Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Entwicklung eines XML-basierten Software-Prototypen zur Unterstützung der strategischen Internetbeschaffung

Semesterarbeit von David Hausheer
TIK-Nr. 2000.29

Prof. Dr. P. Schönsleben, Prof. Dr. B. Stiller
Betreuer: Dipl.-Ing. R. Alard, Dr. E. Wilde

14. August 2000

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<form name="general_information"
  owner="root" type="profile"
  created="Thu Jun 22 12:28:56 CEST 2000">
<fieldset name="company">
<fieldset name="address">
<field name="country_name" type="select">
<option value="switzerland"/>
<option value="italy"/>
[...]
```

Danksagung

Diese Arbeit wurde von verschiedenen Seiten unterstützt. Vom Betriebswissenschaftlichen Institut (BWI), vertreten durch Prof. Dr. P. Schönsleben und betreut durch Dipl.-Ing. R. Alard. Vom Institut für Technische Informatik (TIK), vetreten und betreut durch Prof. Dr. B. Stiller und in technischen Belangen unterstützt durch Dr. E. Wilde. An dieser Stelle möchte ich mich für ihre Betreuung und Unterstützung herzlich bedanken.

Voraussetzungen

Es wird im Wesentlichen vorausgesetzt, dass die im Bericht verwendeten technischen Begriffe und die in der Lösung verwendeten Technologien dem Leser bekannt sind. Am Ende dieses Berichtes ist eine ausführliche Liste mit Verweisen zu den Dokumentationen der verschiedenen Technologien aufgeführt. Auf der beiliegenden CD sind zudem weitere Tutorials mit Beispielen und Hilfen zu den verwendeten Standards und Tools enthalten.

Inhaltsverzeichnis

1 Einleitung

Moderne E-Technologien erleben derzeit im Zuge der raschen Entwicklung des Internets im allgemeinen und durch E-Business Anwendungen im speziellen einen enormen Aufschwung. Vor einigen Jahren war es Java als die "Internet-Programmiersprache", heute ist es XML als das universelle Datenformat, welches das World Wide Web erneut zu revolutionieren scheint. Das muss aber nicht heissen, dass deswegen die "alte" HTML-Technologie gleich überall durch XML ersetzt wird. Denn vor allem auf Clientseite und im privaten Bereich wird HTML wahrscheinlich noch lange nicht verschwinden, da es sehr einfach zu bedienen ist, die wichtigsten Funktionen weitgehend abdeckt und von allen bekannten Browsern gut unterstützt wird. Auf Serverseite hingegen und vor allem in Bereichen, wo die Funktionalität von HTML für gewisse Anwendungen nicht mehr ausreicht, wird in Zukunft wahrscheinlich vermehrt XML eingesetzt werden. Gerade im Business-to-Business-Bereich, wo es darum geht, grosse Datenmengen über das Internet auszutauschen, scheint deshalb ein starker Trend in Richtung XML-Unterstützung vorhanden zu sein.

Das Ziel der vorliegenden Arbeit war es, einen Software-Prototypen zu entwickeln, welcher die Möglichkeiten von neuen Internet-Technologien aufzeigt und mit den Anforderungen von heutigen E-Business Anwendungen verknüpft. Im Konkreten ging es um die Unterstützung des Austauschs von Informationen bei der strategischen Beschaffung über das Internet. Parallel zu dieser Arbeit wurde eine Studie erstellt über die Kriterien bei der strategischen Internetbeschaffung [13], auf der die vorliegende Arbeit im Wesentlichen aufbaut. Diese beiden Arbeiten entstanden in enger Zusammenarbeit.

Die strategische Beschaffung, die Aspekte wie das Lieferantenmanagement oder die Bündelung von Bedarfen umfasst, wird derzeit nur unzureichend unterstützt. Dies im Gegensatz zur operativen Beschaffung, welche die Abwicklung der Bestellvorgänge umfasst. Eine Mehrzahl der derzeit eingesetzten "Internet Procurement"-Lösungen fokussieren sich auf diese operativen Beschaffungsvorgänge. In der strategischen Beschaffung herrscht hingegen offensichtlich für zahlreiche Unternehmen ein grosser Handlungsbedarf. Die heutigen Lösungen sind dabei oft mit den folgenden Nachteilen verbunden:

- ein Lieferant muss zunächst auf die spezifische Webseite eines Kunden zugreifen (und diese erst einmal kennen)
- der Lieferant muss das potentielle Produkt "manuell" begutachten (es wird i.d.R. kein Datenformat eingesetzt, das ihn bei dieser Aufgabe unterstützen könnte)
- es besteht ein potentielles Konfliktpotential mit bestehenden Lieferanten (wenn z.B. ein neuer Lieferant für ein Bauteil evaluiert wird, das bereits beschafft wird)

Eine Lösung für diese Probleme scheinen internetbasierte Portale zu bieten, die z.B. ein Dienstleister zur Verfügung stellen könnte, über die Produkte spezieller Branchen anonym ausgeschrieben werden und für diese sich Lieferanten spezifisch bewerben könnten.

1.1 Aufgabenstellung

In Rahmen der vorliegenden Arbeit sollten die in der Einleitung beschriebenen Problematiken untersucht und eine technologische Lösung dafür gefunden werden. Dabei

sollten insbesondere moderne Webtechnologien und sie unterstützende Tools evaluiert und eingesetzt werden. Die zu entwickelnde grundlegende Datenstruktur sollte den Benutzer in seiner Aufgabe, der Strategischen Internetbeschaffung, unterstützen und ihm ermöglichen, die dazu nötigen Informationen über eine technologische Plattform (das "Portal") strukturiert auszutauschen. Die gefundene Lösung sollte schliesslich in eine Benutzeroberfläche (die "Website") integriert werden, welche eine gewisse Personalisierung und eine möglichst einfache Bedienung der Applikation erlaubt. Die gefundene Datenstruktur und die Benutzeroberfläche sollten danach auf einem Webserver implementiert und und die verschiedenen Funktionen getestet werden.

2 Rahmenbedingungen

Aus dem in der Einleitung beschriebenen Umfeld dieser Arbeit und der davon abgeleiteten Aufgabenstellung, können nochmals explizit die folgenden Anforderungen an die zu entwickelnde Lösung extrahiert werden:

2.1 Allgemeine Anforderungen

Folgende Funktionalitäten sollten u.a. von der Lösung erfüllt werden:

- für die Lieferantenauswahl soll es ermöglicht werden, Informationen in einem standardisierten Datenformat selektiv auszutauschen und diesen Vorgang soweit als möglich zu automatisieren
- die Lösung soll eine modulare Aufbauweise besitzen und erweiterbar sein
- es soll eine Administratorensicht implementiert werden, welche die Datenpflege, Erweiterung und Aktualisierung der Datenbank erlaubt
- neben den technischen Aspekten eines Produktes sollen kooperationsrelevante Kriterien in die Beschreibung mit eingebunden werden (diese wurden im Rahmen einer zweiten Studienarbeit [13] erarbeitet)
- die Lösung darf im Hintergrund beliebig komplex sein, für den Benutzer muss sie aber so einfach wie möglich zu bedienen sein

2.2 Technische Rahmenbedingungen

Neben den allgemeinen Anforderungen an die Lösung sollten insbesondere folgende Technologien evaluiert und in die Arbeit eingebunden werden:

- die Lösung soll auf modernen Webtechnologien (XML, XSL) aufbauen
- die Benutzer- und Administratorsicht sollen webbasiert und browserunabhänig sein (da XML auf Clientseite bisher noch zuwenig unterstützt wird, wird als kleinster gemeinsamer Nenner ein HTML-Client wie Netscape 4.7 oder Explorer 5.0 vorausgesetzt)
- für die Datenspeicherung soll ein XML-Server der Software AG (Tamino) eingesetzt werden

3 Vorgehensweise

Die Implementierung der Lösung kann im Wesentlichen in vier verschiedene Phasen (Einarbeitung - Anforderungsanalyse - Entwurf - Realisierung) eingeteilt werden. Da bei der Umsetzung vor allem auch neue Technologien evaluiert werden sollten, nahmen die Einarbeitungs- und Anforderungsphase eine relativ grosse Zeit in Anspruch. Jedoch kam auch der Entwurfs- und Realisierungsphase eine grosse Bedeutung zu, da die Lösung schliesslich eine gewisse Funktionalität erreichen sollte, die für die Benutzer einen sichtbaren Nutzen generiert und damit die Akzeptanz wesentlich erhöht.

Einarbeitungsphase Zu Beginn wurden im Rahmen der Einarbeitungsphase einige Beispiele basierend auf XML implementiert und damit die Möglichkeiten dieser neuen Technologie untersucht. Zu diesen Möglichkeiten zählen beispielsweise Sortier- und Suchfunktionen, die Abbildung der Datenhierarchie in der DTD sowie die Selektion und Darstellung der Daten mit XSL-Stylesheets [4]. Gleichzeitig wurden verschiedene Tools evaluiert, die in der Lage sind, XML und XSL zu verarbeiten. Dabei stellte sich heraus, dass XML-Parser zwar in den meisten Umgebungen (untersucht wurden u.a. Perl, Java, PHP) vorhanden sind, XSL-Prozessoren hingegen erst wenig verbreitet sind, da dieser Standard noch nicht stabil ist. Es stellte sich zudem die Frage, ob es sinnvoller ist, die Transformation der XML-Dokumente nach HTML serverseitig oder clientseitig vorzunehmen. Da XML clientseitig momentan erst durch den Internet Explorer unterstützt wird, fiel die Entscheidung auf die serverseitige Transformation. Die dazu verwendeten Tools werden im Abschnitt "Architektur" näher erläutert.

Anforderungsanalyse Nachdem in die Stärken von XML und XSL und der sie unterstützenden Tools durch die Evaluation bekannt waren, konnten die Anforderungen an die Lösung erarbeitet und mit den Möglichkeiten dieser neuen Technologien verknüpft werden. Aus dem daraus resultierenden Anforderungsprofil an die Funktionalität der Lösung konnte der Rahmen dieser Arbeit festgelegt werden. Der Fokus wurde dabei in erster Linie auf die Kundenseite gelegt. Auf mögliche Erweiterungen dieser Lösung wird im letzten Abschnitt näher eingegangen.

Entwurfsphase Aus den erarbeiteten Anforderungen und den abzubildenden Daten, konnte eine geeignete Datenstruktur entworfen werden. Zunächst wurde eine fixe dreistufige Datenhierarchie eingesetzt. Es zeigte sich aber, dass diese Datenstruktur zuwenig flexibel war. Deshalb wurde eine n-stufige Hierarchie mit rekursiven Elementen zugelassen, was die Lösung zwar komplexer gestaltete, dafür aber die nötige Flexibilität in der Anwendung brachte. Daraufhin wurden XSL-Stylesheets entwickelt, die in der Lage waren, diese Datenstrukturen zu verarbeiten. Gleichzeitig wurde eine Grundstruktur für das Benutzerinterface in HTML entworfen, in das die Kernelemente der Datentransformation eingebunden wurden. Dabei wurde grossen Wert auf eine möglichst modulare Aufbauweise der Lösung gelegt.

Realisierungsphase Schliesslich mussten die erarbeiteten Datenstrukturen im XML-Server angelegt und die angefertigten Stylesheets in die Server-Architektur eingebunden werden. Die nötigen Methoden in den Java Servlets mussten implementiert und die Schnittstellen zwischen den einzelnen Komponenten der Architektur aufeinander abgestimmt werden. Mit Testdaten konnten die einzelnen Funktionalitäten überprüft wer-

den. Auf diese Tests folgten entsprechende Modifikationen und Erweiterungen. Danach wurden die endgültigen Daten über die Administrationsseite erfasst. Damit konnten die verschiedenen Funktionen des Benutzerinterface getestet und die Arbeit schliesslich abgeschlossen werden.

4 Datenstruktur

In der dieser Arbeit zugrundeliegenden Studie über die Strategische Internetbeschaffung [13] wurde ein umfassender Kriterienkatalog erarbeitet, der die bei der Beschaffung relevanten Daten enthält. Bei der Suche nach einer geeigneten Datenstruktur für die Abbildung dieser Kriterien auf Webtechnologien lagen folgende Überlegungen zugrunde:

- die Hierarchie der Daten soll direkt in der Datenstruktur abgebildet werden können und diese soll einfach transformierbar sein
- die Datenstruktur soll modular aufgebaut sein, damit die Semantik der Kriterien unabhängig von der gewählten Struktur definiert werden kann

Unter der "Datenhierarchie" wird dabei die reale Struktur der Daten verstanden. Dagegen ist mit der "Datenstruktur" die Struktur des Datenmodells gemeint (Abbildung 1).

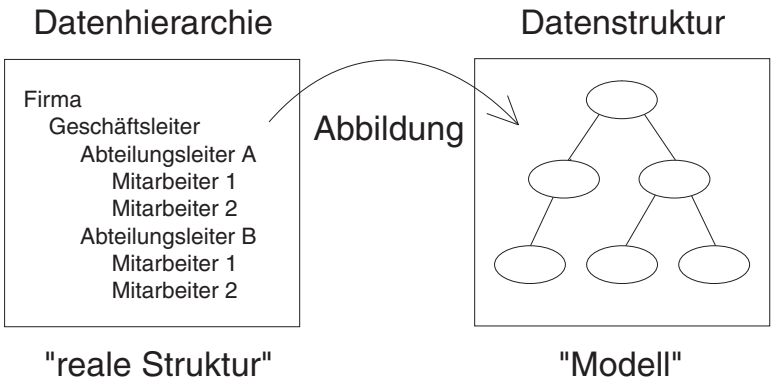

Abbildung 1: Abbildung der Daten

Die hierarchische XML-Struktur hat gegenüber relationalen Datenstrukturen (wie z.B. SQL) den enormen Vorteil, dass sich die Datenhierarchie direkt in der Datenstruktur abbilden lässt. In relationalen Datenbanken können Datenhierarchien dagegen nur über sogenannte $1:1$- oder $1:n$-Abhängigkeiten zwischen mehreren verschiedenen Tabellen beschrieben werden. Transformationen solcher Datenbanken können daher sehr komplex werden. In XML lassen sich dagegen durch die Schachtelung der Elemente beliebige Datenhierarchien direkt abbilden und Transformationen dieser Strukturen lassen sich durch Stylesheets relativ einfach bewerkstelligen. Im Rahmen dieser Arbeit wurde der Fokus auf XML gelegt. Es ist aber grundsätzlich fraglich ob XML den üblichen Anforderungen am besten genügt. Sobald die Performance, die in dieser Arbeit nicht berücksichtigt wurde, eine grössere Rolle spielt, ist man möglicherweise mit einer anderen, daraufhin optimierten Technologie besser bedient, da XML eine hohe Datenredundanz aufweist. Auf diesen Nachteil wird später noch näher eingegangen.

Um die Semantik der Kriterien unabhängig von der Struktur definieren zu können, wurde eine modulare dreiteilige Datenstruktur entworfen (Abbildung 2). Die drei Komponenten dieser Struktur unterscheiden sich folgendermassen: Die "Grundstruktur" bildet gewissermassen die "atomare" Struktur der Daten. Sie wurde in der DTD abgebildet. In dieser Grundstruktur können verschiedene "Formularstrukturen", d.h. verschiedene Semantiken von Kriterien abgebildet werden. Diese bilden XML-Dokumente, also Instanzen der DTD. In der "Formulardaten"-Struktur werden schliesslich die Daten aus einem "ausgefüllten" Formular der zugrundeliegenden Formularstruktur zugeordnet.

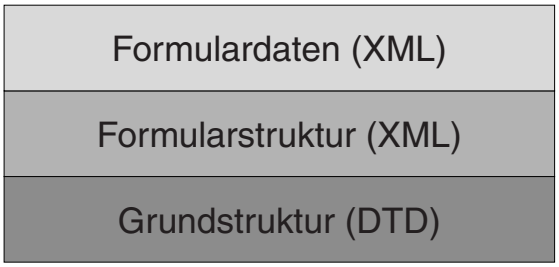

Abbildung 2: Datenstruktur

Diese Unterscheidung erinnert einen möglicherweise an die Struktur von XML-Schemas. Tatsächlich wird dort dieselbe Trennung von Grundstruktur (DTD) und Datenstruktur (XML-Schema) vorgenommen. Die gewählte Struktur besitzt aber noch eine weitere Affinität zu einem derzeit entwickelten Standard des W3-Konsortiums: den XForms. Dieser Standard soll in Zukunft den Formularteil von XHTML bilden. Der Umfang von XForms ist natürlich um einiges umfassender als die in dieser Arbeit beschriebene Datenstruktur. Sobald für XForms und XHTML Implementationen vorhanden sind, wäre allenfalls eine Übernahme dieser Strukturen in die vorliegende Lösung zu prüfen.

Alternativ zur gewählten Struktur wäre es auch möglich gewesen, sowohl die Hierarchie als auch die Semantik der Kriterien in der DTD zu definieren. Dies hätte zwar eine der drei Komponenten erübrigt und somit die gesamte Datenstruktur vereinfacht, dafür hätten aber die Kriterien nicht mehr unabhängig von der Grundstruktur definiert und dynamisch dazu aufbereitet werden können.

4.1 Grundstruktur (DTD)

Definition Die Grundstruktur stellt die Menge aller möglichen Formularstrukturen dar. Formell wird sie durch die zugrundeliegende DTD (`form.dtd`) dargestellt. Die Grundstruktur ermöglicht eine einheitliche Klassifizierung und hierarchische Abbildung der Daten.

form.dtd

