

Programmiersprache zur Erstellung und Spezifikation von Netzwerkdiensten

Markus Pöstinger
Universität zu Lübeck, Fachbereich Informatik
Email: markus.poestinger@informatik.uni-luebeck.de

0) Zusammenfassung

Diese Arbeit ist die Ausarbeitung zu einem Seminarvortrag und umfasst einen sehr kurzen Umriss über das Thema XL (XML Language). XL ist eine Programmiersprache zur Erstellung und Spezifikation von Netzwerkdiensten, die unter anderem an der Universität München entwickelt wurde. Mittlerweile ist das Projekt an die Universität Heidelberg gewechselt.

Diese Arbeit gliedert sich in mehrere Kapitel: Zuerst wird eine Definition eines Netzwerkdienstes präsentiert und ein Überblick geschaffen, was es momentan für Sprachen gibt, um Dienste dieser Art zu spezifizieren. Danach werden die Probleme erläutert, die dabei entstehen und die Bedingungen entwickelt, die eine neue Sprache erfüllen sollte. Schließlich wird ein kurzer Einblick in XL gegeben, vor allem in Beziehung auf seine verwendeten Standards, die Syntax und seine Möglichkeiten. Schlußendlich wird noch ein Beispiel präsentiert und ein kurzer Blick auf die Architektur von XL geworfen.

1) Einführung

1.1) Netzwerkdienste

Es gibt keine standardisierte Definition vom W3C [1] und auch im Internet findet man nur sehr wenig in dieser Beziehung. Darum sei hier nur eine Definition angegeben, die in der Wikipedia [2] zu finden ist:

“Ein Netzwerkdienst ist eine abstrahierte Funktion, die von einem Computernetzwerk den [...] teilnehmenden Geräten bereitgestellt wird. Der Fokus liegt dabei darauf, dass ein Dienst eine in sich geschlossene Funktionskomponente aus Anwendersicht darstellt, diese Funktion kann technisch über eines oder auch mehrere Netzwerkprotokolle der Anwendungsschicht realisiert werden. Netzwerkdiensten ist gemeinsam, dass vorhandene Ressourcen gemeinsam genutzt werden. [...]”

Zusammengefasst ist ein Netzwerkdienst ein Programm, das einen Server bereitstellt, der einzelne Funktionen und Operationen des Dienstes für Clients mittels Protokollen bereitstellt, wobei dieses Programm auf eine Anzahl von Ressourcen zugreifen kann.

Als Beispiel wird korrekt der Dienst World Wide Web angeführt, den die Rechner bereitstellen, die sich zum Internet zusammengeschlossen haben. Verwendete Protokolle sind hierbei zum Beispiel HTTP (HyperText Transfer Protocol), SMTP (Simple Mail Transfer Protocol) oder FTP (File Transfer Protocol).

1.2) Existierende Sprachen zur Spezifikation von Netzwerkdiensten

Klassische Dienste werden in den letzten Jahren häufig in php (Php – Hypertext Preprocessor) [3],

in Java [4], in Perl [5] oder in Visual Basic [6] geschrieben. Darüber hinaus besteht meist eine Anbindung an eine Art eines SQL-basierten relationalen Datenbanksystems (häufig MySQL [7]), um möglichst einfach auf eine große Anzahl von Daten zuzugreifen oder sie in der Datenbank abzulegen. Dies gestaltet sich allerdings nie so einfach, wie es eigentlich gewünscht ist.

Zum einen sind die Typsysteme von relationalen Datenbanken mit denjenigen von objektorientierten Sprachen wie Java oder (neuerdings) php inkompatibel. Will man ein Objekt aus beispielsweise Java in eine Datenbank übertragen, muß es zuerst für die relationale Tabellenstruktur dieser konvertiert werden. Ebenso muß eine Konvertierung stattfinden, wenn das Objekt aus der Datenbank wieder zur Verwendung in Java übertragen werden soll. Diese Konvertierung kostet natürlich Rechenzeit.

Es wird dann noch komplizierter, wenn darüberhinaus Daten in XML-Struktur vorliegen; beispielsweise wenn eine Kommunikation mit einem anderen Dienst notwendig ist und XML als Austauschformat verwendet wird. Auch hier müssen die Daten erst konvertiert werden; im schlimmsten Fall sogar doppelt, wenn die XML-Daten zuerst in ein Java-Objekt verwandelt werden müssen und dann in die relationale Struktur konvertiert werden, um sie in die Datenbank einzufügen. Optimal wäre hier ein Typsystem für Kommunikation, Datenbank und Programmiersprache.

