Technische Universität München

Forschungs- und Lehreinheit III
Prof. R. Bayer Ph.D.

Hauptseminar „Web Services“ im Sommersemester 2002

Thema: Sprachen I – Water und JWIG

Referent: Andreas Töllich, andreas@toellich.net
Betreuer: Dipl. Inf. Michael Bauer, bauermi@in.tum.de
Abgabedatum: 09.06.2003
Vortragsdatum: 12.06.2003

Inhaltsverzeichnis:

1. Einleitung

Über kein anderes Thema wurde in der letzten Zeit so viel geschrieben, diskutiert und gestritten wie über Web Services. Das Konzept der Web Services scheint das Gegenteil von dem zu erreichen, was sie eigentlich beabsichtigen: Statt Integration zu schaffen, gehen die Meinungen über das Was und Wie von Web Services weit auseinander. Über eine klare Definition von Web Services konnte sich die Industrie bis heute nicht einigen, selbst über die Schreibweise ist man sich nicht einig.

Keiner der großen Softwaregiganten möchte den Zug aber verpassen und so sind Web Services fest in das Produktportfolio von Sun, Bea, IBM und Co. verankert. Selbst Microsoft spürte den Hauch der Zeit – die Plattform .NET basiert fest auf XML, SOAP und den anderen Technologien, auf denen auch Web Services basieren.
Web Services versprechen vor allem:

- universelle Interaktion von Anwendungen (z.B. Service Agenten);
- bessere Integration und Verknüpfung von geschäftsübergreifenden Anwendungen;
- Einsparung von Kosten bei der Anwendungsentwicklung durch kürzere Entwicklungszyklen.

Die unterschiedlichsten Gerätetypen, von gewöhnlichen PCs und Servern über Handhelds bis zu intelligenten, eingebetteten Systemen sollen nahtlos miteinander kommunizieren können. So wie HTML und HTTP es den Menschen erlauben, in die Welt der ungegrenzten Information des Internets einzutauchen, sollen Web Services die nächste Evolutionsstufe des Internets darstellen. In diesem service-basierten Internet soll die Maschine-zu-Maschine Kommunikation ebenso wie die Entwicklung von service-basierten Anwendungen Realität werden.

So viel versprechend diese Aussichten für Entscheider, Entwickler und Anwender sind, so steinig ist der Weg, Anwendungen auf Basis von Web Services zu erstellen. Web Services sind jetzt schon seit gut zwei Jahren in aller Munde, bis heute wurden aber wenig brauchbare Web Service Anwendungen entworfen, die über die Funktionalität von Euro-Rechner, Aktienkurs-Anfrage und dem obligatorischen „Hello World“ hinausgehen. Auch im Bereich der Enterpreise Application Integration (EAI) erfreuen sich nach wie vor Technologien wie CORBA und Middleware Applikationen wie Tuxedo von IBM größter Beliebtheit. Dies mag daran liegen, dass einfach Erfahrungswerte beim Entwurf von Web Services fehlen und Entscheider zögerlich sind wenn es darum geht, Investitionen zu tätigen, die auf einer Technologie aufbauen, deren Entwicklung und Standardisierung

nicht abzusehen ist. Nicht zuletzt wird man als Entwickler von Web Services mit vielen (im Folgenden ein Auszug etwas salopp formulierter) Fragen konfrontiert:

- Welche Programmiersprache soll eingesetzt werden? Java, C# oder doch Perl?
- Welche Werkzeuge werden benötigt?
- Welcher XML-Parser soll eingesetzt werden? Welcher SOAP Client und welche Server Software?
- Welche Komponenten arbeiten optimal zusammen, welche nicht?
- Warum denn noch ein paar von XML abgeleitete Sprachen lernen?

Genau an dieser Stelle wollen zwei Produkte weiterhelfen, die im Folgenden vorgestellt werden:
Zum einen *Water* [1], eine Programmiersprache auf Basis von XML, deren primärer Einsatzzweck die Entwicklung von Web Services und –Anwendungen ist.
Zum anderen *JWIG* [2], einer Erweiterung der Programmiersprache Java, um schnell und komfortabel Web Services und –Anwendungen zu entwickeln.

2. Water

2.1 Überblick

Water ist eine Programmiersprache für Web Anwendungen und Web Services, die in der Syntax der Extensible Markup Language (XML) notiert wird. Ziel dieser Sprache ist es, den Entwurf von Web Services stark zu vereinfachen. XML wird in vielen Bereichen wie der Modellierung von Daten, für Remote Procedure Calls (RPC), der Definition von Interfaces und zur Präsentation eingesetzt. Water erweitert XML, um Präsentationsschicht, Programmlogik und Daten zu vereinen. Es basiert auf den Ideen einer Vielzahl von Sprachen wie Scheme, Self, Lisp, Java, SmallTalk, JavaScript, HTML und natürlich XML.

Water wurde von Mike Plusch und Christofer Fry entworfen - zwei Studenten, die sich während ihres Studiums am Massachusetts Institute of Technology intensiv mit XML beschäftigten. Jene begannen 1998 ihre Arbeit an der Programmiersprache – sie sind auch die Mitgründer des Startups Clear Methods Inc. in Cambridge [3], welche *Steam*, eine Web Service Plattform basierend auf Water kommerziell vertreibt. Im Januar 1998 entstand die Version 1.0. Derzeit ist die Version 3.0 aktuell.

Will man Web Services mit einer Plattform wie J2EE oder .NET entwickeln, bedarf es vieler Technologien und Sprachen wie XML, HTML, SQL, XSLT, JSP, ASP.NET, SOAP,

JavaServlets, JavaScript etc. Im Gegensatz dazu verfolgt Water die Strategie "Learn Once, Use Everywhere" – Water soll viele dieser sehr spezifischen Sprachen überflüssig machen und so Entwicklungskosten einsparen.

2.2 Architektur

Werfen wir einen Blick auf typische n-Schichten Architekturen; für jede Schicht wird eine andere Sprache benötigt. Die Client-Schicht benötigt zum Beispiel HTML und Java Script, der Server (Business-Schicht) benutzt JSP und Java. JSP ist für die Client-Schicht aber nicht interpretierbar. JSP muss HTML generieren. Water dagegen kann in jeder Schicht benutzt werden, da die Laufzeitumgebung für Water für jede Schicht verfügbar ist. Für den Web Browser als Java-Applet, für Web- und Application Server existiert ein stand-alone Server, der aber auch als Servlet in einem existierenden JavaServlet Container wie zum Beispiel Apache Tomcat arbeiten kann.

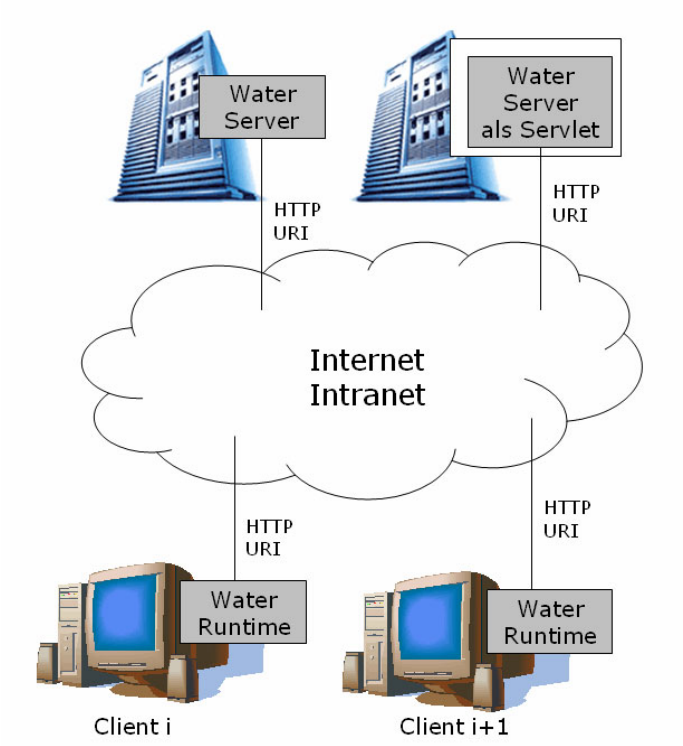

Abbildung 1: Architektur von Web Anwendungen mit Water

Einer der geistigen Väter von Water schreibt in einem seiner Paper [Fry2002], dass all die bekannten Sprachen und Technologien nicht flexibel genug sind, um Entwicklern all die Funktionalität an die Hand zu geben, die sie benötigen, um moderne Internet basierte Anwendungen zu entwickeln. Es fragt sich, warum es für jede Schicht einer neuen Sprache oder Technologie bedarf. In der Tat muss sich der Entwickler bei der Erstellung von Web Services beispielsweise mit Java mit den unterschiedlichsten APIs vertraut machen, die die Brücke zw. der algorithmischen Programmiersprache (in diesem Falle Java) und der Daten-Sprache (XML, XSLT, SOAP, WSDL, UDDI etc.) schlagen. Das Web

Services Developer Pack (Early Access Release 1) von Sun [5] enthält allein vier APIs und zahlreiche Tools, mit denen sich Entwickler von Web Services mit Java vertraut machen müssen: Dazu gehören Java API for XML Messaging (JAXM), Java API for XML Processing (JAXP), Java API for XML Registries (JAXR), Java API for XML-based RPC (JAX-RPC), JavaServer Pages, Standart Tag Library (JSTL), Ant Build Tool, Java WSDP Registry Server, Tomcat Java Servlet & JavaServer Pages container. Neue Standards wie die Java Architecture for XML Binding (JAXB), JavaServer Faces und eine neues JavaServer Pages Expression Language werden bald verabschiedet.