```
<!ELEMENT form (fieldset*)>
<!ATTLIST form name CDATA #REQUIRED>
<!ATTLIST form owner CDATA #REQUIRED>
```

```
<!ATTLIST form type CDATA #IMPLIED>
<!ATTLIST form created CDATA #IMPLIED>

<!ELEMENT fieldset (field*,fieldset*)>
<!ATTLIST fieldset name CDATA #REQUIRED>
<!ATTLIST fieldset type CDATA #IMPLIED>

<!ELEMENT field (option*)>
<!ATTLIST field name CDATA #REQUIRED>
<!ATTLIST field type (text|textarea|password|
          select|checkbox|radio) #REQUIRED>
<!ATTLIST field unit CDATA #IMPLIED>
<!ATTLIST field size CDATA #IMPLIED>

<!ELEMENT option EMPTY>
<!ATTLIST option value CDATA #REQUIRED>
```

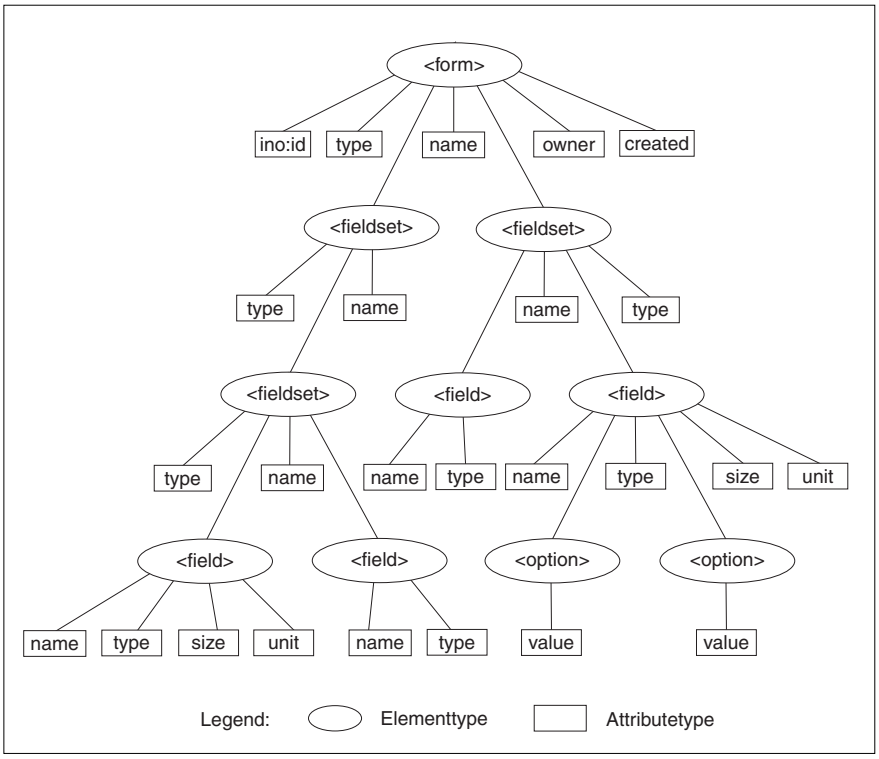

Abbildung 3: Baumstruktur eines Formulares

In Abbildung 3 wird die Hierachie dieser DTD als Baumstruktur illustriert. Im folgenden werden nun die einzelnen Elemente etwas näher beschrieben.

Form Dies ist das Toplevel-Element. Dieses Element kann nur `fieldset`-Elemente beinhalten. Dies ist zwar eine Einschränkung, die Praxis zeigte aber, dass diese nicht ins Gewicht fällt. Das `form`-Element beschreibt - wie in HTML - ein Formular.

Fieldset Dieses Element strukturiert das Formular. Neben den `field`-Elementen können - im Gegensatz zu HTML - auch `fieldset`-Elemente selber wieder enthalten sein. Dieser rekursive Aufruf hat eine zentrale Bedeutung und wirkt sich stark auf alle späteren Methoden aus. Zuerst war dies nicht so vorgesehen. Es zeigte sich aber, dass eine dreistufige Hierarchie alleine nicht ausreicht, die Struktur der Kriterien abzubilden, ohne starke Einschränkungen in Kauf nehmen zu müssen.

Field Dieses Element bildet jeweils zusammen mit den `option`-Elementen das Ende eines Astes in der Baumstruktur. Es steht stellvertretend für diejenigen Elemente, die in HTML als Formularfelder vorkommen können: `input` und `select`. Die Attribute, denen hier die grösste Bedeutung zukommt, werden im nächsten Abschnitt näher erläutert.

4.2 Formularstruktur

Definition Die Formularstruktur stellt eine Instanz der Grundstruktur dar. Formell wird sie durch ein XML-Dokument beschrieben, das eine Instanz der oben abgebildeten DTD darstellt. Die Formularstruktur stellt gewissermassen ein "leeres" Formular dar.

Im Wesentlichen werden vier Informationen in der Formularstruktur abgebildet:

- gewisse Grundinformationen zu jedem Formular in den Attributen `created` (Zeitstempel), `type` (Name der verwendeten Semantik) und `owner` (`root`, oder die `id` des Erstellers)
- die Semantik der Kriterien in den `name`-Attributen der `fieldset`- und `field`-Elemente
- die Information über den Typ eines Feldes in den Attributen `type`, `size` und `unit`
- die vordefinierten Werte für die Auswahlfelder in den `value`-Attributen der `option`-Elemente

Das folgende Beispiel (ein Ausschnitt aus dem Standard-Formular) soll dies veranschaulichen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<form ino:id="173" created="Thu Jun 22 12:28:56 CEST 2000"
      type="profile" name="default_profile" owner="root">

  <fieldset name="general_information">
    <field name="name" type="text" size="30"/>

    <fieldset name="address">
      <field name="street" type="text" size="30"/>
      <field name="zip_code" type="text" size="5"/>
      <field name="city" type="text" size="30"/>
      <field name="country" type="select">
        <option value="switzerland"/>
        <option value="germany"/>
        <option value="france"/>
```

```
        <option value="italy"/>
      </field>
      <field name="phone_number" type="text" size="30"/>
      <field name="fax_number" type="text" size="30"/>
    </fieldset>
    ...
  </fieldset>
  ...
</form>
```

Die Attribute des `field`-Elementes haben dabei folgende Bedeutung:

Type Dieses Attribut beschreibt den Darstellungstyp eines Formularfeldes. Sechs Möglichkeiten stehen - wie in HTML - in dieser Lösung zur Verfügung: `text`, `textarea`, `password`, `select`, `checkbox` und `radio`. Eine Erweiterung dieser Typen ist natürlich möglich. Einige Beispiele dafür wären u.a. Links, Bilder oder die Möglichkeit von File-Upload. Auch Typen, die zwar von den sechs Grundtypen abgeleitet werden können, aber immer wieder verwendet würden, könnten dazu erweitert werden, z.B. Ja/Nein-Felder für den boolschen Datentyp und weitere Datentypen. Die Stylesheets für die Darstellung der Formulare müssten entsprechend ergänzt werden.

