sourceforge.net
87. Nigel McFarlane, ***Create Web applets with Mozilla and XML***, 21. 10. 2003 http://www-106.ibm.com/developerworks/web/library/wa-appmozx/
88. Mozilla, XBL, 2003, www.mozilla.org/projects/xbl/xbl.html
89. Neil Deakin, XUL Tutorial, 2004, http://www.xulplanet.com/tutorials/xultu/
90. Nathan Dunlap, ***Creating a customized GelButton*****,** 21. 01. 2004 http://www.longhornblogs.com/ndunlap/articles/2203.aspx
91. Trolltech AS, Qt Overview, 2005, http://www.trolltech.com/products/qt/
92. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen Williams, Jonathan E. Shuster Harmonia, Inc., ***UIML: An Appliance-Independent XML User Interface Language,*** 1999 http://www.harmonia.com/resources/papers/www8_0599/index.htm
93. Trolltech AS, Qt/Embedded, 2001 http://mirror.aarnet.edu.au/pub/qt/pdf/QtEWhitepaper.pdf
94. Jeff Bogdan Microsoft Corporation, ***Timing Is Everything***, 13. 01. 2004 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnavalon/html/avalon01062004.asp
95. Alex Sansom, ***Using XForms With PHP, Servlets, ASP and ColdFusion,*** 2005 http://www.formsplayer.com/community/howto/howto-server-scripting.html
96. Mozilla, Mozilla Suite, 2004, http://www.mozilla.org/products/mozilla1.x/
97. Mozilla, Blackwood Project, 2003, http://www.mozilla.org/projects/blackwood
98. xulux.org, xulux, 2002-2004, http://www.xulux.org/
99. Mozilla, Embed Gecko XUL, 2003, http://www.mozilla.org/projects/embedding/embedoverview/EmbeddingGecko.pdf
100. Xamlon Inc, Xamlon, 2004, www.xamlon.com/
101. Mobiform Software Ltd. ,Mobiform, 2005, www.mobiform.com
102. Chiba Projekt, Chiba, 2004, http://chiba.sourceforge.net/
103. Mobiform Software Ltd., SVGViewPlus, 2005, www.mobiform.com
104. Claus Wahlers, Dani Wahlers, DENG Projekt, 2004, http://claus.packts.net/
105. X-Smiles.org, X-Smiles von Universität Helsinki, 2000-2004, www.xsmiles.org
106. Hamonia Inc., LiquidUI, 2005, www.harmonia.com/products/LiquidUI/
107. Mark Birbeck, ***Authoring a Simple XForms Form***, 2005, http://www.formsplayer.com/community/howto/howto-hello-world.html

Die alle gegebenen Verknüpfungen werden zuletzt am 07. Feb. 2005 besucht.

B Abkürzungsverzeichnis

AUIML	Abstract User Interface Markup Language
CLR	Common Language Runtime
CSS	Cascading Style Sheets
DOM	Document Object Model
DTD	Data Type Definition
GRE	Gecko Runtime Environment
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IRC	Internet Relay Chatting
JNLP	Java Network Launch Protocol
KDE	K Desktop Environment
MFC	Microsoft Foundation Classes
OASIS	Organization for the Advancement of Structured Information Standards
PDA	Personal Digital Assistant
PDF	Portable Document Format
RAD	Rapid Application Development
RIA	Rich Internet Applications
RDF	Resource Description Framework
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
SOAP	Simple Object Access Protocol
SVG	Scalable Vector Graphics
UIDL	User Interface Description Language
URI	Uniform Resource Indicator
W3C	World Wide Web Consortium
WinFX	Windows Framework Extensions
WVG	Windows Vector Graphics
WML	Wireless Markup Language
XAML	Extensible Application Markup Language
XBL	Extensible Bindings Language
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XPCOM	Cross-platform Component Object Model
XPFE	Cross Platform Front End
XPInstall	Cross Platform Install

XPToolkit	Cross Platform Toolkit
XRE	XUL Runtime Environment
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations
XUI	Extensible User Interface
XUL	XML User Interface Language
UIML	User Interface Markup Language

C Tabellenverzeichnis

D Abbildungsverzeichnis

E Quellcode

Quellcode von den Beispielen im Abschnitt 4.7.