Als großer Vorteil von Water wird konstatiert, dass keine solcher APIs zwischen der algorithmischen Sprache (dem eigentlichen Quellcode der Anwendung) und der Daten-Sprache (den XML Derivaten) benötigt werden, da diese in Water eng miteinander verwoben sind: Statische und dynamische Daten, Algorithmen und Präsentation.

Eine gemeinsame Sprache über die einzelnen Schichten hinweg, vereinfacht die Entwicklung von Web Services und Web Anwendungen. Aus der Sicht des Entwicklers bietet Water eine einheitliche Methode, Programmlogik, Daten und deren Präsentation in jeder Schicht zu verarbeiten. Ein Funktionsaufruf stellt sich einheitlich in jeder Schicht dar. Lokale Funktionsaufrufe gleichen Remote Procedure Calls über HTTP. Ein Funktionsaufruf eines HTML-Formulars gleicht einem Funktionsaufruf zwischen zwei Servern. Auch eine Email Nachricht kann als Funktionsaufruf agieren.
Die Architektur von Water ähnelt stark des REST Architektur [Fielding2000], wie in 2.6.1 beschrieben.

2.3 Water und XML

Water ist eine Programmiersprache für das Internet. Sie vereint die drei Hauptfunktionalitäten, die für die Präsentation und Manipulation von Informationen benötigt werden, in einer einzigen Sprache. Dazu zählen Programmcode, Daten und Markup.

Die Syntax von Water ist eine Obermenge von HTML, aber auch eine Obermenge von XML. HTML ist eine Auszeichnungssprache für Seiten im World Wide Web. XML ist der Standard, wenn es darum geht, strukturierte Daten im Web bereitzustellen. Water erweitert nun die Syntax von XML dahingehend, dass damit Anwendungen im Sinne einer Programmiersprache entworfen werden können. Diese erweiterte Syntax wird als *ConciseXML* bezeichnet. Hierbei ist es aber wichtig, dass man XML aus einer bestimmten Sicht sieht: Der datenorientierten Sicht. Die XML-Gemeinschaft ist nämlich zweigeteilt: Da wäre zum einen das „Dokumenten“-Lager, zum anderen das „Daten“-Lager. „Die

dokumentenorientierte Sicht von XML suggeriert, dass es sich bei einem XML-Dokument um eine kommentierte Textdatei handelt, die Markup-Direktiven zur Kontrolle des Formats und der Darstellung des enthaltenen Textes enthält. Für die datenorientierte Sicht ist XML lediglich eine von vielen Repräsentationen eines typbehafteten Wertes, die von Software-Agenten zum Datenaustausch und für Zwischenoperationen benutzt werden“ [Box01]. Die Verfechter des ersten Lagers sind der Ansicht, dass XML von Werkzeugen wie emacs oder teueren XML-Authoringwerkzeugen erzeugt wird. Die Anhänger der datenorientierten Sicht betrachten XML dagegen als reines oder weiteres Speicherformat – als Syntax für Daten und Objekte – nicht als Syntax für Dokumente.

2.4 ConciseXML

In der Fachliteratur spricht man von XML (1) als einen Standard, um Daten in einer Form zu speichern, die von Maschinen verstanden wird, (2) als die Lösung für all die Integrationsprobleme von Anwendungen, (3) als eine Programmiersprache. Die geistigen Väter von Water sind der Meinung, dass keiner der drei genannten Punkte auf XML zutrifft. Ihre These: XML ist reine Syntax. Das scheint logisch und konsequent: Der XML 1.0 Standard definiert keine Tags oder Attribute, die Semantik mit sich tragen. XML beschreibt auch nicht, wie Objekte und Daten abgebildet werden. Viele Standards wie SOAP, WSDL, UDDI, SVG (Scalable Vector Graphics), XHTML und ebXML („XML für E-Business“) basieren auf der Syntax von XML, um Daten zu beschreiben. Das Problem dabei ist, dass jeder dieser Standards Daten auf andere Weise abbildet. Folglich ist der Austausch von Daten zwischen diesen Standards problematisch und mit viel Aufwand verbunden.

Will man Daten in XML abbilden, steht man vor einem Problem, über welches die Fachwelt heftig diskutiert und welches bis heute ungelöst ist. Es geht um die Frage, wann Elemente und wann Attribute zur Modellierung von Daten benutzt werden. Dies ist vor allem vor dem Hintergrund interessant, dass Water eine objekt-orientierte Programmiersprache ist und folglich Objekte in XML bzw. ConciseXML dargestellt werden müssen. Ein Objekt besitzt Felder, diese Felder tragen Werte, die wiederum Objekte sind. Zu Verdeutlichung der Ambiguität wird im Folgenden ein Java Objekt der Klasse „Student“ in XML modelliert um die Ambiguität von Elementen und Attributen zu zeigen:

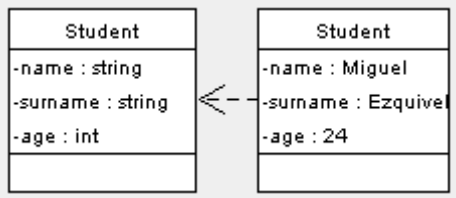

Abbildung 2: Java Klasse Student mit Instanz

Möglichkeit 1

```
<Student>
   <name>Miguel</name>
   <surname>Ezquivel<surname>
   <age>24</age>
</Student>
```

Möglichkeit 2

```
<Student name="Miguel" surname="Ezquivel" age="24">
```

Listing 1: Ambiguität von XML

In Water ist ein Objekt eine Kollektion von Feldern. Jedes Feld hat einen Schlüssel und einen Wert. Der Wert kann wiederum ein Objekt sein. Dementsprechend entspricht die *Möglichkeit 2* unseres Beispiels der Modellierung von Objekten in ConciseXML. Objekte werden ausschließlich auf diese Weise modelliert und so die Ambiguität von XML vermieden.

Bisher haben wir gesehen, wie mit Hilfe von ConciseXML Objekte – dem zentralen Thema der Objekt Orientierung – modelliert und abgebildet werden. Mit ConciseXML soll aber nicht nur die diesbezügliche Ambiguität des XML 1.0 Standards eliminiert werden; Mit Hilfe von ConciseXML sollen dynamische Anwendungen entwickelt werden. Werfen wir also einen Blick auf die wesentlichen Merkmale von Water als Programmiersprache.

2.5 Sprachmerkmale von Water

2.5.1 Ausdrücke

Um mit ConciseXML dynamische Anwendungen zu entwickeln, bedarf es neben den Objekten auch Operatoren, boolschen Ausdrücken und Kontrollstrukturen. Diese werden ebenfalls in der Tag Darstellung notiert. Eine Kurzreferenz findet man unter [4].

Verglichen mit anderen Programmiersprachen bietet Water einen eingeschränkten Satz an Strukturen, um den Ablauf einer Anwendung zu steuern. ConciseXML bietet hierfür lediglich *if-Anweisungen* und *for-each Iteratoren*. Anweisungsblöcke können mir *do* gekennzeichnet werden.

```
<if> cond1      action1
     cond2      <do>
                 action2a
                 action2b

                </>
     else       action3
</if>
```

Listing 2: if-Anweisung

An Boolschen Ausdrücken unterstützt ConciseXML die boolschen Konstanten *true* und *false*, sowie die Operatoren *and*, *or* und *not*. Mit *is* und *equal* können auch Objekte und deren Felder verglichen werden.

```
<thing foo=5/>.<equal <thing foo=5/> />

      Rückgabe: true
```

Listing 3: Vergleich von Objekten

Eine wichtige Rolle kommt den Operatoren *get*, *set* und *remove* zu. Mit diesen können Felder Werte zurückgegeben, zugewiesen und entfernt werden. Der Ausdruck *has* prüft, ob ein Objekt ein bestimmtes Feld enthält.
Nehmen wir an, wir haben ein Objekt *Student* mit dem Feld *name*. Mit Hilfe von *get Feldname* erhalten wir den Wert des Feldes.

```
<student name=”Miguel” age="24"/>.<get "name"/>

      Rückgabe: "Miguel"
```

Listing 4: Rückgabe via get-Operator

2.5.2 Klassen und Objekte

Alles in Water ist ein Objekt. Objekte bilden die Grundlage von Water. Es gibt primitive Objekte wie Nummern und Zeichenketten. Zudem können Objekte beliebig verknüpft werden um komplexe, zusammengesetzte Datenstrukturen zu schaffen. Eingebaute zusammengesetzte Datenstrukturen sind zum Beispiel Vektoren und hash tables. Dies wird dadurch ermöglicht, dass Objekte Felder haben, die Schlüssel und Werte besitzen, wobei die Werte wiederum Objekte sein können. Zwischen den zusammengesetzten Datenstrukturen gibt es ebenfalls nur minimale Unterschiede. Vektoren und hash tables werden auf dieselbe Art repräsentiert – als ein Objekt mit einer Menge von Feldern, die wiederum einfache und komplexe Objekte tragen. Wenn der Entwickler ein Objekt als Vektor behandelt, iteriert ein Loop-Mechanismus über alle Felder. Ein Vektor (als vordefinierter zusammengesetzter Datentyp) besitzt ein internes Feld „length“, der die Länge des Vektors beinhaltet.