Unit Falls es sinnvoll ist, kann einem Feld mit diesem Attribut eine Einheit zugeordnet werden. Als Beispiel könnte die Währung genannt werden. In der vorliegenden Implementation wurde dieses Attribut allerdings nicht verwendet. Beim Matching der Formulardaten, auf das im letzten Abschnitt noch eingegangen wird, würde das `unit`-Attribut aber eine grosse Bedeutung in der Spezifikation und Gewichtung der Daten erhalten.

Size Diese Attribut wird momentan nur für die Typen `text` und `textarea` verwendet. Damit kann die Darstellungslänge der Felder in HTML spezifiziert werden.

4.3 Formulardaten

Definition Die Formulardaten stellen eine Instanz der Formularstruktur dar. Formell werden sie ebenfalls durch ein XML-Dokument (basierend auf der DTD `postform.dtd`) beschrieben, allerdings wird die Datenhierarchie nicht mehr in den Elementen sondern nur noch im Feldnamen abgebildet. Es werden nur die ausgefüllten Felder übernommen. Die Formulardaten stellen gewissermassen ein "ausgefülltes" Formular dar.

Das folgende Beispiel zeigt ein mögliches Formulardaten-Dokument für die obige Formularstruktur.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<postform idref="173">

  <field name="1.1">
    <option value="Muster AG"/>
  </field>
```

```
    <field name="1.2.1">
      <option value="Musterstrasse 23"/>
    </field>

    <field name="1.2.2">
      <option value="9999"/>
    </field>

    <field name="1.2.3">
      <option value="Musterstadt"/>
    </field>

    <field name="1.2.4">
      <option value="switzerland"/>
    </field>

  </postform>
```

Es fällt auf, dass die semantischen Bezeichnungen der Feldnamen ersetzt worden sind und die Datenstruktur keine Hierarchie mehr besitzt. Um eine Zuordnung zwischen der Formularstruktur und den Formulardaten gewährleisten zu können, wird die Position des Feldes in der Baumstruktur in eine eindeutige Nummer abgebildet. So erhält beispielsweise das Feld `general_information.address.city` die Nummer `1.2.3` zugeordnet.

4.4 Vor- und Nachteile von XML

Nach der Evaluation von XML als Datenstruktur können abschliessend folgende Vor- und Nachteile genannt werden:

Vorteile

- Im Vergleich zu HTML bringt die XML-Struktur eine Entflechtung zwischen der Grundstruktur, den Daten und der Präsentation. Präsentationslayouts lassen sich durch Stylesheets modular adaptieren. Die klare Trennung ermöglicht die völlig dynamische Erstellung der verschiedenen Sichten, z.B. der HTML-Sicht.

- Die Hierarchie der Daten lässt sich in XML 1:1 als Baumstruktur abbilden. Rekursive Elemente erlauben sogar die Mehrfachverwendbarkeit bereits definierter Strukturen. Gerade bei Daten mit einer komplexen Hierarchie sind diese Möglichkeiten von enormem Vorteil.

- Die XSLT-Technologie [5], auf die später noch ausführlich eingegangen wird, kann für viele verschiedene Methoden verwendet werden. Neben den verschiedenen Präsentationslayouts lassen sich auch Such- und Sortieralgorithmen, Datenfilter und Transformationen komplexer Strukturen implementieren. Beispiele dafür werden unter dem Abschnitt "Implementation" vorgestellt.

Nachteile

- Transformationen von Baumstrukturen gerade mit rekursiven Elementen können wegen der schwierigen Handhabbarkeit dieser Strukturen komplex werden, v.a. wenn sie ausserhalb von XSLT (z.B. in Java) implementiert werden müssen. Dagegen sind zweidimensionale Tabellen viel besser und einfacher zu verwalten.
- XML-Dokumente besitzen eine hohe Datenredundanz. Diese Technologie wurde mit der Absicht entwickelt, Daten in einer auch für die Menschen lesbaren Form abzubilden. Bei der Transformation von XML-Dokumenten müssen diese zuerst eingelesen (geparst) durch den XSL-Prozessor verarbeitet und schliesslich wieder in eine lesbare Form gebracht (serialisert) werden. Daraus resultiert eine insgesamt eher schwache Performance.

5 Architektur

Die Architektur der Lösung besteht aus drei Teilen (HTTP-Client, Web-Server und XML-Server). Diese Komponenten kommunizieren miteinander über das TCP/IP-Protokoll. Somit kann jede Komponente physikalisch auf je einem separaten Rechner implementiert werden, die über das Internet miteinander verbunden sind. In Abbildung 4 ist die generelle Verteilung der einzelnen Komponenten im Gesamtsystem ersichtlich.

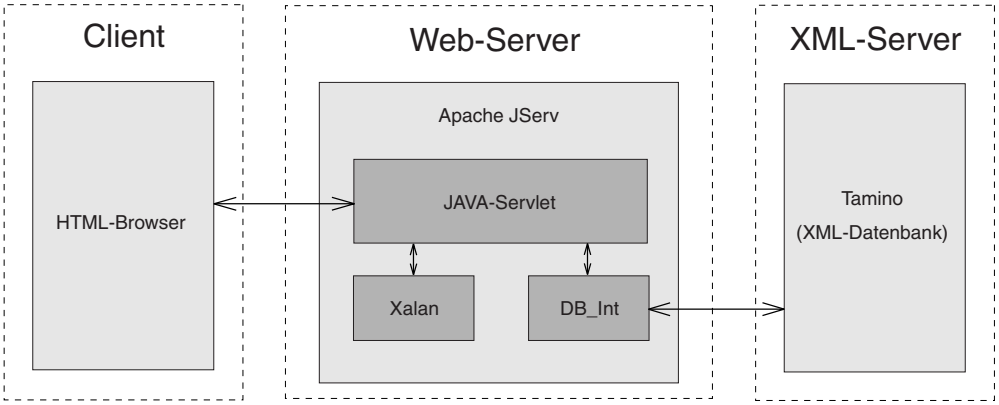

Abbildung 4: Architektur des Software-Prototypen

5.1 HTTP-Client

Als HTTP-Client kann auf Benutzerseite ein HTML-Browser wie Netscape 4.7 oder Internet Explorer 5.0 eingesetzt werden. Auch andere Browser sollten prinzipiell mit dieser Lösung funktionieren. Die Anforderungen an den Browser richten sich nach der HTML-Spezifikation des W3C [3]. Auf Administratorenseite wird ein Browser mit UTF-8-encoding für die Sprachverwaltung benötigt. Mit Netscape ergaben sich Probleme bei der Schrifterkennung, deshalb ist hier ein Internet Explorer ab Version 5.0 nötig. Für die Navigation muss zudem JavaScript aktiviert werden.

In Zukunft ist es auch denkbar ein eigenes Java Interface als HTML- oder sogar XML-Client zu verwenden. Je nach Entwicklung der Browser kann in einer späteren Lösung

auch die XHTML-Spezifikation mit XForms Unterstützung eingebunden werden. Dies würde auf Serverseite natürlich einige Umformungen wesentlich vereinfachen.

5.2 Web-Server

Der Web-Server basiert momentan auf dem frei erhältlichen Apache Server mit JServ-Erweiterung. Die verschiedenen Methoden wurden in Java-Servlets implementiert. Prinzipiell kann irgendein Server mit Java Servlet Unterstützung verwendet werden. Folgende Produkte wurden in dieser Lösung eingesetzt:

- Apache HTTP Server (Version 1.3.12), *http://www.apache.org/*
- Apache JServ (Version 1.1), *http://java.apache.org/*
- Java Development Kit (JDK) (Version 1.2.2, für die Kompilierung eigener Servlets), *http://java.sun.com/*
- Java Runtime Environment (Version 1.2, in JDK enthalten)
- Java Servlet Development Kit (JSDK) (Version 2.0), *http://java.sun.com/*
- Xalan (XSLT Stylesheet Prozessor) (Version 1.0.1), *http://xml.apache.org/*

Eine Hilfe für die Installation und Konfiguration der einzelenen Komponenten wird mit den einzelnen Produkten mitgeliefert. Damit auf die Java Servlets zugegriffen werden kann, muss der Java Classpath richtig gesetzt und eine Servlet Zone errichtet werden. Folgende Anpassungen sind in den Konfigurationsdateien nötig:

Apache JServ (jserv.properties) Der Java Classpath muss alle benötigten Klassen enthalten:

```
wrapper.classpath=/path/to/ApacheJServ.jar
wrapper.classpath=/path/to/jsdk.jar
wrapper.classpath=/path/to/xalan.jar
```

Danach können die benötigten Servlet Zonen errichtet werden:

```
zones=portal,portaladmin
```

Für jede Zone muss ein separates Konfigurationsfile definiert werden:

```
portal.properties=/path/to/portal/zone.properties
portaladmin.properties=/path/to/portaladmin/zone.properties
```

Apache JServ (jserv.conf) Die neuen Zonen müssen auf den gewünschten URL-Pfad gemountet werden:

```
ApJServMount /admin /portaladmin
ApJServMount /portal /portal
```

Servlet Zone (zone.properties) Schliesslich muss in den Servlet Zonen der Pfad für die eigenen Java-Klassen angegeben und die weiteren Init-Parameter übergeben werden, die in den Servlets verwendet werden.