Zum anderen findet häufig eine Vermischung von Applikationslogik und Datenüberprüfungen mit der Implementation von Low-Level-Protokollen und Leistungsverbesserungen in derselben Programmiersprache statt. Das bedeutet extreme Unübersichtlichkeit des Codes und führt häufig zu einer Verkomplizierung der einzelnen Funktionen des Dienstes. Optimal wäre hier eine Sprache, die zahlreiche Funktionen, die in vielen Netzwerkdiensten vorhanden sind, von selbst unterstützt, ohne daß sich ein Programmierer um eine Implementierung bemühen müßte.

1.3) **Vorraussetzungen für eine neue Sprache**

Man kann nun schon einige Vorraussetzungen definieren, die eine neue Sprache für Netzwerkdienste erfüllen müßte.

- Eine Vielzahl wiederkehrender Probleme sollte automatisch durch die Sprache gelöst werden.
- Die Implementation soll sich auf nur ein Typsystem und ein Datenmodell stützen, nämlich XML in Kombination mit einer XQuery-Schnittstelle [8].
- Eine neue Programmiersprache sollte mit den W3C-Standards konform gehen.
- Die Programmiersprache sollte möglichst deklarativ sein; es sollte beschrieben werden, **was** gewünscht ist und nicht **wie** es funktionieren soll.

2) **XL**

2.1) **Einführung**

XL bedeutet ausgeschrieben "XML Language" und wurde von Daniela Florescu (damals XQRL Inc., heute bei bea.com [9]), Andreas Grünhagen (damals Tech. Universität München, heute Universität Heidelberg) und Donald Kossmann (damals Tech. Universität München, heute Universität Heidelberg) entwickelt. Es gibt eine Projekt-Seite zu XL unter [10] und eine Demonstration eines laufenden XL-Systems anhand eines Auktions-Skripts unter [11].

Die Entwickler behaupten, daß XL alle obigen Voraussetzungen erfüllt und noch weitaus mehr leistet. Die Sprache erfüllt die W3C-Standards und fußt auf dem Typsystem XML. Außerdem ist

XL in weitem Maße deklarativ und high level, was bedeutet, daß die Sprache schon eine Anzahl wiederkehrender Funktionen bei zahlreichen Netzwerkdiensten über die Angabe von Schlüsselwörtern unterstützt.

XL unterstützt direkt folgende Funktionen:

- Logging: Die Eingaben der Clients werden optional gespeichert
- Error-Handling: Einfache Methoden, um mit Exceptions umzugehen
- Repetition von fehlerhaften Aktionen: Bei Fehlschlag einer Anfrage Neuversuch
- Workload Management: Verteilung auf mehrere Systeme
- Events: Variablenüberwachung (bei Wert X von Variable Z mache Y)
- Performance Tuning (Caching): Automatische Optimierung des Codes

Einige dieser Funktionen sind über sogenannte “Declarative Web Service Clauses” steuerbar, die im Kopf der Spezifikation des Netzwerkdienstes angegeben werden können.

XL fußt auf XML als einzigem Datenmodell und nutzt die Anfragesprache Xquery [12], um mit der Datenbank zu kommunizieren. Zur Kommunikation mit anderen Netzwerkdiensten wird das Nachrichtenprotokoll Simple Object Access Protocol (SOAP) [13] verwendet, mit dem man sehr einfach XML-Objekte zwischen Programmen austauschen kann. Zuletzt wird die Web Service Description Language (WSDL) [14] zur Spezifikation des Interfaces benutzt.

2.2) Syntax

2.2.1) Definition des Netzwerkdienstes

```
service <uri>  
  < Functions & Local Operations >  
  < Local Declarations >  
  < Declarative Web Service Clauses >  
  < Public Operation Specifications >  
endservice
```

Mittels des reservierten Wortes “service” wird der Dienst definiert. Dahinter steht unbedingt die eindeutige URI (Unified Resource Identifier) des Dienstes, über den dieser erreichbar ist.