Water besitzt einen generischen Datentyp *thing*. Wird dieser instantiiert, erhält man ein Objekt, das standardmäßig keine Felder enthält, diese aber beliebig hinzugefügt werden können.

In Water gibt es zwar Klassen und Instanzen – mit einer interessanten Verschränkung: Jedes Objekt ist entweder Instanz einer Klasse, kann aber auch selbst die Rolle einer Klasse spielen. Eine Klasse oder ein Objekt ist somit der Prototyp für andere Objekte (*Prototyp System*). Eine Klasse beinhaltet Daten und Methoden die von allen Instanzen gemeinsam genutzt werden. Standardmäßig kennt eine Klasse ihre Instanzen nicht.

Objekte können von anderen Objekten abgeleitet werden, wobei sie bestimmte Eigenschaften ihrer Eltern erben, andere dagegen werden modifiziert, um Objekte zu erzeugen, die sich von ihren Elternobjekten unterscheiden (*Mehrfachvererbung*).

Das Objekt System von Water erlaubt es, dass zur Laufzeit das Elternobjekt eines Objekts geändert wird. Mit dieser Möglichkeit der *Laufzeitvererbung* können sozusagen Eltern(objekte) ein Kind(objekt) „adoptieren“.

Interessant ist nicht zuletzt die Tatsache, dass in Water XHTML Tags als Objekte behandelt werden. Water beinhaltet nämlich eine eingebaute Klasse *Hypertext*.

2.5.3 Variablen und Methoden

Auch zwischen Instanzvariablen und Methoden wird in Water nicht unterschieden. Beide werden als Werte von Feldern in einem Objekt gespeichert. Man erhält den Wert einer Instanzvariablen oder den Rückgabewert einer Methode, indem man einfach den Namen des zugehörigen Feldes aufruft. Methoden werden wie alle anderen Dinge in Water durch Objekte implementiert. Die Syntax eines Methodenaufrufes unterscheidet sich nicht von der Syntax eines HTML-Tags.

Im folgenden Beispiel wird die in UML modellierte Klasse *Student* (Abbildung 2) in Water definiert (Listing 5), eine Instanz a_Student erstellt und der Wert der Variablen *name* und das Ergebnis der Methode *show()* zurückgegeben. Die Methode birthday() inkrementiert das Alter (Listing 6).

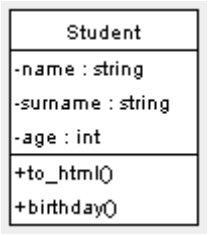

Abbildung 3 : Klasse Student in UML

```
<defclass Student name=string surname=string age=int>

      <defmethod show>
            <H1><do .name/></H1>
      </>
      <defmethod birthday>
            <do .age.<plus 1/>/>
      </>

</defclass>
```

Listing 5 : Klassendefinition

```
<set a_Student =
<Student name="Miguel" surname="Ezquivel" age=24 />
/>
```

Listing 6: Instantiierung

```
A_Student.name
```
Rückgabe: “Miguel”

```
A_Student.show
```
Rückgabe: “Miguel” als HTML Headline1

```
A_Student.birthday
```
Aufruf der Methode birthday(), die age inkrementiert

Listing 7: Rückgabewerte von Variablen und Methoden

2.5.4 Methodenaufrufe und Objekt-Erstellung

Wie bereits erwähnt werden in den Feldern von Objekten, Methoden und wiederum Objekte (einfache und zusammengesetzte) gespeichert. Ein XML-Tag bezieht sich immer auf das Eltern Objekt. Die Parameter nach dem Tag-Namen dienen entweder als Methoden Argumente oder als Feld-Initialisierer, abhängig davon, ob der Wert eines Tags eine Methode oder ein Objekt ist.

2.5.5 Objekte und Typen

Es gibt keinen Unterschied zwischen Objekten und Typen. Zum Beispiel kann ein String als Typ behandelt werden; in Wirklichkeit ist ein String ein Objekt der Objekthierarchie, welches als Elternobjekt aller Strings fungiert. Alle String Objekte gleichzeitig können über Pfadangaben referenziert werden, genauso wie alle Felder von Objekten damit

referenziert werden können. Diese Pfadangaben sind nichts anderes als XPath ähnliche Ausdrücke.

2.5.6 Anmerkungen

In Water bekommen Anmerkungen eine ganz neue Bedeutung. Water bietet die Möglichkeit Informationen über Felder hinzuzufügen. Zur Erläuterung: Nehmen wir an, wir haben eine Feld „Farbe“ in einem Objekt „Auto“. Es könnte sein, dass eine Anmerkung dazu gemacht werden soll, z.B. „alle Farben außer schwarz und weiß kosten extra“. Auch constraints à la „das Feld ‚Verkaufspreis’ darf nur Ganzzahlen beinhalten“. Ein Loop-Mechanismus sorgt dafür, dass alle Anmerkungen zu einem bestimmten Feld gefunden werden.

2.5.7 Auswertungen

Water bietet Entwicklern die Möglichkeit selbst zu bestimmen, wie bestimmte Parameter eines Methodenaufrufes interpretiert werden. Zum einen kann ein Parameter als Quellcode behandelt und zur Laufzeit ausgewertet werden (vgl. Tcl, das in der Linux Welt weit verbreitet ist). Es ist auch möglich, dass der Ausdruck lediglich geparst, aber nicht ausgewertet wird. Zuletzt kann ein Parameter auch als Hypertext interpretiert werden. Hier werden Zeichenfolgen innerhalb der Winkelklammern (<>) als Quellcode interpretiert, der ausgewertet werden soll und andere Zeichenfolgen als Strings. Dies ist vor dem Hintergrund wichtig, dass in Water HTML Tags als Water-Objekte oder Funktionsaufrufe behandelt werden.

2.6 Water und Web Services

Die Idee hinter Water, die Entwicklung von Web Services zu vereinfachen, besteht darin, dass die gängigen Technologien und Sprachen dafür, in einer einzigen Programmiersprache zu vereinen. Zu den genannten Technologien und Sprachen zählen in erster Linie:

- SOAP (Simple Object Access Protocol)
- WSDL (Web Service Description Language)
- UDDI (Universal Description, Discovery and Integration)
- XPATH (XML Path Language)
- XSLT (XSL Transformations)

Da all die „Sprachen“ in XML Syntax notiert werden, folgt Water ebenfalls dieser Syntax um die Integration zu erleichtern. Water bietet aber in seinem Sprachschatz keine Objekte und Methoden an, um direkt mit dieses Standards zu arbeiten, sondern stellt vielmehr Objekt und Methoden in einer einheitlichen Weise bereit, um diese

Funktionalitäten „nachzuahmen“. Für jeden der fünf genannten Standards gibt es also eine propriertäre Antwort von Water; die Ziele (Datenaustausch, Datentransformation, Finden und Binden von Ressourcen etc.) bleiben die gleichen.

2.6.1 Water und REST

Obwohl nicht von ihr beeinflusst, entsprechen Philosophie und Charakteristika denen der REST Architektur. REST steht für *Representational State Transfer*, einen Begriff den Roy Fielding in seiner Dissertation “Architectural Styles and the Design of Network-based Software Architectures” [Fielding2000] prägte. REST ist ein Modell für verteilte Anwendungen. Es stützt sich auf der größten verteilten Anwendung der Welt: dem World Wide Web. REST geht davon aus, dass das Web aus einer kleinen Menge von Methoden (HTTP Methoden wie GET und POST) und aus einer potentiell unendlich großen Anzahl von Ressourcen (URI) besteht. Unabhängig davon, welcher Web Server und welcher Web Browser eingesetzt wird, können aus einer URL wie z.B. „http://www3.in.tum.de/thema_webservices/webforum“, ohne weitere Unterstützung oder Wissen, Informationen mittels dieser HTTP Methoden gewonnen werden. Auf Web Services bezogen, basiert REST auf dem Trio XML, URI und HTTP. Wesentliche Charakteristika von Water, die der REST Architektur [Fry2002], [Fielding2000] gleichen sind:

- HTTP und SMTP als Standard-Kommunikationsprotokolle;
- URI und URL um auf Ressourcen zuzugreifen;
- XML, XHTML, GIF, JPEG als Repräsentationsmöglichkeit von Ressourcen;
- Zustandslosigkeit: kein Verbindungskontext muss auf einem Applikation Server gespeichert werden;
- Einheitliches Interface: der Zugriff auf alle Arten von Ressourcen geschieht über ein generisches Interface;
- Verbundene Ressourcen: URIs verbinden vielfache Ressourcen;
- Layered components: *intermediaries*, wie Proxy Server, Cache Server, Gateways etc. können zw. Clients und den Ressourcen geschaltet werden und Performance zu steigern, Sicherheit zu erhöhen etc.;
- Unabhängige Entwicklung von Clients, Servern und *intermediaries*;
- Pull-basierte Interaktion zw. Client und Server mit der Möglichkeit, dass ein lokaler Applikation Server in einer Homepage eingebettet ist. Clients können einen push-basierten Server für push-basierte Interaktion beinhalten;
- Daten sollten stufenweise angezeigt werden.

Es sollte offensichtlich sein, dass SOAP und UDDI nicht diesen Prinzipen folgen. In Internet Foren sind heftige Diskussionen („Debate that Spawned a Thousand Threads“) darüber (REST vs. SOAP) entbrannt [Dodds2002], [6].

