

# Diplomarbeit

## Konzeption, Erstellung und Bewertung eines Online-Redaktionssystems unter Verwendung des Applicationservers Tango2000

vorgelegt am: 16. Mai 2001

von: Stephan Menzel

Fachbereich Physikalische Technik / Informatik  
Fachgruppe Ingenieurinformatik  
Westfälische Hochschule Zwickau (FH)

Matrikelnummer: 962079 / 11 7671

Betreuer: Dr.-Ing. Birk Bockelmann  
Prof. Dr. Rolf Urban

# Inhaltsverzeichnis

<b>1. EINLEITUNG .....</b>	<b>4</b>
1.1. VORWORT .....	4
1.2. AUFGABENSTELLUNG.....	6
1.3. DANK .....	7
<b>2. GRUNDSÄTZLICHE BETRACHTUNGEN, HTML-STRUKTUR, SYSTEMENTWURF .....</b>	<b>8</b>
2.1. HTML UND DESSEN DATEN .....	8
2.1.1. Nutzerdaten.....	8
2.1.2. Strukturdaten .....	8
2.1.3. Metadaten .....	10
2.2. AUFGABEN UND ANFORDERUNGEN .....	11
2.3. LÖSUNGSMODELLE.....	11
2.3.1. Clientbasiertes System .....	11
2.3.2. Serverbasiertes System.....	12
2.3.3. Eine komplexe Client-Server-Struktur .....	13
2.4. ENTWURF .....	14
<b>3. VERWENDETE MITTEL UND RESSOURCEN.....</b>	<b>16</b>
3.1. MIDDLEWARE UND APPLICATIONSERVER.....	16
3.2. DER APPLICATIONSERVER TANGO2000 .....	17
3.2.1. Tango Server.....	17
3.2.2. Tango Editor.....	19
3.2.3. Der Tango Web Analyzer.....	22
3.3. PERVASIVE.SQL.....	24
3.3.1. MicroKernel Database Engine (MKDE).....	25
3.3.2. SQL Relational Database Engine (SRDE).....	25
3.3.3. Pervasive Database Utilities.....	25
3.4. DIE EINGESETZTE HARDWARE.....	26
<b>4. REALISIERUNG .....</b>	<b>28</b>
4.1. NUTZERVERWALTUNG.....	28
4.2. SEITENVERWALTUNG UND METADATEN .....	29
4.3. DIE MÖGLICHKEITEN ZUR ABBILDUNG DER DOKUMENTENSTRUKTUR IN DER DATENBANK .....	31
4.3.1. Blockbasierte Datenhierarchie .....	31
4.3.2. Seitenbasierte Datenhierarchie.....	33
4.4. DIE TABELLENSTRUKTUR DER KONSTRUKTIONSDATEN .....	35
4.4.1. Tags als Tabellenelemente .....	36
4.4.2. Inhalte als Tabellenelemente .....	38
4.4.3. Demonstration .....	39

4.5. DIE STILDEFINITIONEN .....	41
4.6. DIE SEITENKONSTRUKTIONSKLASSE .....	42
4.6.1. Die Membervariablen .....	43
4.6.2. Das Interface der Klasse <i>Seite.tcf</i> .....	44
4.6.3. Die temporäre Bearbeitungsseite als Resultat der <i>edit</i> -Methode .....	51
4.6.4. Die Funktionsweise der Seitengenerierung .....	56
4.6.5. Behandlung von Umlauten .....	58
4.7. DIE ANWENDUNGSSCHICHT .....	59
4.7.1. Ein Element hinzufügen .....	59
4.7.2. Das Aufrufen der Bearbeitungsseite .....	61
4.7.3. Das gemeinsame Bearbeitungs-TAF .....	61
4.7.4. Die Menüstruktur .....	62
4.7.5. Allgemeines und Dokumentenweite Einstellungen .....	62
<b>5. BEWERTUNG UND ANALYSE .....</b>	<b>64</b>
5.1. PROBLEME UND MIBSTÄNDE .....	64
5.1.1. Geschwindigkeit .....	64
5.1.2. Nutzerführung innerhalb des Browsers .....	65
5.1.3. Bedienkomfort und Oberfläche .....	65
5.2. ERFÜLLTE VORGABEN .....	66
5.3. ZUKÜNFTIGE VERBESSERUNGEN UND ERWEITERUNGEN .....	67
<b>ANHANG A .....</b>	<b>69</b>
THESEN .....	69
<b>ANHANG B .....</b>	<b>70</b>
GLOSSAR UND ABKÜRZUNGSVERZEICHNIS .....	70
<b>ANHANG C .....</b>	<b>77</b>
QUELLENVERZEICHNIS .....	77
<b>ANHANG D .....</b>	<b>78</b>
ERKLÄRUNG ZUR SELBSTÄNDIGEN ANFERTIGUNG .....	78
<b>ANHANG E .....</b>	<b>79</b>
ANMERKUNG ZU QUELLCODES .....	79