Danach folgt die Definition und Implementierung lokaler Funktionen und Operationen, die nur vom Server selbst aufgerufen werden können. Dann ist Raum, um lokale Variablen zu deklarieren und die oben angesprochenen Schlüsselwörter bzw. “Declarative Web Service Clauses” anzugeben, mittels denen das Dienstverhalten beeinflusst werden kann. Zuletzt werden die öffentlichen Funktionen und Operationen definiert, die die Clients aufrufen können.

2.2.2) Variablentypen

Es gibt zwei Typen von Variablen, die verwendet werden können: Zum einen Variablen, die XML-Objekte umfassen und nur einen Wert enthalten können. Zum anderen die kontextbezogenen Variablen, die man sich in der Form eines Arrays visualisieren kann, das über die URIs der Clients indiziert ist. Diese Variablen enthalten die Werte, die für die Kommunikation mit den Clients wichtig sind, beispielsweise eine Session-ID oder die Usernamen der eingeloggten User in einem System.

```

service <uri>
    let [<type>] <name> [:= <expression>];
    context let [<type>]<name> [:= <expression>];
endservice

```

Nötig ist immer nur die Angabe eines Variablennamens. Optional kann sie davor noch als ein bestimmter XML-Typ deklariert werden, der z.B. über ein Schema definiert wird, und dahinter durch eine XQuery-Expression oder direkt durch XML-Code initialisiert werden.

2.2.3) Statements

Es gibt einige grundlegende Statements zur Manipulation von XML-Objekten. So kann man recht einfach ein Tag mit einem bestimmten Inhalt in ein Objekt einfügen, herauslösen oder ersetzen. Außerdem kann man ein Tag direkt umbenennen oder innerhalb des Objektes verschieben.

```

insert <tag>Content</tag> into $object
delete $object/tag[type="subtag"]
replace $object/tag with <tag>Other Content</tag>
rename $object/tag as "tag2"
move $object/tag after $object/tag2

```

2.2.4) Dienstaufrufe

Externe Dienstaufrufe sind in XL über das Verschicken von SOAP-Nachrichten an eine bestimmte URI möglich. Die Aufrufe können sowohl im synchronen als auch asynchronen Modus gestartet werden:

```

<expression> --> <uri>[::<operation>] [--><variable>]
<expression> ==> <uri>[::<operation>] [--><operation>]

```

Im ersten, dem synchronen, Modus wartet XL mit der weiteren Ausführung des Programms bis eine Antwort des anderen Systems empfangen wurde. Im asynchronen Modus läuft XL weiter; sollte eine Antwort empfangen werden, kümmert sich optional eine Funktion über die Verarbeitung der empfangenen Nachricht.

2.2.5) Möglichkeiten zur Kombination von Statements

In XL gibt es mehrere Möglichkeiten, Statements zu kombinieren. Neben der bekannten sequentiellen Abarbeitung der Statements, wobei die einzelnen Befehle durch Semikolons getrennt werden, gibt es vier andere Typen.

Bei der Fehlerbehandlung (<statement1> ? <statement2>) wird das zweite Statement nur dann ausgeführt, wenn das erste zu einem Fehler führte. Bei der Auswahl (<statement1> | <statement2>) wird zwischen zwei oder mehreren Statements eine nichtdeterministische Auswahl getroffen. Dies ist beispielsweise bei der Verteilung von Serverlast nützlich. Bei der Parallelisierung (<statement1> || <statement2>) werden die Statements parallel ausgeführt und bei der Abhängigkeit (<statement1> & <statement2>) wird Statement1 nach Statement2 ausgeführt, wenn es von Statement2 abhängig ist. Andernfalls wird Statement2 nach Statement1 ausgeführt.

3) Beispiel

Es wird nun ein kurzes Beispiel präsentiert, in dem ein Dienst mit einer Funktion zum Login und zum Wareneinkauf realisiert wurde. Wir beginnen mit der Definition des Dienstes und den Deklarationen der internen Variablen.

```
service http://www.shop.de
  namespace xf = "http://www.w3.org/2001/08/xquery-operators"

  let $lagerDB           := <lager></lager>;
  let $kundenDB         := <kunden></kunden>;
  context let $kunde     := <kunde></kunde>;

  !! ...

endservice
```

Zuerst wird die eindeutige URI gesetzt, unter der der Dienst erreichbar ist. Danach wird ein Namespace gesetzt, unter dem die XQuery-Funktionen aufrufbar sind. Hiernach wird das Lager des Shops mit einem leeren Inhalt initialisiert, genauso wie die Kundendatenbank. Schlußendlich wird noch eine kontextbezogene Variable initialisiert, über die später alle eingeloggtten Kunden erreichbar sind.

```
  context let $kunde     := <kunde></kunde>;

  history;
  defaultoperation unknownOP;
  conversationpattern mandatory;
  conversationtimeout xf:duration("P10D") logout;
  on error invoke logout;
```

Nun werden einige Schlüsselwörter angegeben, über die das Dienstverhalten näher beschrieben wird, die sogenannten "*Declarative Web Service Clauses*".