2.6.2 Water vs. Standards

2.6.2.1 SOAP vs. Water Protocol:

Ein Protokoll ist der Prozess und das Format um auf lokale oder entfernte Ressourcen zuzugreifen. Die weit verbreiteten Protokolle für den Zugriff auf entfernte Ressourcen sind HTTP, SMTP, FTP und SOAP, wobei nur die ersten beiden in Water eine Rolle spielen. Aus Gründen der Interoperabilität von Web Services, werden aber Methoden bereitgestellt, um SOAP Nachrichten zu erzeugen und zu interpretieren.
Ressourcen werden via URI adressiert, zum Beispiel wird auf eine HTML-Seite (Ressource) wie folgt zugegriffen:

```
<set my_homepage = www.<web http://www.toellich.net/>/>
my_homepage.<request/>
```

Listing 8: Zugriff auf Ressourcen via URI

Der Rückgabewert *my_homepage* ist eine Instanz von *request_response* und beinhaltet die komplette Homepage im Feld *my_homepage.body_string*.

Mit Water Protocol können eigene Protokolle definiert werden, zum Beispiel ein Protokoll, das eine Anfrage via HTTP sendet und die Antwort asynchron via FTP erhält. Diese Flexibilität eröffnet neue Wege für die Industrie um Daten und Prozesse gemeinsam zu nutzen.

2.6.2.2 WSDL vs. Water Web:

WSDL ist der XML Standard um das Interface und Protokoll einer entfernten Ressource zu beschreiben. Entfernte Ressourcen im Word Wide Web werden zu lokalen Ressourcen in Water Web. Einen Verbindung zu einer Ressource ist eine Instanz der Klasse *web*. Diese repräsentiert die Fähigkeit, auf die Ressource zuzugreifen.
Ein Water Web Objekt trägt all die notwendigen Informationen mit sich, die benötigt werden, ein WSDL zu generieren. Das Wesentliche an WSDL ist die Möglichkeit, die Namen und Typen der Argumente für den Aufruf eines Web Service zu bestimmen und zu überprüfen. Mit Hilfe von *Water Web Contract* ist dies möglich. Dazu muss kein externes WSDL Dokument mehr erzeugt werden; wir wissen ja, dass jede Ressource in Water ein Objekt, und jedes Objekt eine Instanz einer Klasse ist. Zu einer Ressource wird also eine Klasse entworfen und in der Klassendefinition Argumentennamen, Typen und Bedingungen angegeben.

Nehmen wir an, wir definieren einen Web Service, der die Temperatur von verschiedenen Städten ausgibt. Dieser Web Service (Ressource) wird mit „http://www.in.tum.de/current/temperatur?city=irgendeine“ aufgerufen. Der Parameter city darf nicht optional sein. Dazu definieren wir uns mit *defclass* eine Klasse, von welcher unser Web Service eine Instanz ist und definieren mit *defmethod* einen einfachen *contract*. Der Web Service wird nur ausgeführt, wenn das Argument city ein String ist:

```
<defclass web
     a_uri = required
     contract = optional
/>
www.<web
      <uri  „http://www.in.tum.de/current/temperatur“/>
contract = <defmethod city=required=string/>
/>
```

Listing 9: Definition der Parameter eines Web Service

Das Protokoll zur Interaktion mit dem Web Service wird durch die URI bestimmt, in unserem Beispiel HTTP. Ein selbstdefiniertes Protokoll „HTTP über Port 9910“ kann leicht erstellt werden. Der Aufruf erfolgt dann mit

```
<uri „http_9910://www.in.tum.de/current/temperature“/>
```

Listing 10: Bindung an selbstdefiniertes Protokoll

2.6.2.3 UDDI vs. Water Registry

Ziel von UDDI ist es, Web Ressourcen im einen von Maschinen lesbaren Format bereitzustellen. *Water Registry* bietet zwar Interfaces zu diesem Standard, erhebt aber den Anspruch, flexibler und einfacher handhabbar zu sein. Ein Water Server kann all seine oder die verfügbaren Ressourcen anderer Server publizieren. Die URIs/URLs, Typen und Einschränkungen sind über die Water Registry verfügbar. Dadurch können Geschäftspartner sehr leicht diese Ressourcen finden und nutzen.

Jede Web Seite ist nichts anderes als eine Menge von Ressourcen, durch die der Benutzer browsen kann. Er greift auf Ressourcen zu, indem er auf einen Hyperlink klickt. Der Hyperlink ist eine Referenz auf die Ressource; die Web Seite gewissermaßen das Benutzerinterface um erfolgreich in und zu den Ressourcen navigieren zu können.

Die Registry ist, wie bereits erwähnt, eine Liste von Ressourcen, auf die Maschinen zugreifen und verarbeiten können. Wenn eine Maschine (z.B. ein Agent) auf eine Web Seite zugreift, muss jener „verstehen“ können, um welche Art und welchen Typen von Ressource es sich jeweils handelt.

Jedes Water Objekt kann als Registry fungieren, weil jedes seiner Felder ein wohldefinierter Container für ein anderes Objekt ist, welches als Ressource agiert. Die

Ressource kann ein lokales oder ein entferntes Objekt sein. Der Zugriff auf eine Ressource ist entweder ein (parametrisierter) Methodenaufruf oder resultiert in der einfachen Rückgabe des Ressourceninhalts in Form eines Objektes.
Ein Water Registry Eintrag ist nichts anderes als ein Objekt bzw. Klasse, die mit *defclass* erzeugt werden und deren Felder die eigentlichen Ressourcen sind. Die Repräsentation der Ressource in XML kann vom Client instantiiert werden. Danach können Zugriffe auf die Ressource stattfinden – diese unterscheiden sich nicht von Methodenaufrufen eines Objektes. Methoden, um die Water Registry zu durchsuchen und zu filtern, können frei definiert werden.

2.6.2.4 XPATH vs. Water Path

Mit Hilfe von XPATH können Teile eines XML Dokumentes adressiert werden. Wie wir bereits wissen, basiert Water auf Klassen und Instanzen. Sowie XPATH benutzt wird, um auf bestimmten Teile eines XML Dokumentes zuzugreifen, wird *Water Path* eingesetzt, um Objekte anzusprechen bzw. in der Objekthierarchie zu navigieren. Das macht insoweit Sinn, da alles in Water als Objekt gilt. Water Path erhebt den Anspruch, durch den objekt-orientierten Ansatz, leichter verständlich und flexibler als das XPATH zu sein. Will man beispielsweise auf das Feld „surname“ eines Objektes „student“ zugreifen, reicht die Punk-Notation (vgl. XPATH) vollkommen aus:

```
student.surname
```

Listing 11: Zugriff auf das Feld surname des Objektes student

Will man mehrer Objekte zurückgeben, die einem bestimmten Kriterium entsprechen, erreicht man das wie folgt:

```
Student.<get_with_value „surname“ „Miguel“ returns=’all’>
```

Listing 12: Zugriff auf all die Studenten, die den Vornamen Miguel tragen

Ein interessantes Feature ist die Tatsache, dass Water Path in eine URI konvertiert werden kann, indem die Punkte durch Slashes ersetzt werden. Dadurch kann via Web Browser und URI auf Water Objekte zugegriffen werden, was die Entwicklung von Web Services wiederum stark vereinfacht.

2.6.2.5 XSLT vs. Water Transform

In fast jeder Anwendung müssen früher oder später externe Daten verarbeitet werden. Hierbei müssen meist Daten in eine andere Darstellungsform gebracht werden. XSLT ist eine spezielle Transformationssprache für Dokumente. Water kapselt diese Funktionalität,

indem es in der Programmiersprache selbst Methoden für Transformationen bereitstellt. Auch können eigenen Methoden entworfen werden.

Die Konventionelle Methode um ein XML Dokument in ein anderes Format zu transferieren: Das Ausgangsdokument wird von einem Stylesheet Prozessor mit Hilfe eines XSLT Stylesheets in das transformierte Dokument überführt. Der Stylsheet Prozessor (z.B. Xalan [7]) wird meist aus einem Programm (z.B. Java) aufgerufen. Mit Hilfe des Parsers (z.B. Xerces [8]) kann auf das transformierte Dokument zugegriffen. (vgl. Abb. 2)

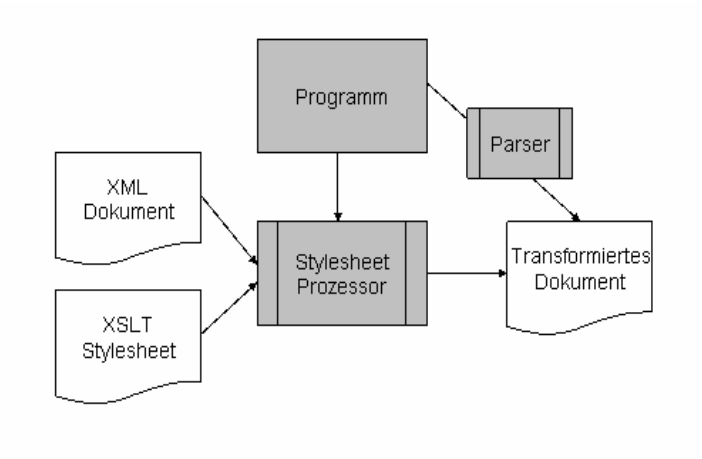

Abbildung 4: Transformation eines XML Dokumentes

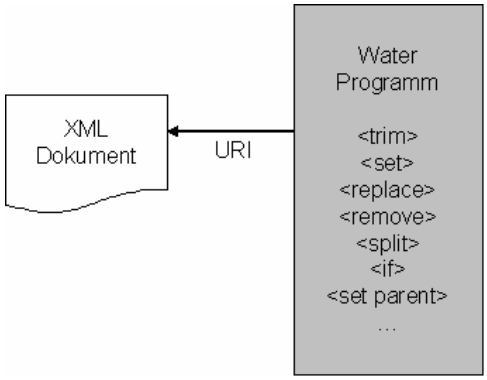

Abbildung 5: Transformationen mit Water Transform

In Water dagegen werden via URI auf die Objekte des externen XML Dokumentes zugegriffen. Die Transformationen erfolgen durch Methodenaufrufe auf diese Objekte. (vgl. Abb. 3)