*Das Internet ist ein großes aber schlichtes Reich.*

*Ein bißchen wie Rußland.*

Max Goldt

# 1. Einleitung

## 1.1. Vorwort

Das Verständnis des Internet durch breite Bevölkerungsschichten hat in den Jahren seit seiner Entwicklung einen tiefgreifenden Wandel erfahren. Diese Veränderungen interagierten mit den Entwicklungen des Netzes an sich und gestalteten es. Waren es anfangs noch wenige, die per Telnetsitzungen und FTP im jungen und, dank Gopher und Archie überschaubaren, Internet ihren beruflichen Tätigkeiten nachgingen um später die Probleme dabei im Usenet zu diskutieren, so verlagerte sich mit der Verbreitung von HTML Anfang der Neunziger Jahre das Nutzerspektrum zunehmend in den kommerziellen und damit in den privaten Bereich. Aus einem Werkzeug wurde ein Unterhaltungsmedium, welches sich sinnvolle Nutzungsmöglichkeiten erst zurückerobern mußte. Aus einer sich durch Vernunft und Logik selbst regierenden Anarchie ein auch mühsam nicht wirklich kontrolliertes Chaos.

„Multimedia“ hielt Einzug. Bunte Bilder, MP3, Flash und Internetradio sorgten für kurzweilige Unterhaltung und nie dagewesene Datenfluten. Bandbreiten, die noch vor einem Jahr völlig ausreichten, waren plötzlich hoffnungslos unterdimensioniert. Die technische Entwicklung konnte kaum mit den Bedürfnissen mithalten, welche die Anbieter derartiger Dienste gern bei Ihren potentiellen Nutzern zu sehen pflegten. Die Realität blieb nicht selten hinter den Erwartungen zurück. Statt seine Freizeit vor dem Rechner zu verbringen, erlosch das Interesse meist nach kurzer Zeit und der Erkenntnis, daß hinter den bunten Fassaden meist nur die sprichwörtlichen Hohlräume zu finden sind. „They came, they surfed and they went back to the beach“ formuliert es Sally Wyatt<sup>1</sup> treffend.

Neue, nützliche Angebote wollten dem vor Kaufkraft nur so strotzenden Netzpublikum zugänglich gemacht werden. „eCommerce“ wurde das neue Schlagwort. Buchstäblich alles sollte fortan im Internet leichter, preiswerter und generell viel besser zu haben sein als im

---

<sup>1</sup> <http://www.independent.co.uk/news/Digital/Update/2000-12/internet051200.shtml>

nicht vernetzten Laden an der Ecke. Und wieder machte die Wirklichkeit des Pragmatismus den kühnen Plänen der alles am besten sofort haben wollenden ein jähes Ende. Drastisch zu spüren bekamen das die Verfechter des neuen großen Hype. Aus dem gleichnamigen Boom wurde schnell das DotCom-Sterben. Technologieaktienindizes auf der ganzen Welt setzen nach einem beispiellosen Höhenflug, in dem bereits alle Gesetze der Logik außer Kraft schienen, zu einer harten aber notwendigen Landung an. Leicht prophetisch bemerkte die New York Times 1997: „Im Jahr 2000 werden 90% aller Unternehmen im Internet sein. Aber nur 15 % werden wissen warum.“

Doch wo liegen die Ursachen für diese Entwicklung ?

Zum einen mag es sicher mangelndes Verständnis für die neuen Technologien in der breiten Bevölkerung sein. Nur weil jemand theoretisch die Möglichkeit hat, die Waren seiner Wahl im Internet zu erwerben, bedeutet das nicht, daß er oder sie das automatisch auch tut. Viele ziehen eben einen realen Einkaufsbummel vor, statt sich mit Interneteinwahl, Verschlüsselung, eMail und Suchmaschinen zu befassen.