```
repositories=/path/to/portaladmin,/path/to/portal
servlets.default.initArgs=XSL_DIR=/path/to/stylesheets-dir,
 DB_URL=/url/to/tamino-server
```

Optional können die Java-Klassen noch mit einem Alias-Namen versehen werden:

```
servlet.form.code=Form
```

Die Servlets sind dann unter der folgenden URL zugänglich:

```
http://www.mydomain.com/admin/form
```

Die einzelnen Java Servlets sind alle nach dem gleichen System aufgebaut. Je nachdem welche HTTP-Funktionen (POST, GET, etc.) benützt wurden, mussten die entsprechenden Methoden (`doPost, doGet`) implementiert werden. Die einzelnen Funktionen der Servlets, u.a. die Sessions und die Parameterübergabe werden an dieser Stelle nicht näher beschrieben. Es wird an die entsprechende Dokumentation [10] verwiesen. Die Xalan-Klassen `Xalan` werden zur XSL-Transformation der Daten von XML in HTML benötigt. Diese Funktionen werden in der Klasse `Transform.java` im Abschnitt "Implementation" näher erläutert. Die Schnittstelle zum Tamino-Server wurde in der Klasse `InoConnection.java` implementiert. In Abbildung 5 ist zur Illustration der Schnittstellen der zeitliche Ablauf und die Kommunikation zwischen den einzelnen Komponenten bei einem Formular Request dargestellt.

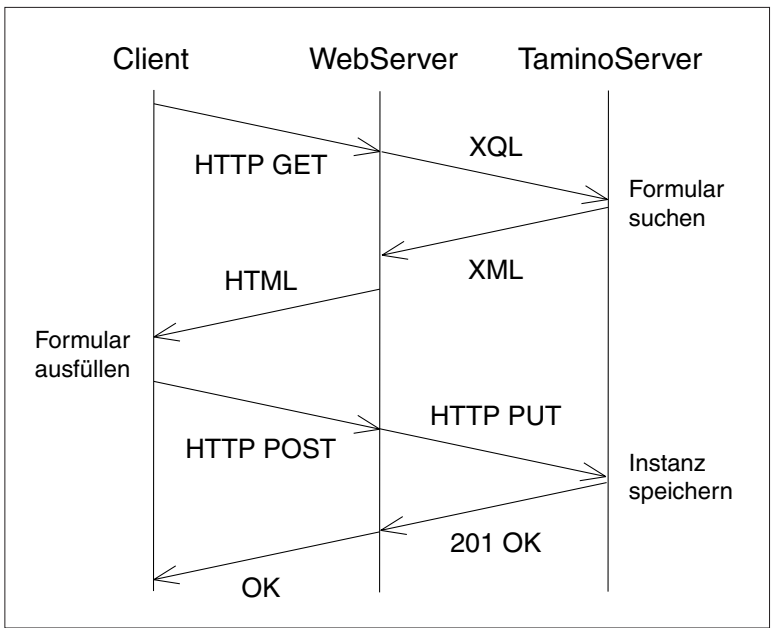

Abbildung 5: Formular Request

5.3 XML-Server (Tamino)

Schliesslich bildet der Tamino-Server die dritte Komponente dieser Architektur. Tamino kann XML-Dokumente verwalten und erlaubt einen schnellen Zugriff auf die Daten. Der Zugriff verläuft über XQL-Requests [6] beziehungsweise HTTP-Requests. Für eine genaue Beschreibung des Tamino Servers sei an die entsprechende Dokumentation [12] verwiesen. Alternativ zum Tamino-Server könnte als XML-Server auch ein eigenes Modul verwendet werden. Als Anforderungen an dieses Modul ergeben sich die Indexierung der verwalteten Dokumente und eine Implementierung von XPath [2] bzw. XQL [6].

6 Implementation

Die vorliegende Lösung bietet im Wesentlichen zwei Sichten, eine Benutzer- und eine Administratorensicht, wobei sich die Benutzersicht wie erwähnt auf die Kundenseite beschränkt. Im folgenden werden die Funktionen der einzelnen Javaklassen und Stylesheets für die beiden Sichten beschrieben und anhand dieser die Kernbestandteile der Implementation erklärt. In Tabelle 1 ist eine Übersicht aller Module abgebildet.

Benutzersicht	**Administratorensicht**
Registration / Login	Formularverwaltung
Formulare	Sprachverwaltung
Produkte	Benutzerverwaltung
Resultate	

Tabelle 1: Übersicht aller Module

6.1 Benutzersicht

Registration / Login

Die verschiedenen Komponenten dieses Modules sind in Tabelle 2 abgebildet.

`Registration.java`	`Login.java`
`registration.xsl`	`login.xsl`
	`enter.xsl`
	`home.xsl`
`company.dtd`	

Tabelle 2: Registration / Login

Die Klasse `Registration.java` übernimmt die Registrierung von neuen Benutzern. Das hierzu vom Benutzer ausgefüllte Registrations-Formular wird auf dem XML-Server gespeichert. Ebenso wird ein neues Firmenprofil erstellt. Dazu wird ein XML-Dokument mit der Struktur `company.dtd` mit dem Namen der Firma, der Emailadresse des Benutzers (als Login- oder Username) und dem Passwort zusammengestellt. Dieses Dokument wird separat auf dem XML-Server abgelegt und dient der Benutzeridentifikation beim Login. Der Klasse `Login.java` kommen mehrere Aufgaben zu. Einerseits ist sie zuständig für die Vergabe von neuen Sessions. Sobald sich ein Benutzer erfolgreich angemeldet hat, wird eine Session mit seinen Daten aus der Registrierung erstellt. Dabei

überprüft das Stylesheet `enter.xsl` die Gültigkeit von Username und Passwort. Falls bereits eine Session für diesen Benutzer besteht, wird anhand der SessionID überprüft, ob diese noch gültig ist, denn durch den Abmeldevorgang wird eine gültige Session gelöscht. Falls die Anmeldung erfolgreich war hat der Benutzer Zugriff auf die gesamten Daten (Produkte, Formulare, Resultate, Profil) seiner Firma. Diese Übersicht wird durch das Stylesheet `home.xsl` generiert. Darin werden sämtliche erstellten Formulare und deren Resultate, sowie die erstellten Produkte dargestellt. Es werden Links zur Verfügung gestellt, die es ermöglichen neue Formulare oder neue Produkte zu erstellen, sowie diese anzuzeigen oder wieder zu löschen.

Formulare

Die verschiedenen Komponenten dieses Modules sind in Tabelle 3 abgebildet.

NewForm.java	SaveForm.java	DeleteForm.java
newform.xsl	filter.xsl	
form.dtd		

ShowForm.java	PostForm.java
unframedform.xsl	
form.dtd, postform.dtd	

Tabelle 3: Formulare

Die Klasse `NewForm.java` ermöglicht es dem Benutzer ein neues Formular zu erstellen. In Abbildung 6 sind die einzelnen Schritte bei diesem Vorgang dargestellt.

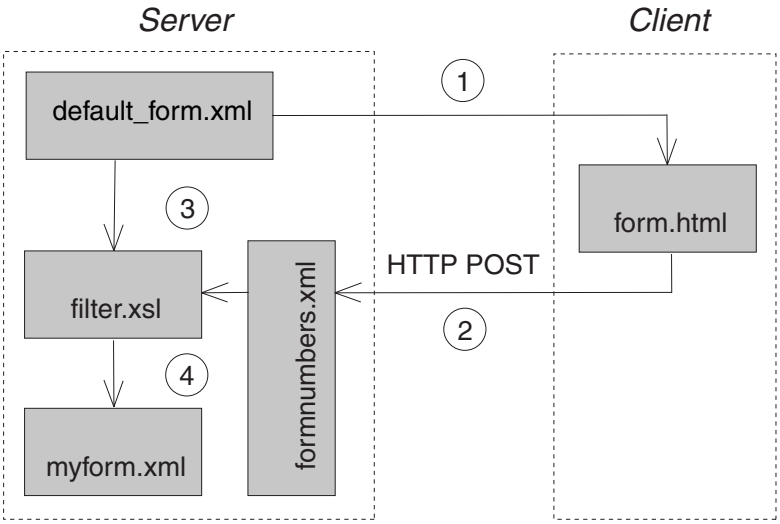

Abbildung 6: Formular erstellen (Filterung der Formularstruktur)

Der Benutzer erhält zunächst das vom Administrator zur Verfügung gestellte Dokument `default_form.xml`. Es folgt als Beispiel ein kleiner Ausschnitt aus dieser Struktur:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<form>
  <fieldset name="general_information">
    <fieldset name="address">
      <field name="country_name">
        <option value="switzerland"/>
        <option value="italy"/>
      </field>
    </fieldset>
  </fieldset>
</form>
```

Dieses Dokument wird durch das Stylesheet `newform.xsl` nach HTML transformiert ①:

```
<html>
  <input type="checkbox" name="1" value="yes">
  <input type="checkbox" name="1.2" value="yes">
  <input type="checkbox" name="1.2.2" value="yes">
</html>
```

Damit die Hierarchie der Daten durch die Transformation nach HTML nicht verloren geht, musste den einzelnen Elementen eindeutige Namen zugeordnet werden. Dafür wurde der Befehl `<xsl:number>` verwendet. Der Benutzer kann nun durch Auswahl der entsprechenden Checkboxes einzelne Fields oder Fieldsets in sein Formular übernehmen oder weglassen. Nachdem das Formular entsprechend angepasst wurde, werden die Angaben mit dem `HTTP POST` Befehl an den Webserver übermittelt ②:

```
1=yes
1.2.2=yes
```

In diesem Beispiel wurden zwei Knoten (1 und 1.2.2) ausgewählt. Weil Knoten 1.2.2 in der Hierachie von Knoten 1.2 abhängt, dieser aber nicht ausgewählt wurde, darf Knoten 1.2.2 in der resultierenden Struktur nicht vorkommen. Die übermittelten Daten werden deshalb auf Serverseite von der Klasse `SaveForm.java` wie folgt verarbeitet: Zunächst wird ein Dokument mit der Filterbedingung (`formnumbers.xml`) erstellt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<formnumbers>
  <formnumer>1</formnumer>
  <formnumer>1.2.2</formnumer>
</formnumbers>
```

Dieses Dokument wird zusammen mit dem ursprünglichen Standard-Formular an ein XML Filter (`filter.xsl`) übergeben ③. Zentral ist hier das Template `node()`:

```
<xsl:template match="node()">

  <xsl:variable name="number">
    <xsl:number level="multiple" count="fieldset|field" format="1.1"/>
  </xsl:variable>
```

```
    <xsl:if test="$formnumbers[formnumber=$number]">
      <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
    </xsl:if>

  </xsl:template>
```

Dieses Template kopiert alle vom aktuellen Knoten abwärts liegenden Elemente und Attribute nur dann, falls die aktuelle Nummer in der Menge `formnumbers.xml` enthalten ist. Damit wird sichergestellt, dass bei der Nichtauswahl eines Fieldsets alle darunter liegenden Elemente weggelassen werden, auch wenn diese ausgewählt wurden. Als Resultat wird schliesslich ein XML-Dokument (`myform.xml`) generiert ④, das eine Teilmenge des ursprünglichen Standard-Formulares darstellt.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<form>
  <fieldset name="general_information"/>
</form>
```

Ein neu erstelltes Formular kann nun durch die Klasse `DeleteForm.java` gelöscht oder durch `ShowForm.java` dargestellt werden. Diese Sicht wird durch das Stylesheet `unframedform.xsl` generiert. Wie der Name schon sagt, wird hier auf den üblichen Rahmen verzichtet, damit das Formular auch in andere Webseiten integriert werden kann. Das gesamte Formular wird dabei in einer einzigen Tabelle dargestellt. Damit die Baumstruktur korrekt in HTML-Tabellen transformiert wird mussten zunächst einige Probleme gelöst werden, welche auf die im allgemeinen unausgeglichene Länge der Äste zurückzuführen sind. Die Darstellung verwendet das bekannte HTML-Attribut `colspan`, das in den `<td>`-Tags verwendet wird. Die Zahl der überspannten Spalten eines Field-Names ergibt sich dabei aus der folgenden Formel:

```
<td colspan="$tot_deepness-$rel_deepness+1+$is_fieldset">
```

In Abbildung 7 wird diese Formel anhand eines Beispieles veranschaulicht. Die maximale

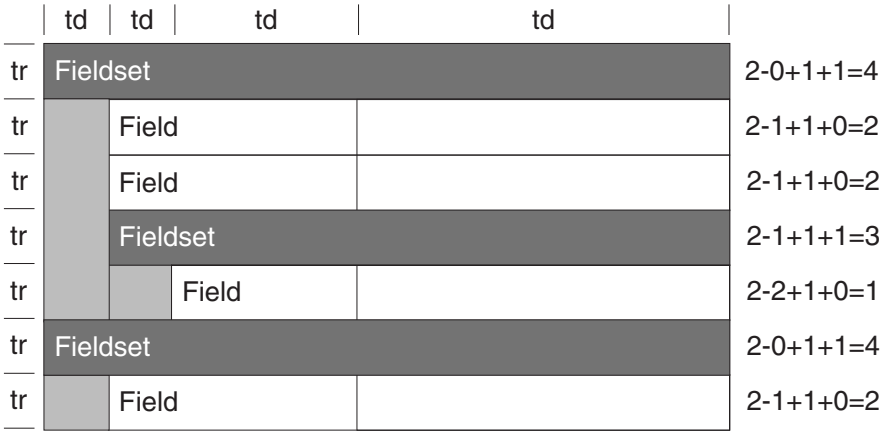

Abbildung 7: Spaltenüberspannung

Baumtiefe (`tot_deepness`) lässt sich folgendermassen berechnen:

```
<xsl:variable name="tot_deepness">
  <xsl:for-each select=".//field[position()=1]">
    <xsl:sort select="count(ancestor::fieldset)"/>
    <xsl:if test="position()=last()">
      <xsl:value-of select="count(ancestor::fieldset)"/>
    </xsl:if>
  </xsl:for-each>
</xsl:variable>
```

Dabei wird über die Endknoten (`field`) iteriert. Die Baumtiefe eines Knotens ergibt sich aus dem Ausdruck `count(ancestor::fieldset)`, der die Summe aller übergeordneten `fieldset`-Elemente zurückgibt. Daraus wird schliesslich der maximale Wert berechnet.