2.7 Tools und Service

Um Water Anwendungen zu entwickeln, benötigt man nichts weiter als einen Texteditor, mit dem man den Quellcode in ConciseXML verfasst. Komfortabler ist die Benutzung der Steam DIE, eine integrierte Entwicklungsumgebung um Water Anwendungen zu entwickeln, auszuführen und zu testen. Um Water Anwendungen dann zu verteilen und den Benutzern und Maschinen bereitzustellen, wird eine Laufzeitumgebung benötigt. Diese *Steam Engine* kann in der Standard Java Plattform verteilt werden. Als Applet (Client Schicht), als Servlet (Business Schicht) oder als stand-alone Anwendung. Über die Lizenzgebühren schweigt sich Clear Methods leider aus. Als Lizenzmodelle für die Laufzeitumgebung werden unter anderem folgende Möglichkeiten angeboten: Pro Client oder Benutzer, pro Session, pro CPU. Der Support erfolgt entweder kostenlos durch die Community [9], ein Premium Support kostet $8000 pro Jahr, für bis zu fünf Entwickler.

Für eine jährliche Gebührt, die 20% der Lizenzkosten nicht übersteigt, erhält der Käufer kostenlose Updates und den Anspruch auf Support via Ticketsystem.

2.8 Zusammenfassung

Zusammenfassend lässt sich sagen, dass es die Entwickler von Water verstehen, all die Sprachen und Technologien, die man zur Entwicklung moderner Web Services benötigt, geschickt in einer einzigen Sprache zu verstecken. Das gestaltet sich umso leichter, da Water ebenso in XML notiert wird, und so der „Bruch“ zwischen Programmiersprache und Auszeichnungssprachen verwischt wird. APIs für die Benutzung von Technologien wie SOAP, WSDL, UDDI und XSLT wie z.B. in Java werden faktisch nicht benötigt, da Water für diese Funktionalitäten geeignete Sprachmittel und Methoden bereitstellt. Fairerweise muss man sagen, dass Water entstanden ist, als jene Technologien bereits auf dem Markt waren. Die Entwickler konnten sich so „die Rosinen herauspicken“, Kritikpunkte aufgreifen und somit ein integriertes Konzept bieten. Java dagegen gibt es schon viel länger – die APIs werden benötigt, um die Sprache gewissermaßen nachzurüsten, ohne den Sprachkern zu verwässern und Abwärtskompatibilität zu gewährleisten.

Water vertritt das Konzept der Vermischung von Daten, Präsentations- und Programmlogik. Dies steht im krassen Gegensatz zum Paradigma, diese drei Schichten strikt zu partitionieren und den Entwicklern Rollen zuzuordnen. Ein Web Designer ist für die Präsentationsschicht, ein Entwickler dagegen für die Programmlogik und ein Datenbankprogrammieren für die Dinge wie Persistenz und Ablage von Daten zuständig. Dies Konzept hat sich in der Industrie bereits tausendfach bewährt.

Die Befürworter von Water argumentieren, dass die Abdeckung aller genannter Technologien für die Entwicklung von Web Anwendungen durch eine einzige Sprache die Total Cost of Ownership reduziert, vor allem durch die Zeiteinsparungen bei der Entwicklung. Ich möchte dazu zu bedenken geben, dass in der Tat die Entwicklung kleiner Web Services oder Anwendungen mit Water zügig von statten geht, und man sich nicht in API Spezifikationen und der Syntax von SOAP und Co. einarbeiten muss. Was aber, wenn die die Anwendungen umfangreicher werden und ein großes Entwicklungsteam benötigt wird? Durch die Aufhebung der Rollentrennung sind Konflikte faktisch vorprogrammiert. Des Weiteren fehlt die Abdeckung bestimmter Bereiche des Entwicklungszyklus durch entsprechende Tools. Als Beispiele wären Software Configuration und Collaboration Management Systeme zu nennen. Auch könnten die eventuell eingesparten Kosten wieder durch die Lizenzgebühren eliminiert werden. Water ist nicht weit verbreitet. Dementsprechend fehlen „best practises“ und die breite Unterstützung von Community und Industrie.

3. JWIG

3.1 Überblick

JWIG [3] verfolgt das gleiche Ziel wie Water: Die Entwicklung von Web Services zu vereinfachen. Dabei geht JWIG einen weniger radikalen Weg als Water. JWIG ist keine neue Sprache, welche gegen Java und Co. positioniert wird und welche Ihrer eigenen Antworten auf Standards wie SOAP liefert. JWIG ist ein auf Java basierendes *Framework*, welches die Entwicklung von Web Services leichter und sicherer machen soll. JWIG ist ein Forschungsprojekt des BRICS Forschungszentrum an der Universität Aarhus, Dänemark [10]. Die Version 1.0 erschien im Juli 2002, derzeit ist die Version 1.1 aktuell.

JWIG basiert auf dem Java 2 Development Kit und dem Apache Web Server (für Linux und Solaris). JWIG besteht aus einem Übersetzer, einem Laufzeitsystem und einem Analysemodul. JWIG ist Open Source - es unterliegt der GNU Public Licence. Das JWIG Laufzeitsystem ist in Form eines Moduls (*DSO, Dynamic Shared Object*) für den Apache Web Server (Version 1.3.x) verfügbar.

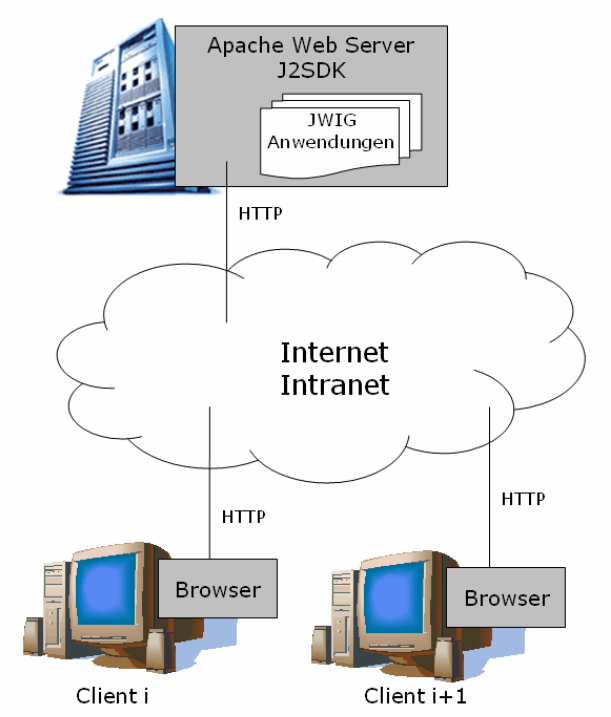

Abbildung 6: Architektur von JWIG

Das Framework adressiert zwei essentielle Aspekte, die bei der Entwicklung von Web Services eine Rolle spielen:

1) Dynamische Konstruktion von Dokumenten. In erster Linie sind das XML und XHTML Dokumente;
2) Session Management, da HTTP einen zustandslosen Charakter hat.

Auch bietet JWIG so genannte „statische Garantien“ - ein Analysemodul prüft bei der Kompilierung, ob dynamisch erzeugte XHTML Dokumente gültig sind und Formularfelder später wie erwartet vom Server empfangen werden können.
Bevor nähere Betrachtungen zu JWIG folgen, soll darauf aufmerksam gemacht werden, dass der Fokus von JWIG auf interaktive Web Services liegt. Das bedeutet, dass die Web Service Clients in diesem Fall Menschen mit ihren Internet Browsern sind und so HTML/XHTML das dominierende „Kommunikationsmittel“ ist. Aus diesem Grund werden Technologien wie SOAP und WSDL nur am Rande erwähnt, da diese eher für *„application-to-application*“ Web Services von Belang sind.
Eine komplette Internet Präsenz, die Web Seite der JAOO Konferenz [11], wurde mit Hilfe von JWIG implementiert. Diese „Anwendung“ besteht aus nur 4000 Zeilen Quellcode und zeichnet sich durch eine vollständige Separation von Geschäfts- und Präsentationslogik, die durch JWIG ermöglich wird, aus.