Mit dem ersten Wort `history` wird gefordert, daß die komplette Kommunikation zwischen Server und Clients aufgezeichnet und gespeichert wird. Danach folgt die Angabe, welche Funktion aufgerufen werden soll, wenn der Client keine explizite Funktion zum Aufruf angegeben hat; in diesem Fall `unknownOP`.

Mit der dritten Angabe `conversationpattern` wird die Kommunikation zwischen Server und Client näher definiert. Das Schlüsselwort `mandatory` bedeutet, daß der Server eine Fehlermeldung ausgibt, wenn sich der Client noch in keiner Konversation mit dem Server befindet. Es gibt noch weitere Konversationsarten wie zum Beispiel `required`, in der eine neue Konversation gestartet wird, wenn zwischen Server und Client noch keine vorhanden war.

Mit den letzten Angaben wird der Timeout einer Konversation, also der Zeitraum, in dem eine Konversation noch am Leben erhalten wird, wenn keine Kommunikation mehr stattfindet, auf zehn Tage gesetzt. Danach soll die Funktion `logout` aufgerufen werden. Dies soll ebenfalls geschehen, wenn ein Fehler irgendeiner Art passiert.

Es folgt nun die Deklaration und Implementation der Funktion `login`. Wir beginnen mit dem Header der Funktion:

```
operation login
```

```
precondition xf:exists(  
  for    $k in $input/kunde, $db in $kundenDB/kunde  
  where  $k/id eq $db/id and $k/password eq $db/password  
  return $k  
);
```

```
conversationpattern requiredNew;
```

```
endoperation
```

Man kann für jede Funktion eine beliebige Anzahl an Vor- und Nachbedingungen angeben. Die Funktion ihrerseits wird nur dann ausgeführt, wenn die Vorbedingung als true evaluiert und ist nur dann korrekt abgeschlossen, wenn die Nachbedingung als true evaluiert.

In diesem Fall wird überprüft, ob die am Client eingegebene KundenID und das eingegebene Passwort in dieser Kombination mit einem Eintrag in der Datenbank übereinstimmt. Es wird jeder Eintrag der Datenbank durchlaufen, und sollte ein entsprechender gefunden werden, dieser zurückgegeben. Erst dann wird der Body der Funktion ausgeführt, den wir uns gleich anschauen.

Außerdem besitzt die Funktion ein eigenes `conversationpattern`. In diesem Fall ist es `requiredNew`, was dazu führt, daß in jedem Fall eine neue Konversation zwischen Server und Client gestartet wird, wenn diese Funktion aufgerufen wird und die Vorbedingung erfüllt ist.

```
body  
  let $kunde := <kunde>  
          <id>$input/kunde/id</id>  
          <bestellungen></bestellungen>  
        </kunde>;  
  let $output := 'Login erfolgreich!'  
endbody
```

Es folgt der Body der Funktion. Hier geschieht nicht viel mehr, als daß die Eingaben am Client in ein XML-Objekt eingefügt werden: Die KundenID wird zusammen mit einer leeren Menge an Bestellungen in ein Kunden-Objekt eingefügt. Danach wird eine Nachricht an den Client geschickt mit der Meldung, daß der Loginvorgang erfolgreich abgeschlossen ist.

Nun die nähere Betrachtung der Funktion für den Wareneinkauf:

```
operation kaufen
```

```
precondition for $p in $lagerDB/produkte,  
              $b in $input/bestellung  
  where $p/id eq $b/id  
  return $p/verfuegbar gt $b/anzahl;
```

```
body  
  insert $input/bestellung into $kunde/bestellungen;  
endbody  
endoperation
```

Auch hier gibt es eine Vorbedingung: Es wird für jedes bestellte Produkt überprüft, ob die bestellte Menge im Lager verfügbar ist. Erst wenn alle Produkte verfügbar sind, wird der Body der Funktion ausgeführt und die eingegebenen Bestellungen am Client in die Bestell-Liste im Kunden-Objekt

übertragen.