Für jeden Knoten lässt sich zudem die aktuelle Tiefe (`rel_deepness`) angeben:

```
<xsl:variable name="rel_deepness">
  <xsl:value-of select="count(ancestor::fieldset)"/>
</xsl:variable>
```

Schliesslich lässt sich durch den folgenden Ausdruck berechnen, ob es sich beim aktuellen Element um ein Fieldset handelt (`is_fieldset`):

```
<xsl:variable name="is_fieldset">
  <xsl:value-of select="self::fieldset"/>
</xsl:variable>
```

Ein Lieferant der schliesslich ein Formular ausfüllen möchte, löst durch die Übermittlung seiner Daten, eine Aktion auf das Java-Servlet `PostForm.java` aus. Diese Daten werden zu einem neuen XML-Dokument der Struktur `postform.dtd` zusammengestellt und als Instanz des entsprechenden Formulares an den XML-Server überliefert. Dabei ist sicherzustellen, dass nur ein wohlgeformtes XML-Dokument übermittelt wird. Bereits durch XML verwendete Spezialzeichen (`< > & '' '`) müssen deshalb entsprechend verschlüsselt werden. Dies geschieht in der Funktion `value2node`:

```
public String value2node (String value, String node) {
    char[] ch = value.toCharArray();
    StringBuffer parsed = new StringBuffer();
    for (int j = 0; j < ch.length; j++) {
        if (ch[j] == '<') {
            parsed = parsed.append("&lt;");
        } else if (ch[j] == '>') {
            parsed = parsed.append("&gt;");
        } else if (ch[j] == '"') {
            parsed = parsed.append("&quot;");
        } else if (ch[j] == '&') {
            parsed = parsed.append("&amp;");
        } else if (ch[j] == '\'') {
            parsed = parsed.append("&apos;");
        } else {
            parsed = parsed.append(ch[j]);
```

```
            }
        }
        return node.concat("<option value=\""+parsed+"\"/>");
    }
```

Resultate

Die verschiedenen Komponenten dieses Modules sind in Tabelle 4 abgebildet.

ShowResult.java
showresult.xsl

Tabelle 4: Resultate

Sobald Formulardaten übermittelt worden sind, können sie über die Klasse `ShowResult.java` angesehen werden. Dabei ist sicherzustellen, dass nur ein authorisierter Benutzer, also derjenige welcher das entsprechende Formular erstellt hat, diese Daten ansehen kann. Dieses Problem wird mit Sessions gelöst. Da in den Formulardaten die Struktur des Formulares und die Feldnamen nicht gespeichert sind, werden zwei Dokumente zur Darstellung benötigt: die ursprüngliche Formularstruktur (`form.xml`) und die Formulardaten (`postform.xml`). Mit dem Stylesheet `showresult.xsl` werden diese beiden Dokumente miteinander verknüpft (Abbildung 8).

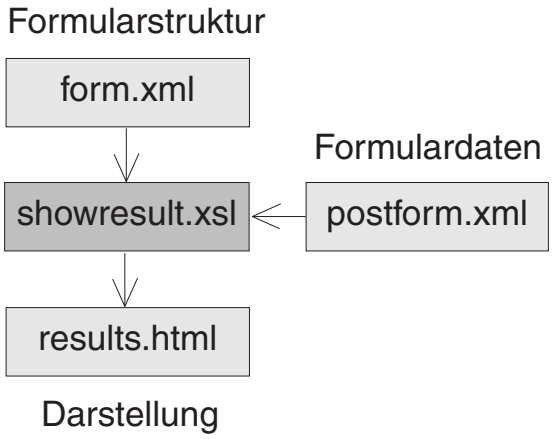

Abbildung 8: Darstellung der Resultate

Produkte

Die verschiedenen Komponenten dieses Modules sind in Tabelle 5 abgebildet.

NewProduct.java	PostProduct.java	ShowProduct.java
newproduct.xsl		showproduct.xsl

Tabelle 5: Produkte

Die Klasse `NewProduct.java` ermöglicht dem Benutzer ein neues Produkt zu definieren. Das Formular zur Produktespezifikation (`product_spec_form`) wird durch den Portalbetreiber (Administrator) zur Verfügung gestellt. Die Produktedaten werden als XML-Dokumente der Struktur `postform.dtd` gespeichert. Die Methoden, welche dazu benötigt werden, sind ähnlich wie bei der Übermittlung der Formulardaten in der Klasse `PostForm.java`. Hier wurden sie in der Klasse `PostProduct.java` implementiert. Eine Produktebschreibung kann durch jeden Benutzer durch die Klasse `ShowProduct.java` dargestellt werden. Die beiden Sichten werden durch die Stylesheets `newproduct.xsl` und `showproduct.xsl` wahrgenommen. Die Darstellung eines Produktes kann wie bei den Formularen als Frame in eine andere Webseite eingebunden werden. Es ist momentan nicht möglich, die erstellten Produktebeschreibungen mit einem bestimmten Bewerbungsformular zu verknüpfen, da in der Datenstruktur der Formulare kein Link-Datentyp implementiert wurde. Deshalb ist es die Aufgabe des Benutzers, diese beiden Dokumente gegenseitig in einen Zusammenhang zu bringen und beispielsweise in der eigenen Webseite gemeinsam als Frameset zu veröffentlichen. Dafür ist es aber auch möglich einem bestimmten Produkt mehrere verschiedene Bewerbungsformulare oder mehreren Produkten ein bestimmtes Bewerbungsformular zuzuordnen.

6.2 Administratorensicht

Die Administratorensicht wurde in einer separaten Servlet Zone implementiert und ist auf einen Pfad gemountet, der als ganzes passwortgeschützt ist. Die Servlet Zone der Administratorensicht (`portaladmin`) hat aber auf alle Javaklassen der Benutzersicht (`portal`) Zugriff.

Formularverwaltung

Die verschiedenen Komponenten dieses Modules sind in Tabelle 6 abgebildet.

`Form.java`	`Fieldset.java`	`Field.java`	`SavedForm.java`
`forms.xsl`	`fieldsets.xsl`	`fields.xsl`	`showforms.xsl`
`form.xsl`	`fieldset.xsl`	`field.xsl`	`showform.xsl`
`saveform.xsl`			
`formsmain.xsl`			
`importform.dtd`	`fieldset.dtd`	`field.dtd`	`form.dtd`

Tabelle 6: Formularverwaltung

Die Formularverwaltung bildet den Hauptbestandteil der Administratorensicht. Im Wesentlichen soll es dem Portalbetreiber ermöglicht werden, die komplexe Datenstruktur in einfachen und übersichlichen Tabellen zu verwalten. Dazu wurde die gewählte Struktur in die drei Klassen Form, Fieldset und Field aufgegliedert. Diese können somit separat verwaltet werden. Auf dem XML-Server wurden dazu drei verschiedene Datenbank Schemas definiert `importform.dtd`, `fieldset.dtd` und `field.dtd`. Diese Strukturen enthalten jeweils eine Menge von "Zeigern" (`fieldsetref, fieldref`) auf Dokumente der untergeordneten Klasse (Abbildung 9). Bei Fieldsets sind auch Zeiger auf andere Fieldsets zugelassen. Diese verteilte Struktur ermöglicht eine Mehrfachverwendung und Vererbung der einzelnen Elemente. Die Vererbung ist allerdings gefährlich.

Man muss sicherstellen, dass keine unendlichen Rekursionen produziert werden. Ein Beispiel: Eine Firma enthält als untergeordnetes Element eine Tochtergesellschaft. Diese ist wiederum eine Firma und enthält deshalb wieder eine Tochtergesellschaft usw. Zur Verhinderung von solchen unendlichen Strukturen muss eine maximale Schranke für die Baumtiefe definiert werden. Die Stylesheets `forms.xsl, fieldsets.xsl` und `fields.xsl` dienen zur Darstellung der verschiedenen Klassen. Via Hyperlinks kann durch die gesamte Formularstruktur navigiert werden. Zudem können einzelne Elemente neu erstellt oder gelöscht werden. Die entsprechenden Stylesheets `form.xsl, fieldset.xsl` und `field.xsl` erlauben es, ein einzelnes Element zu editieren. Dabei wird eine hohes Mass an Flexibilität und Automatisierung geboten. Referenzierte Knoten können aus einem Menu ausgewählt werden. Die Position eines Knotens innerhalb der Struktur kann verändert werden.

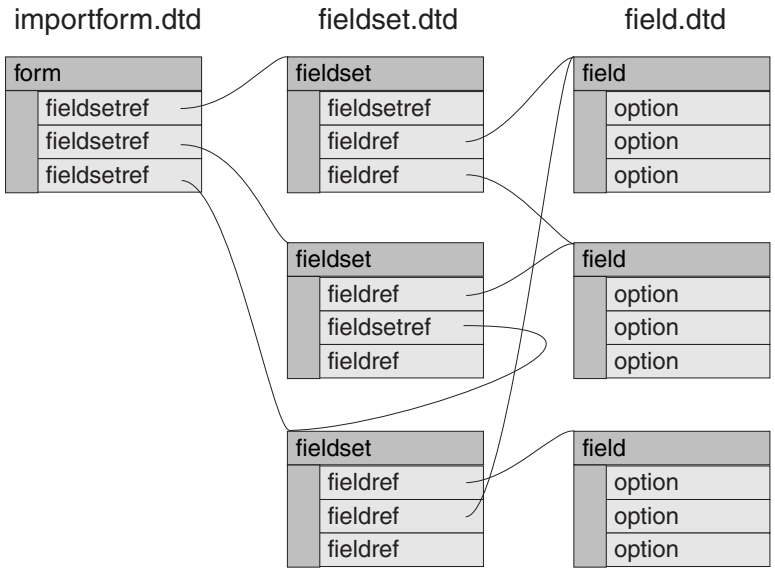

Abbildung 9: Formularverwaltung: Referenzen auf Elemente

Die Klasse `Form.java` enthält zudem eine Methode, um die gesamte Formularstruktur zu speichern. Dabei werden alle Zeiger in den einzelnen Knoten verfolgt und durch die entsprechenden Elemente und Attribute ersetzt. Für diese Transformation wird das Stylesheet `saveform.xsl` verwendet. Als Resultat wird die gesamte Formularstruktur mit mehrstufiger Hierarchie generiert. Bereits gespeicherte Formulare können mit der Klasse `SavedForm.java` dargestellt und verwaltet werden. Daneben werden auch die von den Benutzern abgeleiteten Formulare aufgelistet. Die zur Verfügung stehende Löschfunktion bewirkt, dass nicht nur das entsprechende Formular sondern auch sämtliche bereits darauf erfolgten Antworten gelöscht werden. Das Stylesheet `formsmain.xsl` enthält Templates, die von allen Stylesheets der Formularverwaltung benötigt werden. Darunter fällt beispielsweise der Header im Benutzerinterface, der den Pfad des aktuellen Knotens, das Navigationsmenu und die Sprachwahl enthält.