3.2 JWIG und bestehende Technologien

Wenn es um die die Entwicklung von dynamischen Anwendungen und Web Services geht, in denen in erster Linie HTML zum Browser des Benutzers geschickt wird, werden an erster Stelle Perl, PHP, ASP, JavaServlets und JSP genannt. All dies sind Skriptsprachen, die auf einem Server ausgeführt werden – nur die „Ergebnisse“ werden (in Form von HTML) mit Hilfe von HTTP zum Browser des Benutzers geschickt. So leistungsfähig diese serverseitigen Skriptsprachen sind, sie haben alle mit den gleichen Problemen zu kämpfen:
Die Ausgabedokumente werden als Menge einzelner Strings, durch Befehle wie echo (in PHP) oder println (mit JavaServlets), erzeugt. Dadurch kommt es zu einer Vermischung der Präsentations- mit der Programmlogik bzw. HTML und Quellcode. Listing 13 zeigt die Vermischung von HTML und Java Anweisungen in einem Java Servlet.

```
public class Hello extends HttpServlet {
      public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
      String name = (String) request.getParameter(" handle");
      if (name==null) {
            response.sendError(response.SC_BAD_REQUEST, "Illegal request");
            return;
      }
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Hello World</title></head><body>");
      ServletContext context = getServletContext();
      if (context.getAttribute("users")==null)
            context.setAttribute("users", new Integer(0));
      int users = ((Integer) context.getAttribute("users")).intValue() + 1;
      context.setAttribute("users", new Integer(users));
      HttpSession session = request.getSession(true);
      session.setAttribute("name", name);
      out.println("<form action=\"SendForm\">" +
            "Hello " + name + ", you are user number " + users +
            "<input type=\"submit\" value=\"GoOn\"></form>" +
            "</body></html>");
}
```

Listing 13: Vermischung von HTML und Java Code in einem Servlet

Auch werden die Dokumente meist linear, d.h. von „oben“ nach „unten“, erzeugt. Es ist offensichtlich, dass man bei dieser Methode leicht den Überblick verliert. Eine Rollentrennung zwischen Designer und Programmierer bzw. deren Kooperation wird erheblich erschwert. Des Weiteren existieren keine Technologien, die sicherstellen, dass die erzeugten Dokumente gültig sind. „Gültig“ bedeutet, dass die Syntax von XHTML, XML etc., die durch DTDs oder XML Schema festgelegt sind, korrekt ist. Zum Beispiel findet der Markup Validation Service des W3C [12], welcher Web Seiten unter anderem auf die Konformität mit dem XHTML Standard überprüft, auf www.microsoft.com 92 Fehler bzgl. HTML 3.2, auf www.javasoft.com sogar 611 Fehler bzgl. XHTML 1.1.

Auf dem Gebiet des Session Managements sind die Ansätze „Modifikation von URLs“, „versteckte Formularfelder“ und „Cookies“, die den Status von Sessions zw. den HTTP Request/Response Zyklen sichern sollen, nur unzureichend gelöst. Auch kann die Konsistenz zwischen den generierten Web Seiten und den Informationen der Formulare, die via „Submit“ an den Server geschickt werden, nicht gewährleistet werden (vgl. [Christensen]).

JWIG entgegnet diesen „Unzulänglichkeiten“, indem Dokumente dynamisch, mit Hilfe eine template-Systems erzeugt werden. Auf der anderen Seite unterstützt JWIG explizit Sessions, um die Verbindungslosigkeit des HTTP Protokolls mit Hilfe geeigneter Sprachmittel zu überbrücken.

3.2.1 Services und Sessions

Im Gegensatz zu anderen Sprachen, mit denen Web Services entwickelt werden können, ist JWIG ein Framework, in der den Sessions eine zentrale Bedeutung zukommt. Das Benutzerhandbuch [14] bezeichnet JWIG als „session-centered“. Eine Session besteht bekanntlich aus einer Folge von Interaktionen zw. dem Client und dem Server.
Sessions in JWIG werden zwar ganz traditionell vom Client initiiert, die Kontrolle übernimmt dann aber der so genannte *service-code*, nicht anderes als server-seitige Threads. Interaktion kommt dann zustande, wenn ein Dokument mit einem Formular an den Client geschickt wird. Der Thread wartet dann solange, bis der Benutzer das Formular ausgefüllt hat, und wird dann fortgeführt. (vgl. Abbildung 7 [Christensen])

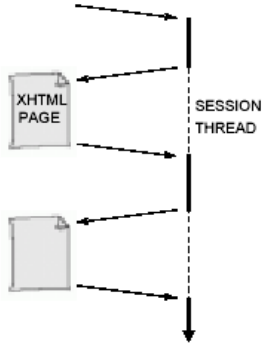

Abbildung 7 : Das Session Model von JWIG

Eine JWIG Anwendung ist eine Subklasse der Klasse *service*. Ein service-Objekt korrespondiert mit einer Instanz des Web Service und beinhaltet gemeinsam benutzte Daten, welches einfach die Felder des Objektes sind, sowie eine innere Klasse für jede Form von Session mit dem Client. Die Klasse *service* wird mit der ersten Anforderung des Clients instantiiert. Es existiert nur diese eine Instanz während der gesamten Lebenszeit des Web Service. Die Klasse *service* bietet darüber hinaus Funktionalitäten wie das Protokollieren von Ereignissen und die Unterstützung von SSL.

Die Klasse *service* beinhaltet eine Zahl von *session* Klassen. Dies sind nicht-statische, innere Klassen der Klasse *service* und Subklassen von *service.session*. Für jeden Request vom Client, der darauf abzielt eine Session zu starten, wird die korrespondierende Session Klasse instantiiert und deren main-Methode aufgerufen, wobei ein Thread gestartet wird. Jede Instanz der service-Klasse korrespondiert mit einem einzelnen Thread, der mit einem bestimmten Client kommuniziert. Das Session Objekt enthält dazu lokale Daten und bietet darüber hinaus show-, receive- und exit-Methoden zur Interaktion mit dem Client und Zugriffskontrollmechanismen via HTTP Authentifizierung. Das Session Objekt existiert solange, bis die Session beendet wird (durch den Aufruf des exit-Statements oder durch einen *timeout*).

Die gesamte Interaktion zwischen Client und Server gestaltet sich durch die Ausführung der show- und exit-Methoden. Das Statement *show param*, wobei *param* eine XML Ausdruck ist, sendet das XML Dokument an den Client, wartet auf dessen Antwort und setzt daraufhin die Ausführung des Session Threads fort. Das Statement *exit param*, sendet das XML Dokument *param* an den Client und beendet den Session Thread. Wenn die main-Methode abgearbeitet ist, wird die exit-Methode explizit aufgerufen.
Ein Dokument, das mit Hilfe von show oder exit an den Client gesendet wird, muss aus einem einzigen *<html>…</html>* Ausdruck bestehen. Die Dokumenttyp-Deklaration von XHTML 1.0 und die Zeichensatzcodierung werden automatisch eingefügt. Ein Dokument, das als Argument an das show-Statement übergeben wird, muss mindestens ein Form-Element und darf keine action-Attribute beinhalten. Das Laufzeitsystem von JWIG fügt ein spezielles action-Attribut ein, welches dafür sorgt, dass die Ausführung des Session Threads fortgeführt wird. Das geschieht dann, wenn der Client das Formular absendet.
Die Werte der Eingabefelder des abgeschickten Formulars könne mit Hilfe von *receive name* gelesen werden, wobei *name* der Name des zu lesenden Feldes ist. Wenn das Feld mehrere Werte enthält kann receive[] name benutzt werden, das das zurückgegebene Array liest. Wenn keine Werte zurückgeliefert werden, ist der Inhalt des Feldes explizit *null*.
Zur Verdeutlichung ein einfacher, obligatorischer “Hello World” Service:

```
import dk.brics.jwig.runtime.*;

    public class Hello extends Service {
        private static final XML wrap = [[
            <html>
              <head>
                 <title>A JWIG example</title>
              </head>
              <body>
                <[body]>
              </body>
            </html>
        ]];

        private static final XML question = [[
            <form>
               What is your name?
               <input type="text" name="person" />
               <br/>
               <input type="submit" name="answer" value="Answer" />
            </form>
        ]];

        private static final XML greeting = [[
            <h1>Hello <[who]>!</h1>
        ]];

        public class Test extends Session {
            public void main() {
```

```
                show wrap <[body = question];
                exit wrap <[body = greeting <[who = receive person]];
            }
        }
    }
```

Listing 14: Ein Web Service in JWIG

JWIG unterscheidet sich vom Servlet/JSP Ansatz dadurch, dass das Konzept der Sessions explizit, nicht implizit unterstützt wird. In CGI-Skripten, Servlets und JSP wird der Workflow von Interaktionen durch action-Attribute der Formulare gesteuert. Die Identität des Clients wird entweder in Cookies, versteckten Feldern oder in den URLs gespeichert. Der lokale Status einer Session muss explizit gespeichert werden und bei jeder Interaktion wiederhergestellt werden. IN JWIG werden Sessions explizit unterstützt. Eine Session wird wie ein sequentielles Programm geschrieben. Interaktionen mit dem Client ähneln dem Konzept von *Remote Method Invocations*. Der Lokale Status entspricht dem Status des Session-Threads. Dadurch können Sessions angehalten und später wieder fortgeführt werden. Der Verlaufliste des Browsers wird nicht mit kryptischen Referenzen gefüllt.

3.2.2 Das XML template-System

Wie wir bereits bei den Sessions gesehen haben, besteht die Ausgabe eines JWIG Service ausschließlich aus XHTML Dokumenten. Diese werden mit Hilfe von XML *templates* aus der JWIG Anwendung heraus erzeugt. Mit Hilfe von template Systemen können Geschäfts- und Präsentationslogik getrennt werden. Ein XML template ist eine Menge von XML Ausdrücken die unspezifizierte Teile, so genannte *gaps*, enthalten können. Diese gaps können zur Laufzeit mit Zeichenfolgen oder Vorlagen gefüllt werden, um komplexere Vorlagen zu gestalten.