XUL: *meinbeispiel.xul*

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin" type="text/css"?>
<!DOCTYPE window>
<window title="Beispiele!"
  xmlns:html="http://www.w3.org/1999/xhtml"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  style="background-color: white;"
  width="250"
  height="150"
  onload="centerWindowOnScreen( )">

<script type="application/x-javascript"
  src="chrome://global/content/dialogOverlay.js" />

<vbox>
  <hbox align="left">
      <label control="InputNachname" value="Nachname"/>
        <spacer flex="1"/>
      <textbox id="InputNachname"/>
  </hbox>
  <hbox align="left">
      <label control="InputVorname" value="Vorname"/>
        <spacer flex="1"/>
      <textbox id="InputVorname"/>
  </hbox>
  <hbox align="right">
      <button label="OK" oncommand="alert('OK');" />
      <button label="cancel" oncommand="alert('cancel');" />
  </hbox>

</vbox>

</window>
```

XAML: *meinbeispiel.xul*

```
<?xml version="1.0"?>

<?Mapping XmlNamespace="wf" ClrNamespace="System.Windows.Forms"
Assembly="System.Windows.Forms" ?>

<wf:Panel Name="meinbeispiel"
xmlns="http://schemas.microsoft.com/2003/xaml"
xmlns:wf="wf" xmlns:def="Definition"  Width="350" Height="180">

      <wf:Panel.Controls >
         <wf:Label Left="10" Width="250" Top="10" Text="mein Beispiel"
Font="Arial, 16pt"/>

         <wf:Label Left="10" Width="70" Top="43" Text="Nachname:" />
         <wf:TextBox Top="40" Name="Nachname" Left="90" Width="200"
Height="30" Text="" />
         <wf:Label Left="10" Width="70" Top="83" Text="Vorname:" />
         <wf:TextBox Top="80" Name="Vorname" Left="90" Width="200"
```

```
Height="30" Text="" />

         <wf:Button Top="120" Width="50" Left="10" Text="OK" />
         <wf:Button Top="120" Width="50" Left="70" Text="Cancel"/>

      </wf:Panel.Controls>
</wf:Panel>
```

XForms: *meinbeispiel.html*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<html
 xmlns="http://www.w3.org/1999/xhtml"
 xmlns:xforms="http://www.w3.org/2002/xforms">

<!--für XFormsplayer-->
      <object width="0" height="0" id="FormsPlayer"
classid="CLSID:4D0ABA11-C5F0-4478-991A-375C4B648F58">
            <b>FormsPlayer has failed to load! Please check your
installation.</b>
            <br />
            <br /></object>

      <?import namespace="xforms" implementation="#FormsPlayer"?>

      <head>
            <title>mein Beispiel!</title>
            <link rel="stylesheet" href="/main.css" type="text/css" />
            <xforms:model id="mdlMessage" >
                  <xforms:instance>
                        <message>
                              <vorname></vorname>
                              <nachname></nachname>
                        </message>
                  </xforms:instance>
            <xforms:submission id="sub1" method="get"  />
            </xforms:model>
      </head>

      <body>
            <div class="body">
            <div class="example">
            <xforms:input ref="nachname">
                  <xforms:label style="width:150px;"> Nachname
                  </xforms:label>
                  <xforms:hint>deiner Nachname</xforms:hint>
            </xforms:input>
                  <br/>

            <xforms:input ref="vorname">
                  <xforms:label style="width:150px;"> Vorname
                  </xforms:label>
                  <xforms:hint>deiner Vorname</xforms:hint>
            </xforms:input>
            <br/>
            <br/>

           <xforms:submit submission="sub1">
           <xforms:label>Submit</xforms:label>
           </xforms:submit>
```

```
            </div>
            </div>
      </body>
</html>
```

UIML:

(1): *to-java.uiml*

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 2.0 Draft//EN" "UIML2_0g.dtd">

<uiml>
  <head>
    <meta name="Date" content="25 February 2000"/>
  </head>
  <interface id="Fig10-28">
    <structure>
      <part class="Frame"         id="frame">
        <part class="Panel"       id="meinbeispiel">
          <part class="Label"     id="title"/>
          <part class="Label"     id="nachname"/>
          <part class="TextField" id="nachnameField"/>
          <part class="Label"     id="vorname"/>
          <part class="TextField" id="vornameField"/>

          <part class="Panel"     id="buttonPanel">
            <part class="Button"    id="okButton"/>
            <part class="Button"    id="cancelButton"/>
          </part>
        </part>
      </part>
    </structure>

    <style>
     <!-- Set the textual content for each part -->
      <property part-name="title"         name="text">mein
Beispiel:</property>
      <property part-name="nachname"
name="text">Nachname:</property>
      <property part-name="nachnameField"     name="columns">25</property>
      <property part-name="vorname"       name="text">Vorname:</property>
      <property part-name="vornameField"  name="columns">25</property>

      <property part-name="okButton" name="label">     Ok    </property>
      <property part-name="cancelButton"  name="label">Cancel</property>

     <!-- Set other properties for parts -->
      <property part-name="title"    name="font">Serif-bold-16</property>
      <property part-name="meinbeispiel"
name="layout">java.awt.GridBagLayout</property>

     <!-- Set GridBag constraints, on per-class basis -->
      <property part-class="Label"        name="anchor">WEST</property>
      <property part-class="TextField"    name="anchor">WEST</property>
      <property part-class="Label"        name="fill">HORIZONTAL</property>
      <property part-class="TextField"    name="fill">HORIZONTAL</property>
      <property part-class="Label"        name="gridwidth">1</property>
      <property part-class="TextField"    name="gridwidth">1</property>