Im letzten Teil des Beispiels passiert nicht mehr viel:

```
operation unknownOP
  body
    nothing
  endbody
endoperation
```

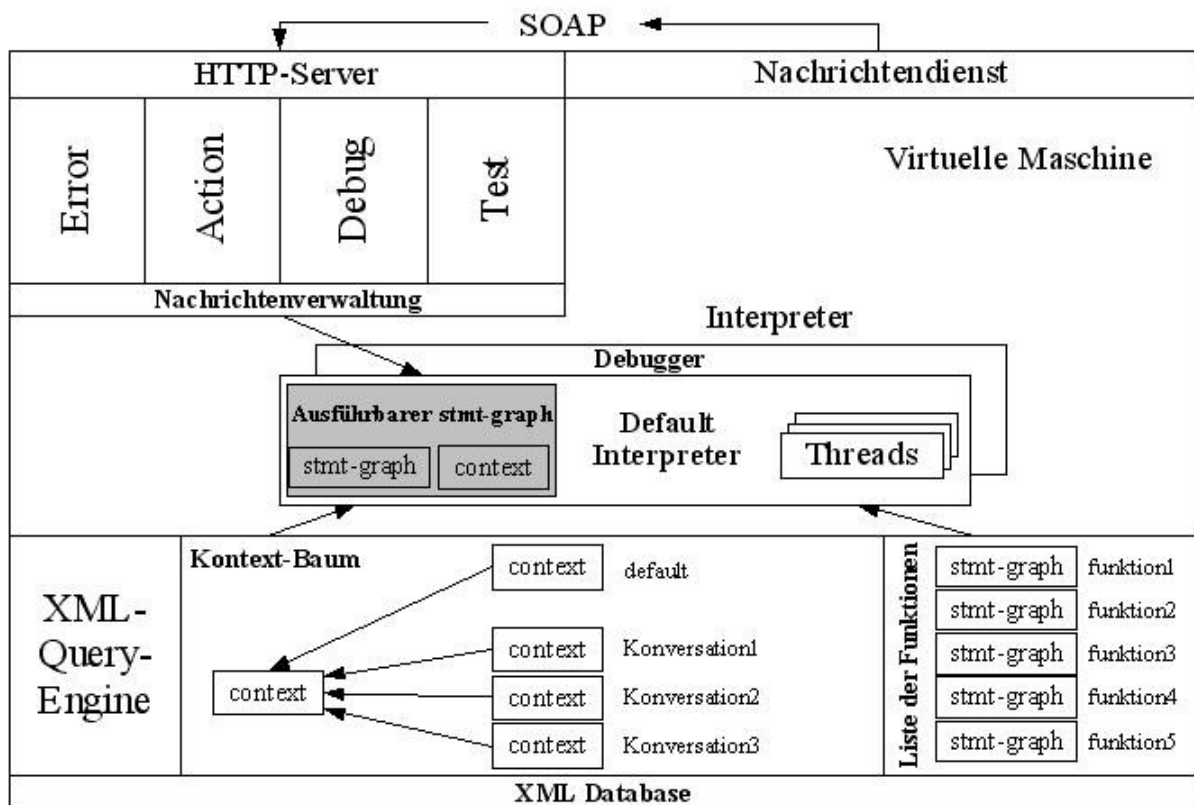
```
operation logout
  body
    nothing
  endbody
endoperation
```

endservice

Hier werden die noch nötigen Funktionen für den Logout und die oben angegebene unknownOP noch deklariert, aber nicht mehr implementiert, und schließlich die Dienstspezifikationen beendet.

4) Architektur

Die Architektur von XL besteht im Wesentlichen aus einer virtuellen Maschine, auf der verschiedene Interpreter laufen und die einen http-Server und einen Nachrichtendienst zur Kommunikation mit der Außenwelt besitzt:



Eine Anfrage an den Netzwerkdienst kann über eine Nachricht im SOAP-Format geschehen. Diese Nachricht umfasst neben der Anfrage noch die eindeutige Konversations-URI des Clients, sofern sich dieser in einer Konversation mit dem Server befindet, sowie den Typ der Anfrage.