Sprachverwaltung

Die verschiedenen Komponenten dieses Modules sind in Tabelle 7 abgebildet.

Terms.java
terms.xsl
term.xsl
newterm.xsl
termsmain.xsl
languages.xml
term.dtd

Tabelle 7: Sprachverwaltung

Die Sprachverwaltung ermöglicht eine von der Implementation unabhängige Übersetzung der in der Benutzeroberfläche und den Kriterien verwendeten Begriffe, Ausdrücke und Fragen. Die verschiedenen Funktionen (Editieren, Suchen, Neu, Löschen) wurden in der Klasse `Terms.java` implementiert. Es können beliebig viele Sprachen verwaltet werden. Bei einer Spracherweiterung sind folgende Änderungen notwendig: Im Klassenheader von `Terms.java` muss die folgende Zeile entsprechend erweitert werden:

```
static String[] languages = {"en","de","fr","jp"};
```

Zudem muss das Dokument `languages.xml` ergänzt werden. Dabei werden an beiden Orten die standardisierten Abkürzungen verwendet. Alle übrigen nötigen Änderungen erfolgen daraufhin automatisch.

Die Funktionen, die durch die verschiedenen Stylesheets erfüllt werden, sind ähnlich wie bei der Formularverwaltung. `terms.xsl` stellt die Begriffe in einer Tabelle dar. Mit dem Stylesheet `term.xsl` kann ein einzelner Begriff editiert werden. In `termsmain.xsl` sind wiederum Templates definiert, die in den davon abgeleiteten Stylesheets verwendet werden können.

Benutzerverwaltung

Die verschiedenen Komponenten dieses Modules sind in Tabelle 8 abgebildet.

Company.java
companies.xsl
company.dtd

Tabelle 8: Benutzerverwaltung

Company.java ermöglicht die Benutzerverwaltung. Es wird sichergestellt, dass beim Löschen einer Firma alle zugehörigen Daten (inkl. erstellte Produkte, Formulare und entsprechende Antworten) gelöscht werden. Das Stylesheet `companies.xsl` stellt die verschiedenen Firmen tabellarisch dar.

6.3 Weitere Komponenten

Die weiteren Komponenten sind in Tabelle 9 abgebildet.

Transform.java	InoConnection.java
frame.xsl	
message.xsl	
unframedmessage.xsl	

Tabelle 9: Hilfsklassen

Die Klasse `Transform.java` wendet den Stylesheet Prozessor von Xalan [8] an. Damit können XML-Dokumente transformiert werden. Diese Klasse wird von den meisten Servlets verwendet. Der Aufruf erfolgt dabei folgendermassen:

```
Transform transform = new Transform(XML_DATA,XSL_DATA,parameter);
String result = transform.getDocument();
```

`XML_DATA` und `XSL_DATA` können entweder URL-Pfade oder lokale Dateipfade sein. Die Variable `parameter` ist eine Hashtable mit den Parametern, die an das entsprechende Stylesheet übergeben werden sollen. Im String `result` wird das generierte XML- oder HTML-Dokument gespeichert. Dieses Dokument kann nun z.B. an den HTTP-Client weitergeleitet werden.

In der Klasse `InoConnection.java` sind die Schnittstellen zum Tamino-Server [12] implementiert. Diese Klasse wurde 1:1 aus der Arbeit von Roger Zimmermann [14] übernommen. Im Wesentlichen vereinfacht sie den Aufruf der verschiedenen Befehle des Tamino-Servers. Am häufigsten wurden die `put`- und `delete`-Methoden verwendet. Die erste Methode löst einen `HTTP PUT`-Request aus, welcher ein XML-Dokument an den Server sendet, die zweite Methode stellt einen `URL` mit dem XQL Query-String `?_delete=<xml-schema>[...]` zusammen, welcher das entsprechende Dokument auf dem XML-Server löscht.

Dem Stylesheet `frame.xsl` kommt die Hauptbedeutung beim Benutzerinterface zu. Darin werden einerseits alle globalen Variablen, das CSS-Stylesheet und die JavaScript-Funktionen beschrieben. Des weiteren wurde hier das Layout der Rahmenstruktur in HTML implementiert. Für die drei Hauptteile, `header`, `menu` und `body` (Abbildung 10) wurde je ein Template als Platzhalter definiert, das von den davon abgeleiteten Stylesheets überschrieben werden kann. Schliesslich werden noch einige weitere Templates zur Verfügung gestellt, wie Sprachwahl `switchlang`, die Übersetzung der Begriffe (`terms`) und die Menueinträge `menuentry`.

Die beiden Stylesheets `message.xsl` und `unframedmessage.xsl` dienen schliesslich der Darstellung von verschiedenen Meldungen, sowohl eingebunden in die Rahmenoberfläche, sowie auch ohne Rahmen.

7 Resultate

Im folgenden werden nun die verschiedenen Module, die im vorhergehenden Abschnitt ausführlich erklärt wurden, nochmals zusammengefasst und die verschiedenen Sichten der Lösung durch Screenshots illustriert.

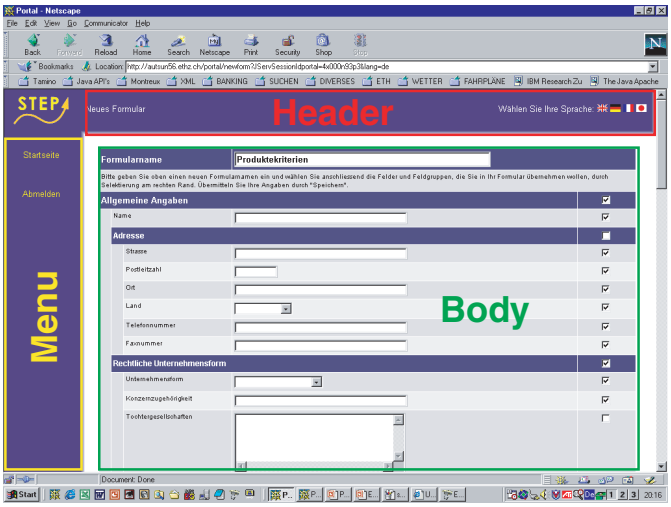

Abbildung 10: Rahmenstruktur der Benutzeroberfläche

7.1 Benutzersicht

Registrierung / Login Da nur ein authorisierter Benutzer auf die eigenen Daten Zugriff haben soll, ist ein Login mit Username und Passwort nötig. Ein neuer Benutzer muss sich dafür zunächst mit den wichtigsten Angaben zur Firma und seiner Person registrieren. Nach dem Login findet der Benutzer eine personalisierte Benutzeroberfläche vor (Abbildung 11). Diese Benutzeroberfläche bietet folgende Funktionen an:

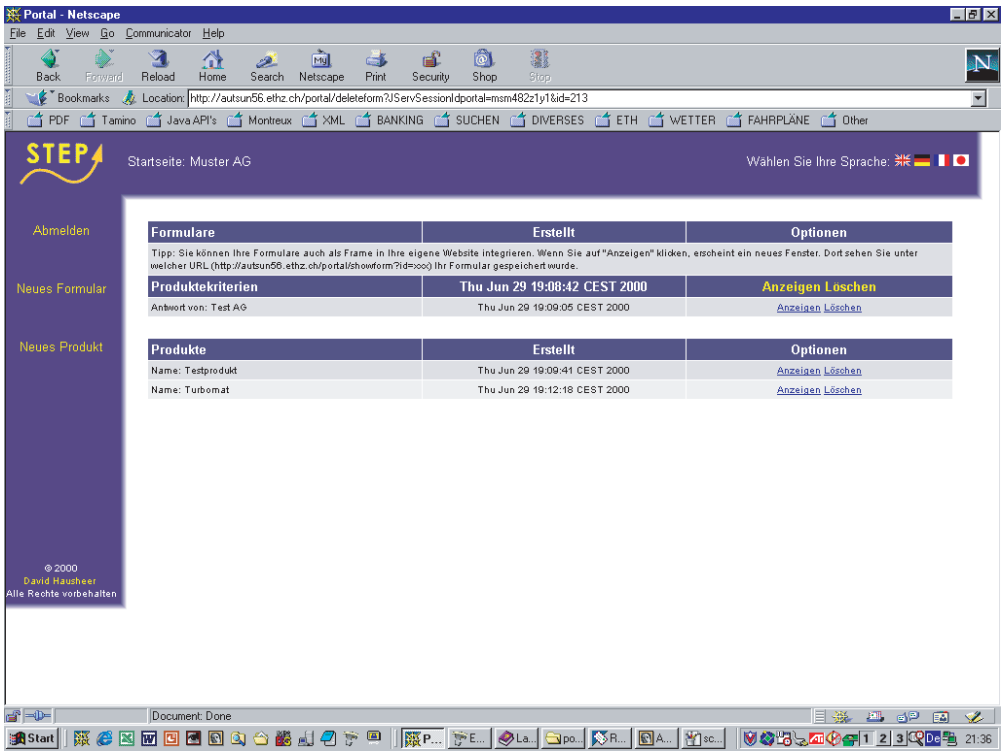

Abbildung 11: Personalisiertes Benutzerinterface

Formular erstellen Aus dem Standardformular können die gewünschten Kriterien ausgewählt und damit ein eigenes Bewerbungsformular erstellt werden (Abbildung 12).

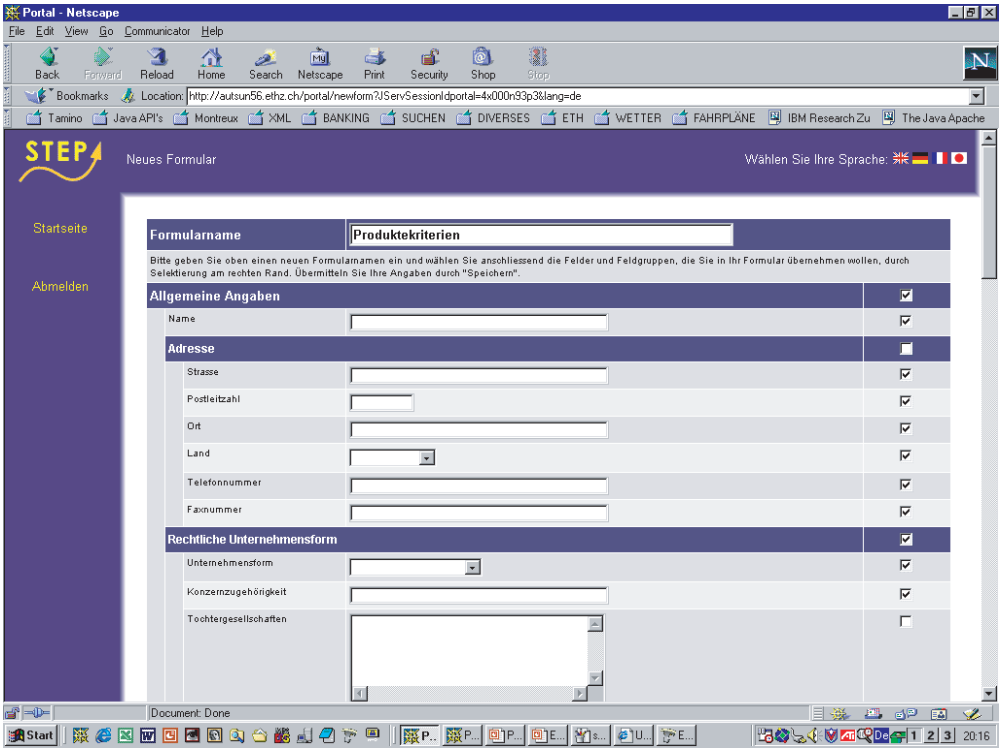

Abbildung 12: Formular erstellen

Formular ausfüllen Ein Lieferant, der sich für einen bestimmten Kunden / ein bestimmtes Produkt bewerben will, findet ein Bewerbungsformular vor, das er mit seinen Angaben ausfüllt. Diese Sicht wird in Abbildung 13 dargestellt.