Ein weiterer wichtiger Grund ist mangelndes Vertrauen. Noch immer existiert kein allgemein akzeptiertes und etabliertes Zahlungsmittel im elektronischen Handel. Kreditkartennummern sollen übermittelt werden, nachdem Sicherheitsprotokolle aktiviert und Passworte abgefragt wurden. Derartige Dinge erzeugen beim Kunden Mißtrauen und lassen ihn wiederum den Einkauf mit Bargeld im Laden vorziehen. Laut einer von IBM in Auftrag gegebenen Studie<sup>2</sup> verlassen sich nur 15% aller Nutzer auf den Datenschutz im Netz. Darüber hinaus ist für viele der Einkauf bei einem virtuellen Gebilde mit weniger Vertrauen in Sachen Produkthaftung verbunden. Wenn tatsächlich mal etwas kaputtgeht, will man sich mit seinen Beschwerden direkt an den Verkäufer wenden können. Populäre Berichte über die unlauteren Machenschaften der neuen Reichen des Internet tun ihr übriges, um die Kaufentscheidung in althergebrachter Weise stattfinden zu lassen.

Von einigen Ausnahmen abgesehen fühlt sich die Mehrheit der Nutzer im Netz nicht heimisch, betrachtet es als eine fremde, technische Welt, die unmöglich zu verstehen und daher nicht vertrauenswürdig ist. Ziel der Netzdienstleister sollte es daher sein, den Bürger und Kunden im Netz heimisch zu machen, ihm deutlich zu machen, daß es in seinem eigenen Interesse liegt, sich mit dem neuen Medium vertraut zu machen und es nicht länger als großes fremdes und nicht beherrschbares Reich zu sehen. Ein beliebter werdender Schritt in diese Richtung ist die eigene Netzpräsenz. Die eigene Anwesenheit im Netz demonstrieren, indem man ihr eine Bühne errichtet. Die erste eigene Homepage. Leider ist die Realisierung dieser

Idee für viele mit technischen Unannehmlichkeiten und dem Erlernen von HTML oder der Bedienung von Editoren verbunden. Nicht immer bieten diese Editoren für den Einsteiger den nötigen Überblick. Oftmals werden, um viele Gestaltungsmöglichkeiten zu bieten, unhandliche Oberflächen in Kauf genommen. Fachbegriffe und Client-Server-Terminologie verwirren den Nutzer. Im Gegensatz dazu läßt bei den gängigen Baukastensystemen die Flexibilität stark zu wünschen übrig. Darüber hinaus sind die Resultate derartiger Systeme oft unbefriedigend. Unsauberes und inkompatibles HTML sind keine Seltenheit.

## 1.2. Aufgabenstellung

Der vorliegenden Arbeit liegt der Wunsch nach einem einfach zu bedienenden und dennoch flexiblen, serverbasierten Redaktionssystem zugrunde. Das System richtet sich an die privaten Kunden eines Internetproviders, denen mit ihrem Netzzugang zugleich die Möglichkeit gegeben werden soll, eine einfache Homepage zu erstellen und sich damit zu präsentieren. Es sollte sich durch möglichst leichte Bedienbarkeit direkt im Webbrowser des Nutzers auszeichnen ohne dabei zu viele Kompromisse im Bereich der darstellbaren Elemente einzugehen. Der Nutzer sollte zwischen verschiedenen Stilvorlagen für seine Seite wählen können. Eine unkomplizierte Benutzerverwaltung für den Systemadministrator war ebenso erforderlich wie eine übersichtliche und transparente Anordnung der gespeicherten Daten. Skalierbarkeit und leichte Wartung waren gefordert. Weiterhin sollte aus betriebstechnischen Gründen der Applicationserver „Tango2000“ Verwendung finden.

Die Realisierbarkeit und praktische Einsatzfähigkeit eines solchen Systems waren durch praktische Entwicklung zu analysieren.

Als Projektname wurde „Athen“ gewählt und im folgenden Text wird diese Bezeichnung verwendet.

Der Autor wird dabei nicht übermäßig intensiv auf die grundlegenden Eigenschaften von HTML und dessen Eigenschaften eingehen. Auch das Basiswissen der objektorientierten Programmierstrategien soll so weit wie möglich unerwähnt bleiben. Zu diesem Thema haben schon zahlreiche andere Diplomierende, Schriftsteller und Philosophen alles verfassenswerte verfaßt, so daß sich der Autor die Freiheit genommen hat, diese Informationen im Geiste der werten Leserschaft voranzusetzen.

---

<sup>2</sup> <http://www.heise.de/newsticker/data/cp-05.10.99-004/>

## 1.3. Dank

Danken möchte in an dieser Stelle den Mitarbeitern des Unternehmens Regionett Leipzig, speziell Herrn Dr-Ing. Dirk Bockelmann, welcher mir die Möglichkeit gab, diese Arbeit anzufertigen und dessen Rat und Kritik stets eine wichtige Hilfe war.