In JWIG Anwendungen werden templates durch die Werte von Variablen des Typs XML repräsentiert. Diese Werte sind unveränderbar. Alle Operationen, die auf XML templates operieren, erzeugen neue Werte, daher auch die Bezeichnung XML template Konstanten. Diese bestehen aus reinen XML Statements, die von doppelten eckigen Klammern eingeschlossen werden. Die folgende Codezeile (Listing 9) deklariert eine Variable hello, die den Wert eines XML templates tragen soll und mit *„Hello World“* initialisiert wird.

```
XML hello = [[Hello <i>World</i>!]];
```

Listing 15 : Deklaration eines XML templates

Template gaps:

XML template Konstanten können benannte gaps beinhalten, die der Syntax <[name]> folgen. Der Name des gaps muss ein gültiger Java Bezeichner sein. Diese gaps können innerhalb von Texten und Tags platziert werden, so als ob sie selbst Tags wären. Derartige gaps werden als *template gaps* bezeichnet.

Attribute gaps:

Gaps können aber auch innerhalb von Tags, als Werte von Attributen, platziert werden. Diese Tags werden als *attribute gaps* bezeichnet. *<a href = [link]>* ist ein HTML anchor Start-Tag, dessen *href* Attribut ein gap namens *link* ist.

Plug Operator:

Gefüllt werden gaps mit Hilfe des plug Operator. Dies ist ein XML Ausdruck nach dem Schema *exp1 <[name = exp2]*, wobei *exp1* ein Ausdruck des Typs XML ist, *name* der Bezeichner, der den Namen des gaps angibt, *exp2* ist ein Ausdruck des Typs XML oder einen Zeichenkette (String). Der Ausdruck liefert eine Kopie von exp1 zurück, wobei alle gaps *name* mit einer Kopie von exp2 ersetzt werden.

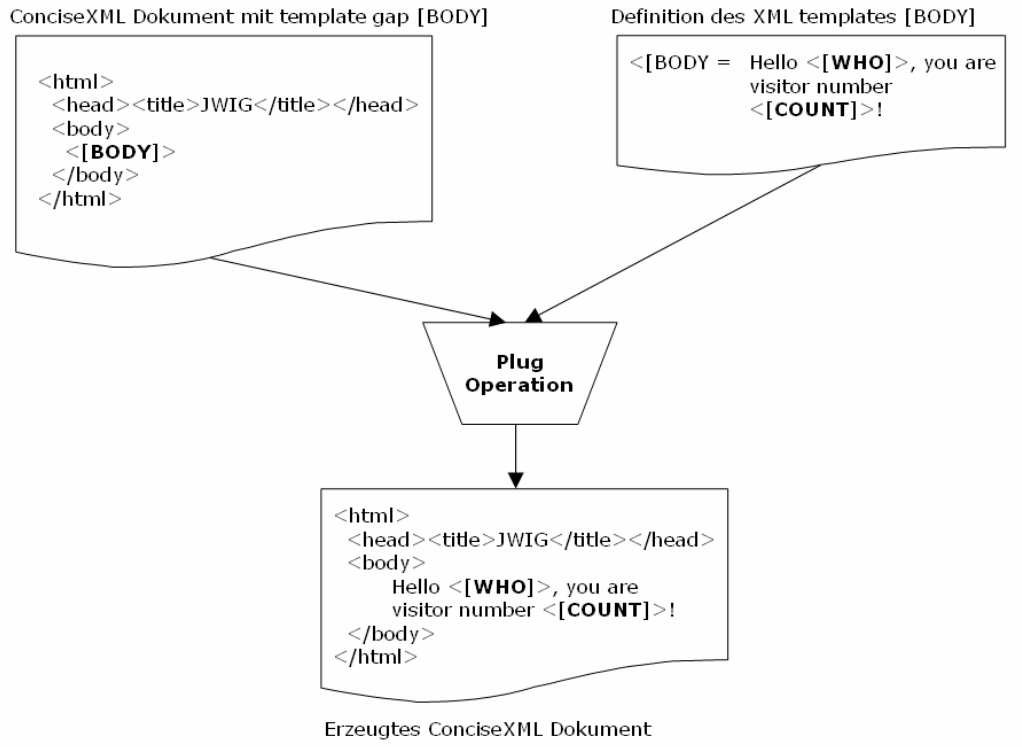

Abbildung 8: der plug Operator – [BODY] wird durch “Hello <[WHO]>, you are visitor…“ ersetzt. Ebenso können [WHO] und [COUNT] ersetzt werden

Code gaps:

Zu den attribute und template gaps kommen noch *code gaps* hinzu, die Java Anweisungen enthalten. Die Syntax für code gaps ist <*{code}*>. Die Java Anweisungen werden ausgeführt, wenn das Dokument angezeigt wird. Das Ergebnis wird an die Stelle des Dokuments eingefügt, an der sich das code gap befindet. Listing 16 fügt das aktuelle Datum ein

```
XML date = [[The current date is now <i><{ return new Date(); }></i>]];
```

Listing 16: Beispiel für ein code gap

Anstelle von inline code, kann eine XML template Konstante in einer externen Datei platziert werden und diese Datei zur Laufzeit mittels des Konstruktes *get url* geladen werden.

In CGI-Skripten und JavaServlets werden XML Statements implizit verwendet, d.h. alle XML Statements sind Ausgaben von print-Anweisungen. Funktionalität und Präsentation eines Web Service sind eng miteinander verwoben. In JWIG werden XML Statements dagegen explizit unterstützt: Sie sind Instanzen der Klasse XML, genauso wie Zeichenfolgen Instanzen der Klasse String sind. Sie sind keine bloßen Ausgabestrings, sondern Objekte, die weiterverarbeitet werden.

Die Manipulation von XML Dokumenten mit Hilfe von JWIG gestaltet sich also komplett anders als mit JavaServlets. Hier müssen Dokumenten linear erzeugt werden; eine Prüfung auf die Gültigkeit der erzeugten Dokumente ist nicht möglich.

Das template-System von JWIG erlaubt es auch, dass die Präsentationslogik zur Laufzeit des Web Services ausgetauscht werden kann.

3.2.3 Statische Analyse

Nach dem Theorem von Rice ist das Verhalten von Programmen nicht immer voraussagbar. Es stellen sich folgende Fragen:

- Terminiert mein Programm (Halteproblem)?
- Wie viel heap space benötigt mein Programm?
- Dereferenziert mein Programm Programm null pointer?
- Erzeugt mein Programm gültige XHTML Dokumente?

Um diese Fragen zu beantworten, bedient sich JWIG einer statischen Analyse: Bei der Kompilierung von JWIG Anwendungen werden spezielle Analysen durchgeführt.

Als erstes werden aus den JWIG Klassen Flussgraphen (flow-graphs) erstellt. Von diesen ausgehend werden so genannten *summary-graphs* erzeugt, mit deren Hilfe plug-,

receive- und show-Analysen durchgeführt werden. Diese prüfen das Vorhandensein und die richtige Verwendung der gaps, die richtige Reihenfolge der Eingabefelder in den Dokumenten, so dass receive und receive[] Operationen immer erfolgreich sind; und zuletzt, dass gültige XHTML 1.0 Dokumente erzeugt werden. Die Gültigkeit von XML bzw. XHTML kann mit Hilfe unterschiedlicher Formalismen geprüft werden. Die bekanntesten dürften DTD, XML Schema und RelaxNG sein. Die Entwickler von JWIG haben sich für DSD2 entschieden, weil es mächtiger und dabei einfacher als jene sein soll und sehr gut für die Analyse der *summary graphs* geeignet ist.

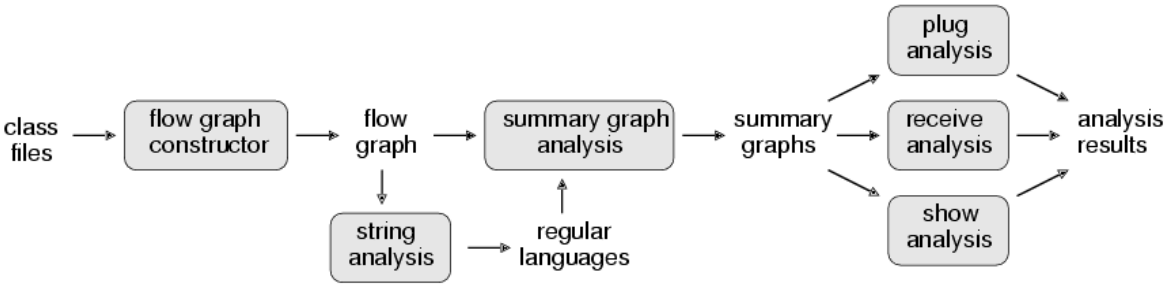

Abbildung 9 : Überblick über die durchgeführten Analysen

3.2.4 JWIG und Standards

2.3.4.1 SOAP

Die Hauptverbindung zw. JWIG und SOAP ist die erwähnte statische Analyse: Sie sorgt dafür, dass vom Entwickler erzeugte *SOAP envelopes* syntaktisch korrekt sind.

2.3.4.2 WSDL

WSDL kann als „Dokumentation“ für ein existierenden Web Service agieren oder dazu benutzt werden, *stub code* für eine bestimmte Programmiersprache zu generieren. Auch für JWIG Programme.

3.2.5 Sonstiges Merkmale

JWIG bietet noch eine Reihe weiterer interessanter Features, die wir kurz betrachten wollen: Zur Laufzeit sorgt eine garbage collector dafür, dass nicht mehr benötigte Sessions aus dem Speicher entfernt werden.

Mit Hilfe von *PowerForms* [14] können Eingabefelder in den Formularen validiert werden. Dazu stehen Boolsche Logik und Reguläre Ausdrücke zur Verfügung. Der dazu notwendige client-seitige Javascript- und server-seitige Javacode wird automatisch erzeugt.