Resultate vergleichen Schliesslich kann der Kunde die eingegangenen Antworten der potentiellen Lieferanten anzeigen lassen (Abbildung 14).

Produkt beschreiben Neben der Formularerstellung ist auch die Beschreibung eines neuen Produktes möglich, für das sich Lieferanten bewerben können.

7.2 Administratorensicht

Die Administratorenseite kann durch ein Navigationsmenu im Header gesteuert werden. Im folgenden werden einige Beispiele der verschiedenen Sichten gezeigt.

Formularverwaltung Wie bereits weiter oben beschrieben wurde, können die drei verschiedenen Komponenten der Formularstruktur, Forms, Fieldsets und Fields verwaltet werden. Die beiden Abbildungen 15 und 16 zeigen wie die Struktur dargestellt und editiert werden kann.

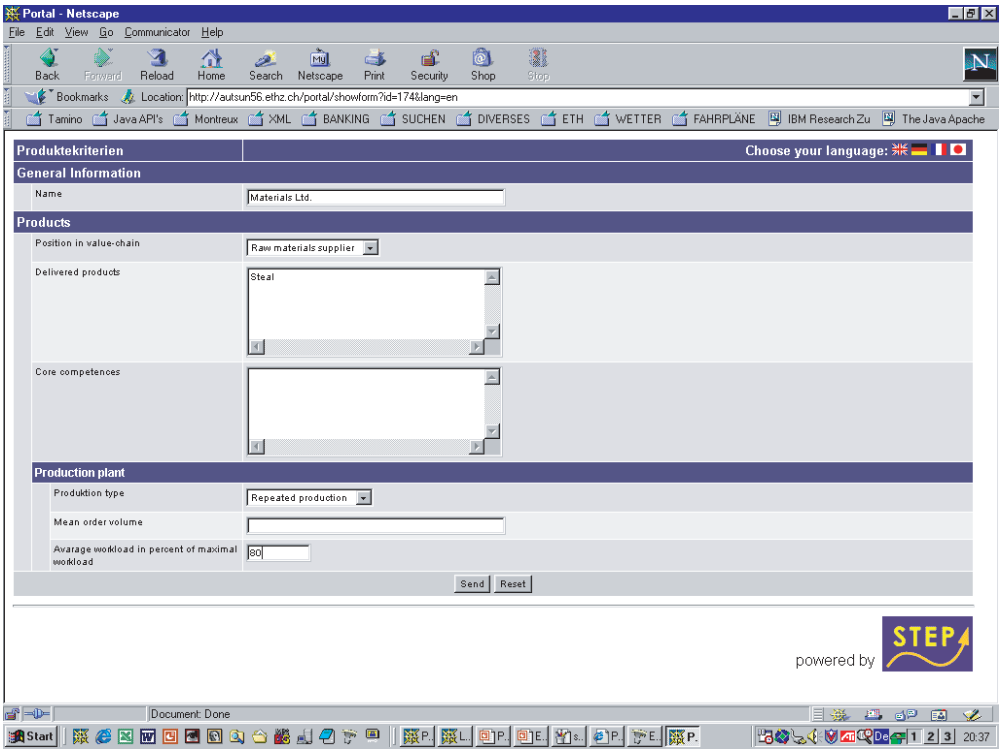

Abbildung 13: Formular ausfüllen

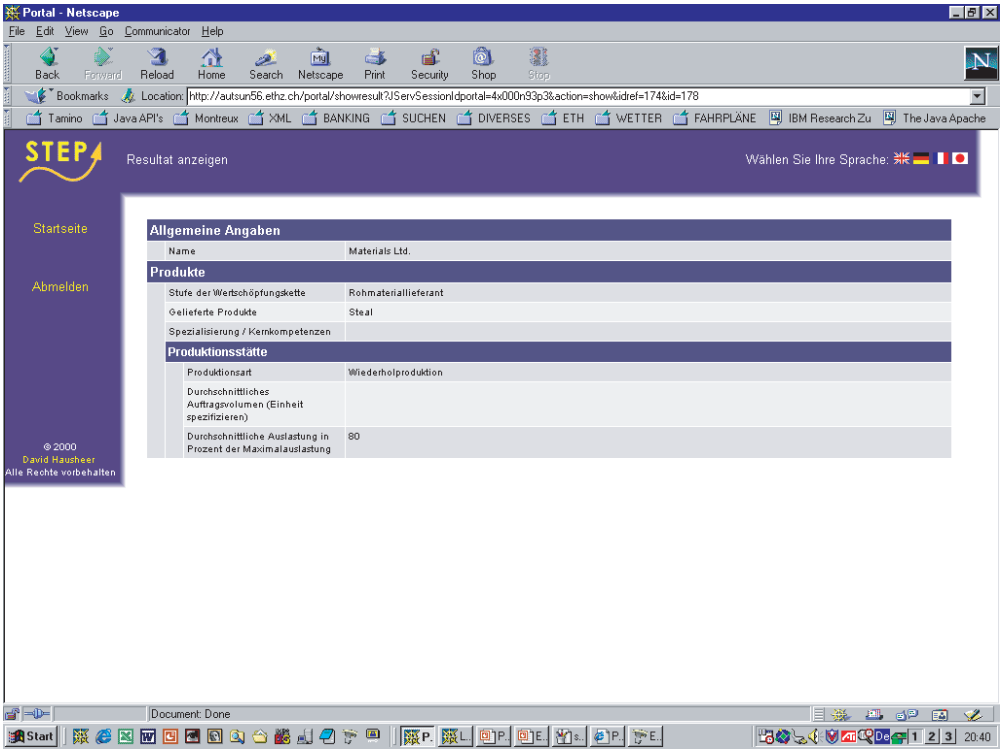

Abbildung 14: Resultate anzeigen

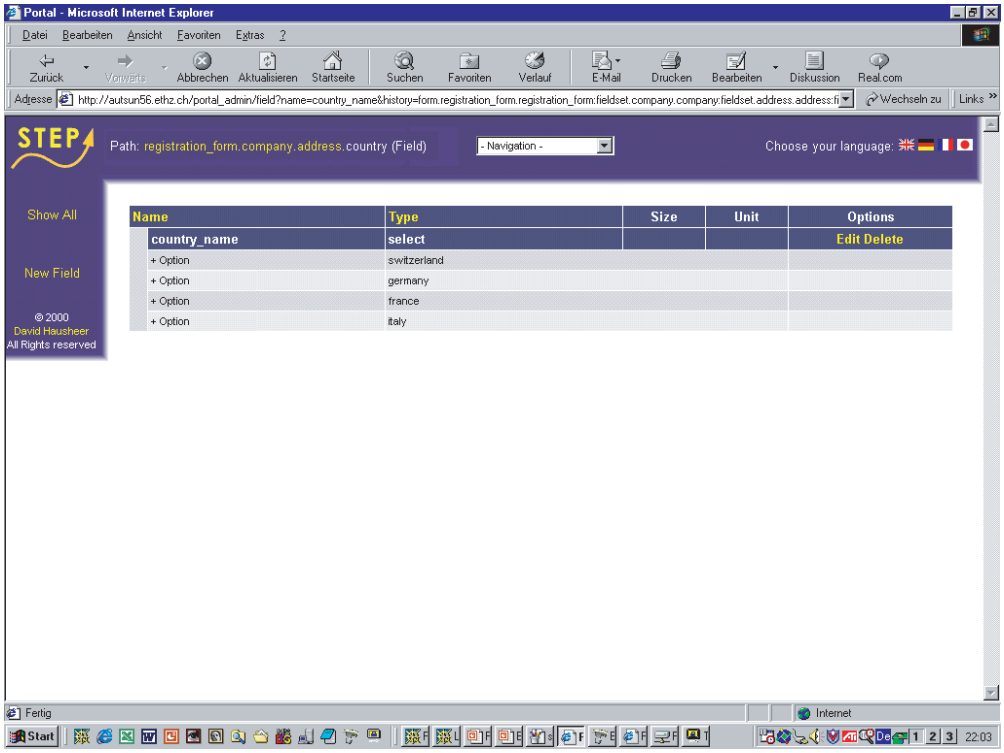

Abbildung 15: Formularverwaltung: Field anzeigen

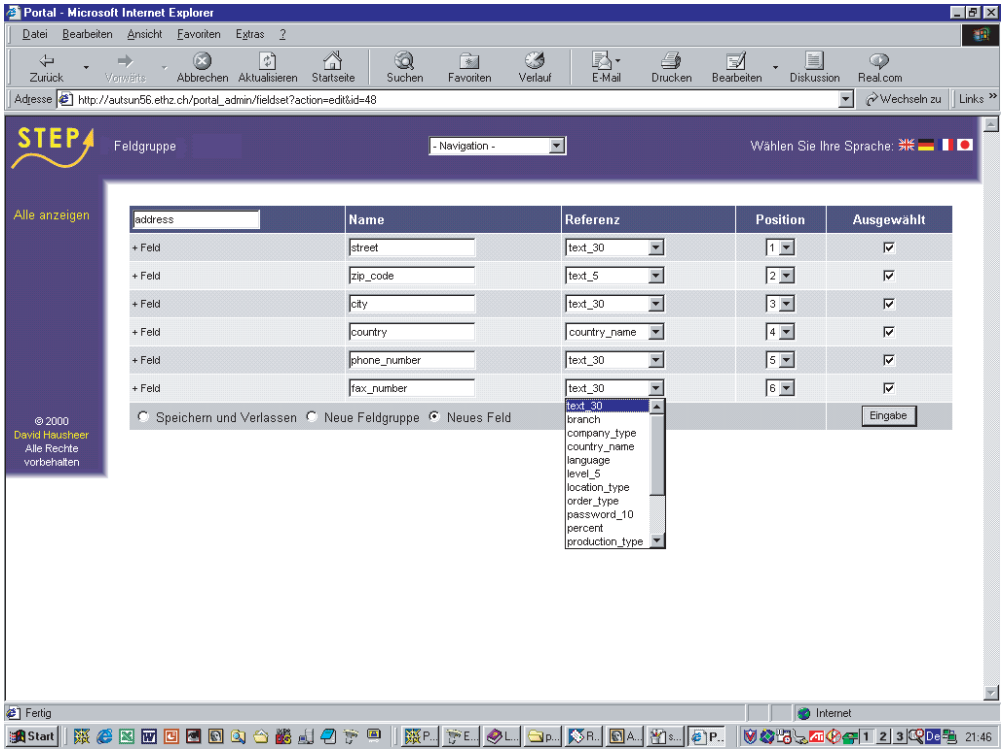

Abbildung 16: Formularverwaltung: Fieldset editieren

Benutzerverwaltung Die Benutzerverwaltung zeigt eine Übersicht der am Portal beteiligten Firmen. Falls eine bestimmte Firma gelöscht wird, so verschwinden automatisch auch deren Formulare und Produkte. Dies verhindert, dass übrigbleibende Daten den XML-Server auf längere Zeit unnötig belasten.