     <!-- Set GridBag constraints, on per-part basis -->
```

```
      <property part-name="title"         name="anchor">NORTH</property>
      <property part-name="title"         name="fill">NONE</property>
      <property part-name="title"
name="gridwidth">REMAINDER</property>
      <property part-name="nachnameField"
name="gridwidth">REMAINDER</property>
      <property part-name="vornameField"
name="gridwidth">REMAINDER</property>

      <property part-name="buttonPanel"   name="anchor">SOUTH</property>
      <property part-name="buttonPanel"   name="fill">HORIZONTAL</property>
      <property part-name="buttonPanel"   name="insets">5,0,0,0</property>
      <property part-name="buttonPanel"   name="gridwidth">4</property>
    </style>

  </interface>

  <peers>
    <presentation base="Java_1.3_Harmonia_1.0"/>
  </peers>
</uiml>
```

(2): *to-html.uiml*

```
<?xml version="1.0"?>

<uiml>
  <interface>
    <structure>
      <Html>
        <Body>
          <Form>
            <P/>
            <Span  content="mein Beispiel"/>
            <P/>
            <Span  content="Nachname:"/>
            <Text  maxlength="20" />
            <P/>
            <P/>
            <Span  content="Vorname:"/>
            <Text  maxlength="20"/>
            <P/>
            <Submit  value="Submit"/>
            <Reset  value="Clear"/>
          </Form>
        </Body>
      </Html>
    </structure>
  </interface>

  <peers>
    <presentation how="replace" source="HTML_3.2_Harmonia_1.0.uiml#vocab"
base="HTML_3.2_Harmonia_1.0"/>
  </peers>
</uiml>
```

Qt-Designer: *meinbeispiel.ui*

```
<!DOCTYPE UI><UI version="3.1" stdsetdef="1">
<class>Form1</class>
<widget class="QDialog">
    <property name="name">
```

```
        <cstring>Form1</cstring>
    </property>
    <property name="enabled">
        <bool>true</bool>
    </property>
    <property name="geometry">
        <rect>
            <x>0</x>
            <y>0</y>
            <width>421</width>
            <height>198</height>
        </rect>
    </property>
    <property name="paletteBackgroundColor">
        <color>
            <red>85</red>
            <green>170</green>
            <blue>255</blue>
        </color>
    </property>
    <property name="caption">
        <string>Bachelorarbeit von PengXu</string>
    </property>
    <widget class="QLabel">
        <property name="name">
            <cstring>textLabel1</cstring>
        </property>
        <property name="geometry">
            <rect>
                <x>30</x>
                <y>30</y>
                <width>122</width>
                <height>41</height>
            </rect>
        </property>
        <property name="text">
            <string>&lt;p align="center"&gt;&lt;font
color="#ffffff"&gt;&lt;b&gt;Familienname&lt;/b&gt;&lt;/font&gt;&lt;/p&gt;</
string>
        </property>
    </widget>
    <widget class="QLineEdit">
        <property name="name">
            <cstring>lineEdit1</cstring>
        </property>
        <property name="geometry">
            <rect>
                <x>180</x>
                <y>40</y>
                <width>170</width>
                <height>30</height>
            </rect>
        </property>
    </widget>
    <widget class="QLineEdit">
        <property name="name">
            <cstring>lineEdit2</cstring>
        </property>
        <property name="geometry">
            <rect>
                <x>180</x>
```

```
                <y>90</y>
                <width>171</width>
                <height>31</height>
            </rect>
        </property>
    </widget>
    <widget class="QLabel">
        <property name="name">
            <cstring>textLabel2</cstring>
        </property>
        <property name="geometry">
            <rect>
                <x>20</x>
                <y>90</y>
                <width>131</width>
                <height>31</height>
            </rect>
        </property>
        <property name="text">
            <string>&lt;p align="center"&gt;&lt;b&gt;&lt;font
color="#ffffff"&gt;   Vorname&lt;/font&gt;&lt;/b&gt;&lt;/p&gt;</string>
        </property>
    </widget>
    <widget class="QPushButton">
        <property name="name">
            <cstring>pushButton1_2</cstring>
        </property>
        <property name="geometry">
            <rect>
                <x>240</x>
                <y>150</y>
                <width>111</width>
                <height>31</height>
            </rect>
        </property>
        <property name="text">
            <string>Cancel</string>
        </property>
    </widget>
    <widget class="QPushButton">
        <property name="name">
            <cstring>pushButton1</cstring>
        </property>
        <property name="geometry">
            <rect>
                <x>30</x>
                <y>150</y>
                <width>111</width>
                <height>31</height>
            </rect>
        </property>
        <property name="text">
            <string>Ok</string>
        </property>
    </widget>
</widget>
<layoutdefaults spacing="6" margin="11"/>
</UI>
```