3.3 JWIG und Java

Wie bereits erwähnt, ist JWIG ein Framework, das die Entwicklung von Web Services *in Java* erleichtern soll. JWIG soll nicht als Konkurrenz zu Java, sondern als Erweiterung

gesehen werden. Java bietet den ideale Basis für die Entwicklung von Web Services: Java ist portabel und plattform-unabhängig. Die Laufzeitumgebung bietet einen garbage collector und lässt Anwendungen in einem sicheren sandbox Modus ausführen. Des Weiteren unterstützt Java Unicode, bietet Funktionen zum Zugriff auf Netzwerke, Unterstützung von Threads, Serialisierung von Objekten, eingebaute Kryptographiealgorithmen wie RSA und nicht zuletzt das dynamische Laden von Klassen.

JWIG Programme werden bei der Kompilierung vollständig in Java übersetzt (vgl. Abbildung 10).

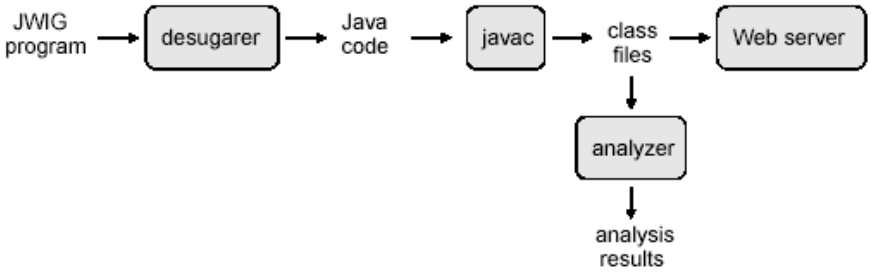

Abbildung 10 : Kompilierung von JWIG Anwendungen

Zuerst werden syntaktische Konstrukte von JWIG mit Hilfe eines einfachen *source-to-source Transformators* in entsprechende Java Konstrukte übersetzt. Diese Java Sourcen werden mit dem Java Compiler in Java Klassen kompiliert. Diese kompilierten Klassen bilden zusammen mit den extern spezifizierten XML template Konstanten den Web Service.
Der in 3.2.3 angesprochene *analyzer* untersucht die erzeugten Klassen auf folgende Punkte:

- All die Statements in den Anwendungsklassen, die Methoden aufrufen, müssen entweder Methoden in den Anwendungsklassen oder Methoden in den Nicht-Anwendungsklassen aufrufen. Anwendungsklassen sind diejenigen Klassen, die dem analyzer als Parameter übergeben werden. All die anderen Klassen sind Nicht-Anwendungsklassen;
- Auf Felder und Methoden von Anwendungsklassen darf nicht durch Nicht-Anwendungsklassen zugegriffen werden. Jene sind für letztere *private*;
- In Nicht-Anwendungsklassen dürfen keine XML Operationen durchgeführt werden.

Für die Analyse erzeugt der *analyzer* die in 3.2.3 erwähnten *flow*- und *summary-graphs*, auf die hier nicht weiter eingegangen werden soll. Näheres dazu in [Christensen].

3.4 Zusammenfassung

Web Services mit Hilfe von JWIG zu entwickeln gestaltet sich für den Java Entwickler sehr einfach. Es muss sich mit keiner neuen Sprache vertraut machen – JWIG integriert sich als Framework nahtlos in Java und nutzt dabei dessen mächtige Funktionalität. Gleichzeitig bügelt es „Unzulänglichkeiten“ aus, die sich bei der Entwicklung von Web Services und Web Anwendungen mit Java ergeben. Im Gegensatz zu JSP untersützt es durch sein template-System die Trennung von Präsentations- und Geschäftslogik und unterstützt Sessions explizit. Ein Novum ist die Prüfung der erzeugten Dokumente auf Ihre Gültigkeit und hat mit Sicherheit Vorbildcharakter für andere Frameworks. Entwickler werden sich darüber freuen, dass Web Services in JWIG nach dem, aus SmallTalk bekannten und in der Java Welt weit verbreiteten *Model-View-Controller* Pattern entworfen werden. Das „Model“ beinhalten dabei die Daten, die „View“ generiert unterschiedliche Präsentationen dieser Daten und der Controller ist für die Interaktion mit den Clients verantwortlich. In JWIG basiert das Model auf XML, die templates übernehmen die Rolle der „View“ und die JWIG Sessions übernehmen den Part des Controllers.

4. Vergleich Water und JWIG

Water ist mit Sicherheit der radikalere Ansatz, die Entwicklung von Web Services und Web Anwendungen vereinfachen zu wollen. Während die gesamte Industrie den Weg geht, Geschäfts- und Präsentationslogik zu trennen, sowie klar definierte APIs zwischen den verschiedensten Technologien und Standards zu schaffen, um die Komplexität von Anwendungen zu reduzieren, Erweiterbarkeit zu sichern und klare Rollentrennungen beteiligten Personen zu fördern, geht Water den umgekehrten Weg. Die Nutzung der Funktionalität von SOAP, WSDL, XSLT etc. soll in einer einzigen, XML notierten Sprache ohne die Zuhilfenahme von APIs und die Einarbeitung in zusätzliche Technologien möglich sein. Die Vermischung von Präsentations- und Geschäftslogik, sowie den Daten wird billigend in Kauf genommen.

JWIG dagegen schmiegt sich sanft in bestehende Systeme ein. Es nutzt die Vorteile von Java und erweitert es lediglich um neue Konzepte wie templates und die verbesserte Unterstützung von Sessions. Durch die Trennung von Geschäfts- und Präsentationslogik folgt JWIG dem eingeschlagenen Weg der Industrie. Damit, denke ich, darf JWIG eine breitere Akzeptanz erwarten als Water. Ich bin auch der Meinung, das die Philosophie, ein breites Thema wie Web Services mit einem Produkt „erschlagen“ zu wollen, veraltet und wenig zukunftsträchtig ist.

Ein weiteres Thema ist die Integration mit bestehenden Systemen und Software Tools. Auch da hat, meiner Meinung nach, JWIG den besseren Ansatz. Vollständig Java basiert

und bei der Kompilation in Java Klassen übersetzt, können JWIG Anwendungen leicht in bestehende Systeme integriert und auf bewährte Technologien der J2EE oder Tools wie Ant [15] und Axis [16] zugegriffen werden. JWIG Anwendungen können so von (künftigen) Entwicklungen der gesamten Java Welt profitieren. Entwickler von Web Services in Water dagegen sind darauf angewiesen, dass der Hersteller entsprechende Erweiterungen im Sprachumfang vornimmt.

Beiden Sprachen gemein ist die intensive Benutzung von XML. In Water und in JWIG gibt es eine Klasse XML - die Erzeugung von XML Dokumenten basiert nicht auf der Ausgabe von Strings, die XML Fragmente enthalten.

Vom Grundkonzept her hat JWIG einen etwas konservativeren Ansatz. Es ist ein weiteres Framework in einer Reihe vorhandener Java Frameworks wie zum Beispiel Apache Struts [17] und Turbine [18]. Basierend auf der REST Architektur darf das Konzept von Water dagegen als sehr progressiv bezeichnet werden. Die Tatsache, dass man komplette Web Seiten als Objekte behandelt, auf denen Methoden etc. angewendet werden können, weckt auf jeden Fall Interesse.

Literaturverzeichnis:

[1] Water: Simplified Web Services and XML Programming: http://www.waterlang.org

[2] Clear Methods: Software Plattform for Pure Web Services: http://www.clearmethods.com/

[3] JWIG: Java Extensions for High-Level Web Service Development: http://www.brics.dk/JWIG/

[4] Water Quick Reference, http://www.waterlang.org/quick_reference_guide.html

[5] Sun: Java Web Services Developer Pack: http://www.javasoft.com

[6] Bay Area Web Services User Group, http://www.myspotter.net/links.html

[7] Xalan Java XSLT Processor, http://xml.apache.org/xalan-j/index.html

[8] Xerces2 Java Parser, http://xml.apache.org/xerces2-j/index.html

[9] Water Language Community, http://groups.yahoo.com/group/waterlanguage

[10] BRICS, Basic Research in Computer Science, http://www.brics.dk/

[11] JAOO Conference on Java Technology and Object-Oriented Software Engineering, http://www.jaoo.org

[12] W3C Markup Validation Service, http://validator.w3.org

[13] DSD2, Document Structure Description 2.0, http://www.brics.dk/DSD/dsd2.html

[14] PowerForms, http://www.brics.dk/~amoeller/WWW/powerforms/

[15] Ant, http://ant.apache.org/

[16] Axis, http://ws.apache.org/axis/

[17] Struts, http://jakarta.apache.org/struts/

[18] Turbine, http://jakarta.apache.org/turbine/

[Fielding2000]
Roy Fielding: Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine, 2000

[Dodds2002]
Leigh Dodds: REST Roundup, http://www.xml.com/pub/a/2002/05/08/deviant.html, 2002

[Christensen]
Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach: Extending Java for High-Level Web Service Construction, BRICS, Department of Computer Science University of Aarhus, Denmark

[Box01]
Don Box, Aaaron Skonnard, John Lam: *Essential XML – XML für die Softwareentwicklung*, Addison-Wesley, München, 2001

[Fry2002]
Mike Fry: Water Rationale, http://www.waterlang.org/doc/water_rationale.htm, 2002