Sprachverwaltung Diese Sicht ermöglicht eine bequeme Verwaltung der verschiedenen Übersetzungen der im Portal und in den Formularen benutzen Begriffe (Abbildung 17).

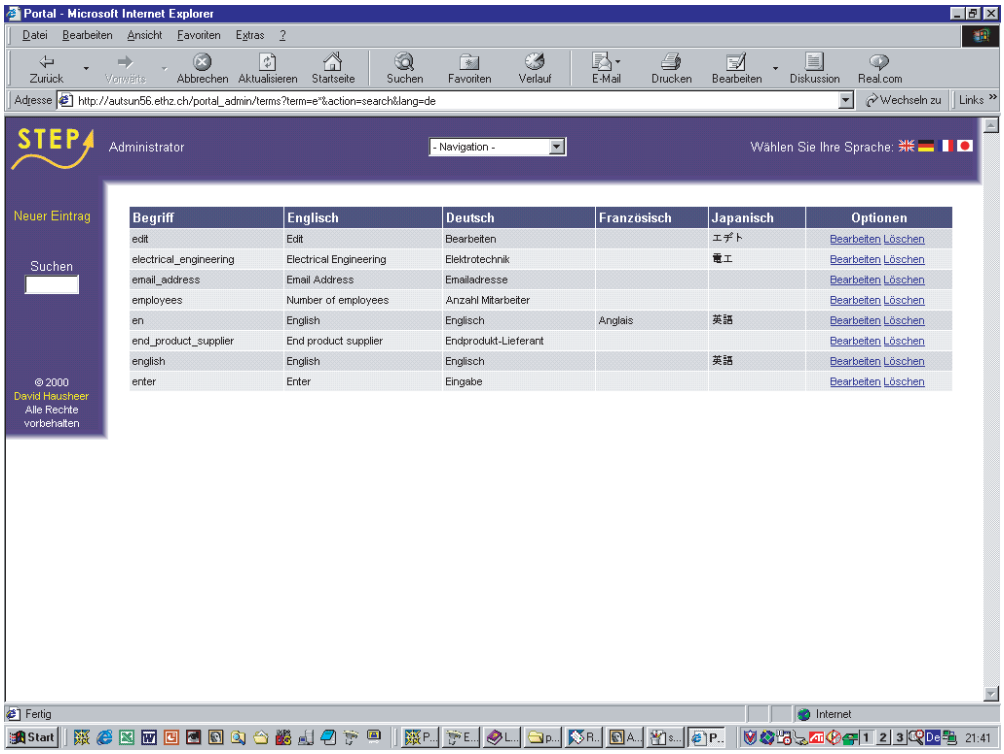

Abbildung 17: Sprachverwaltung

8 Zusammenfassung

In Rahmen dieser Arbeit wurde eine technologische Lösung entwickelt, welche den Austausch von Information bei der Strategischen Internetbeschaffung unterstützt. Es wurde eine grundlegende Datenstruktur entwickelt, die es erlaubt, dynamische, auf dem XML-Format basierende Formulare zu generieren, die ein Lieferant zur Bewerbung für ein bestimmtes Produkt oder einen bestimmten Kunden, bzw. ein Kunde für die Ausschreibung eines bestimmten Produktes verwenden kann. Diese Formulare können sowohl über das Portal erstellt, bzw. ausgefüllt werden, als auch in eine andere Webseite eingebunden werden.

Bei Umsetzung der Lösung auf die technologische Plattform wurden moderne Technologien wie XML, Stylesheets, Java-Servlets eingesetzt und deren Möglichkeiten, Vor- und Nachteile evaluiert und dokumentiert. Bei der Implementation wurde auf eine möglichst

modulare Aufbauweise und Erweiterbarkeit der Lösung grosses Gewicht gelegt. Diese Modularität ist an verschiedenen Orten sichtbar: einerseits bei der Datenstruktur durch die Trennung von Form (Formularstruktur) und Inhalt (Formulardaten), andererseits bei der Architektur in der Trennung von Applikation (Webserver) und Daten (XML-Server) und schliesslich auch in der Implementation durch die Gliederung in verschiedene Sichten und Funktionsgruppen.

Als Resultat wurde eine Benutzeroberfläche erstellt, über welche die verschiedenen Funktionen einfach ausgeführt werden können. Sie macht die Anwendung für den Benutzer transparent gegenüber der dahinterstehenden komplexen Datenstruktur und Implementierung. Dem Administrator werden Verwaltungsdienste zur Verfügung gestellt, die eine sehr bequeme Aufbereitung der Daten erlaubt.

9 Ausblick

Vor der Implementation dieser Arbeit stand ursprünglich die Vision eines Portals mit einer Kunden- und Lieferantenseite, wie in Abbildung 18 ersichtlich. Eine vollständige Entwicklung bis hin zur Marktreife dieses Produktes würde Zeit und Ressourcen benötigen, die den Rahmen dieser Studienarbeit sprengen. Der Fokus wurde deshalb in erster Linie auf die Kundenseite und die Entwicklung einer geeigneten modularen Datenstruktur gelegt. Natürlich sind verschiedene Erweiterungen dieser vorliegenden Lösung möglich und für ein zukünfiges Produkt auch unbedingt nötig.

Lieferantenseite Bisher wurde nur die Kundenseite implementiert. Zur Automatisierung der Formulareingabe durch die Lieferanten ist jedoch unbedingt eine Lieferantensicht nötig. Ein Lieferant soll seine Angaben nur einmal eingeben müssen, bis jetzt muss er dies für jeden Kunden von neuem tun. Zur Implementierung der Lieferantenseite ist lediglich eine Spiegelung der Kundenseite nötig. Die Datenstruktur kann komplett übernommen werden. Die Servlets und Stylesheets müssen entsprechend angepasst werden.

Matching Nachdem beide Seiten vollständig implementiert wurden, ist es möglich, mit den eingegebenen Daten ein intelligentes Matching zu betreiben. Lieferanten sollen automatisch die zu ihrem Profil passenden Kunden zugespielt erhalten. Kunden sollen eine mögliche Auswahl von Lieferanten, die auf ihre Produktebeschreibung passen, erhalten. Dabei soll das Matching für beide Seiten transparent und nach den modernsten wissenschaftlichen Erkenntnissen erfolgen.

Interaktion, Verfeinerung Nachdem das Profil eines Lieferanten mit der Produktebeschreibung eines Kunden in einer ersten Runde gematch hat, sind weitere Verfeinerungsschritte und überhaupt die Interaktion der beteiligten Parteien wünschenswert. Dieser Ablauf muss durch das Portal ebenfalls gesteuert werden können. Denkbar ist auch eine Anonymisierung der Teilnehmer bis zur ersten Kontaktnahme. Dabei können natürlich auch rechtliche und andere Probleme entstehen. Auf diese wird in der Arbeit von Raimundo Sierra [13] näher eingegangen.

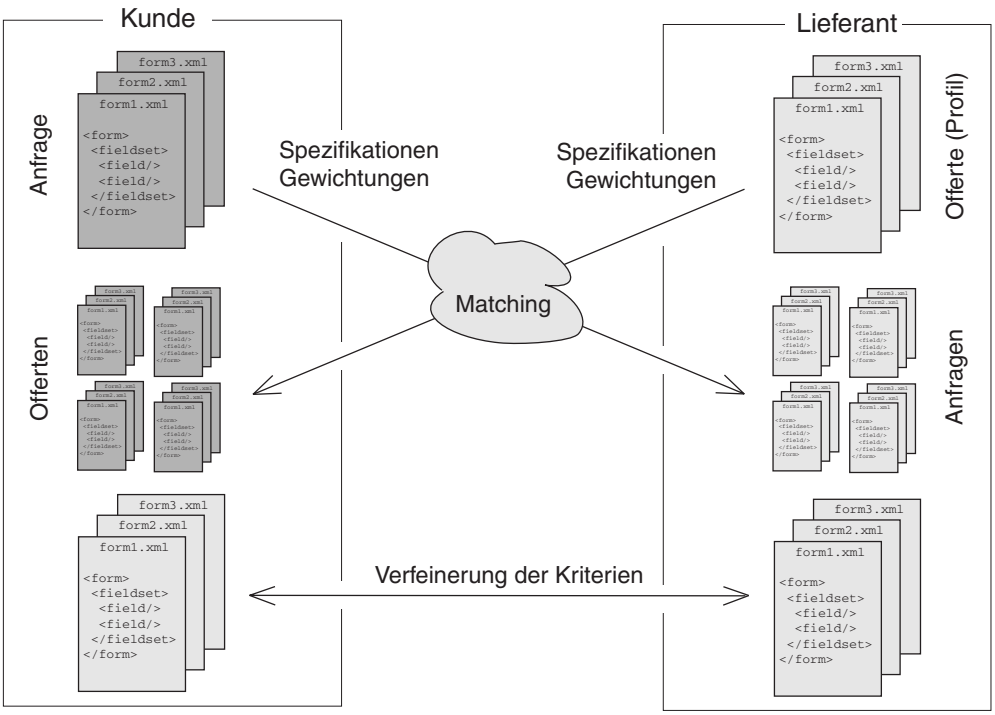

Abbildung 18: Portal-Struktur

Literatur

[1] World Wide Web Consortium. *Extensible Markup Language (XML) 1.0.* W3C Recommendation. http://www.w3.org/TR/1998/REC-xml-19980210

[2] World Wide Web Consortium. *XML Path Language.* W3C Recommendation. http://www.w3.org/TR/xpath

[3] World Wide Web Consortium. *HTML 4.0 specification.* W3C Recommendation. http://www.w3.org/TR/REC-html40

[4] World Wide Web Consortium. *Extensible Stylesheet Language (XSL).* W3C Working Draft. http://www.w3.org/TR/WD-xsl

[5] World Wide Web Consortium. *XSL Transformations (XSLT).* W3C Recommendation. http://www.w3.org/TR/xslt

[6] J. Robie, J. Lapp, D. Schach. *XML Query Language (XQL).* http://www.w3.org/TandS/QL/QL98/pp/xql.html

[7] E. Wilde. *Wilde's WWW.* Technical Foundations of the World Wide Web. http://www.wildesweb.com/

[8] The XML Apache Project. *Xalan.* Implementation of XSLT and XPath. http://xml.apache.org/

[9] The Java Apache Project. *Apache JServ.* Java Server for Apache. http://java.apache.org/

[10] Sun Microsystems. *Java Servlet Specification.* Dokumentation. http://java.sun.com/

[11] Sun Microsystems. *The Java Servlet Development Kit.* Implementation of Java Servlets. http://java.sun.com/products/java-server/servlets/

[12] Software AG. *Tamino XML-Server 1.2.1.* Dokumentation.

[13] R. Sierra. *Strategische Internetbeschaffung.* Studienarbeit. TIK. Juli 2000.

[14] R. Zimmermann. *XML-Wörterbuch.* Studienarbeit. BWI. April 2000.

Sourcecode

Auf der beigelegten CD sind folgende Daten enthalten:

Verzeichnis	**Beschreibung**
`/portal`	Servlet Zone mit Java-Klassen und Stylesheets für die Benutzersicht
`/portaladmin`	Servlet Zone mit Java-Klassen für die Administratorensicht
`/portal_src`	Bilder und andere Webressourcen
`/conf`	Konfigurationsdateien für den Web-Server
`/docs`	Bericht, Screenshots und Präsentationen
`/source`	Diverse Programme die bei der Implementation verwendet wurden
`/dtd`	Datenbank Schemas

Abbildungsverzeichnis

Tabellenverzeichnis

Aufgabenstellung