Eine konventionelle Anfrage an den Dienst gelangt über den http-Server zur Nachrichtenverwaltung, die die Nachricht einem bestimmten Interpreter zuweist. Handelt es sich um eine Debugging-Anfrage, wird der Debugger gestartet, andernfalls der Default Interpreter.

Der Interpreter besitzt eine Anzahl von Threads, in denen die Anfragen der Clients ausgeführt werden. Außerdem wird jede Ausführung einer Funktion vom Interpreter überwacht. Auf die Funktionen hat der Interpreter in Form von sogenannten Statement-Graphen Zugriff. Bei den Graphen handelt es sich um eine umgewandelte Form des Programmcodes, wobei die einzelnen Statements die Knoten der Graphen darstellen. Der zu einem Zeitpunkt aktive Knoten markiert das momentan ausgeführte Statement. Die Kanten, die vom Knoten weg führen, sind mit true oder false markiert. Ist die Ausführung des Statements erfolgreich, werden die Statements aufgerufen, deren Kanten mit true markiert sind; ist die Ausführung nicht erfolgreich, werden die Statements aufgerufen, deren Kanten mit false markiert sind.

Bei einem Funktionsaufruf seitens eines Clients wird nun zuerst die `$input`-Variable mit allen Eingaben des Clients initialisiert, dann die Ausführung gestartet, gewartet bis die Ausführung beendet ist und schließlich das Ergebnis mittels der `$output`-Variablen zurück gesandt.

Auf die Variablen hat der Interpreter über einen sogenannten Kontext-Baum Zugriff. In dieser Baumstruktur sind sowohl die globalen Variablen (oberste Ebene), als auch die Variablen, die nur innerhalb einzelner Funktionen verwendet werden, sowie die kontextbezogenen Variablen (untere Ebenen) gespeichert.

Zuletzt hat der Interpreter noch Zugriff auf die XML-Datenbank über eine Xquery-Schnittstelle.

5) Fazit

Das Ziel der Entwickler von XL ist es nicht, eine marktreife Sprache zu präsentieren, die sich innerhalb der nächsten Jahre durchsetzt. Vielmehr ist XL als Universitätsprojekt zu sehen. In diesem Zusammenhang reden Grünhagen und Kossmann auf der Webseite des Projektes auch davon, daß sie hoffen, daß die Sprache einige Fans gewinnt und daß vielleicht Entwickler anderer Sprachen mit dem Ziel freier Markt einige Ideen aus XL für ihr Werk übernehmen.

XL macht insgesamt einen sehr guten Eindruck. Dadurch daß man sich nur noch auf ein Typsystem stützt, können in der Tat Umformungen vermieden und Rechenzeit gespart werden. Die Sprache wirkt sehr intuitiv und leicht erlernbar; allerdings kann man bisher keinen genauen Blick auf die vorläufige Implementation werfen, da der Source-Code nicht einsehbar ist.

Die Ideen zur Vorbeugung vor der Neuerfindung des Rades bei jedem weiteren Netzwerkdienst, zum Beispiel im Zusammenhang mit dem Logging aller Eingaben, sind lohnenswert. Allerdings müßte noch überprüft werden, wie flexibel die Möglichkeiten der realisierten Lösungen sind. Sollten auch nur kleine Details fehlen, kommt man vermutlich auch hier wieder nicht um Eigenimplementationen herum.

6) Literatur

- Florescu, Grünhagen, Kossmann – XL: An XML Programming Language for Web Service Specification and Composition (1, short)
- Florescu, Grünhagen, Kossmann – XL: An XML Programming Language for Web Service Specification and Composition (2, long)
- Florescu, Grünhagen, Kossmann – XL: A Platform for Web Services

- [1] – www.w3c.org – Webseite des World Wide Web Consortiums
- [2] – www.wikipedia.de – OpenSource-Enzyklopädie
- [3] – www.php.net – php-Webseite
- [4] – java.sun.com – Java-Webseite
- [5] – www.perl.com – Perl-Webseite
- [6] – msdn.microsoft.com/vbasic/ - VisualBasic-Webseite
- [7] – www.mysql.org – MySQL-Webseite
- [8] – www.xquery.com – Xquery-Webseite
- [9] – www.bea.com – Webseite der Firma bea.com
- [10] – sunbayer72.informatik.tu-muenchen.de:8080/xl/auction.jsp – Erster Prototyp von XL in Java
- [11] – xl.in.tum.de – Webseite von XL (inzwischen Redirect zur Universität Heidelberg)