Weiterhin danke ich meinen wahren Freunden und meiner Familie für ihren Rückhalt, ihre Unterstützung und ihr Verständnis, welches sie angesichts eines stets am Computer beschäftigten Mitmenschen aufgebracht haben. Ohne Euch wäre ich gescheitert.

Freundlicher Dank geht außerdem an die reguläre Teilnehmerschaft in den folgenden Newsgroups des Usenet:

`de.comm.infosystems.www.authoring.misc`

`de.comp.datenbanken.misc`

`de.comp.lang.php`

`de.comp.lang.java`

Ihr wart mir eine große Hilfe.

# 2. Grundsätzliche Betrachtungen, HTML-Struktur, Systementwurf

## 2.1. HTML und dessen Daten

Wie schon in zahllosen Betrachtungen über das Internet dargelegt, ist HTML eine vom CERN entwickelte Hypertextsprache auf der Basis von SGML. Sie zeichnet sich durch eine einfache und enorm skalierbare Struktur aus. Alle Inhalte sind in klar definierten Struktureinheiten, sogenannten Tags, untergebracht, die sich in beliebiger Tiefe schachteln lassen. Dabei ist zwischen drei Typen von Daten zu unterscheiden.

### 2.1.1. Nutzerdaten

Die eigentlichen Nutzerdaten sind diejenigen Informationen, die der Autor der Seite dem Leser präsentieren will. Das können zum Beispiel Texte, Bilder, Dateien zum Download und visualisierte Inhalte einer Datenbank sein. In diesem Zusammenhang ist zwischen statischen und dynamischen Webinhalten zu unterscheiden. Erstere werden bei der Erstellung des HTML-Dokumentes vom Autor eingegeben und bleiben statisch unveränderlich. Dynamische Inhalte dagegen werden von einer serverseitigen Software als Antwort auf den HTML-Request des Clienten generiert. Auf derartige Software wird im Kapitel 3.1. „Middleware und Applicationserver“ näher eingegangen.

Im speziellen Fall sind diese Nutzerdaten statischer Natur. Sie werden vom Anwender eingegeben und können von ihm geändert werden, sind aber darüber hinaus keinen dynamischen Veränderungen unterworfen.

### 2.1.2. Strukturdaten

Weiterhin integriert sind die Strukturinformationen für das darstellende Medium (ggf. den Webbrowser). Diese Daten bestimmen die logische Struktur des darzustellenden Dokuments. Unterschieden wird zum Beispiel zwischen Überschriften, Zitaten, Textparagrafen und Tabellen. Die Entscheidung, wie die betreffenden Inhalte dann dargestellt werden, liegt auf der Seite des Browsers. In HTML werden diese Art von Daten im Inneren von sogenannten Tags, eingeschlossen in `<` und `>`, übermittelt.

Beispiele sind:

- <h1> - eine große Hauptüberschrift
- <p> - allgemeiner Absatz
- <li> - ein Listenelement

Wichtig ist dabei der Grundsatz, daß überall dort, wo Nutzerdaten enthalten sind, diese durch ein abschließendes Tag eingeschlossen werden. Vereinzelt (wie am Beispiel des <p>-Tags) kann dieser Abschluß weggelassen werden, was aber bei komplexen Dokumenten zu schlechtem Stil und unter Umständen sogar zu Fehlinterpretationen führen kann. Daher sollte auch bei solchen Elementen stets ein Abschluß vorhanden sein, welcher immer aus einem Schrägstrich („slash“) und dem beginnenden Tag besteht. Die korrekte Form für die Inhalte eines simplen Dokumentes sieht demnach folgendermaßen aus<sup>3</sup>:

```
<h1>Das ist die &Uuml;berschrift</h1>  
<p>Und das hier ist ein Absatz</p>
```

Im Gegensatz dazu sind Elemente ohne eingefügte Nutzerinhalte ohne ein solches Abschlußtag. Auch dazu hier ein Beispiel:

```
<image src="http://athen.regionett.de/test/bild.jpg">
```

Auch hier enthält der Code Daten, die speziell auf den Benutzer zugeschnitten sind, und dennoch erscheinen diese nicht auf dem Bildschirm. Sie sind lediglich für den Webbrowser zur Interpretation und gegebenenfalls Auslösung (in diesem Falle Anzeige des Bildes) bestimmt, und somit natürlich indirekt schon sichtbar.

Dieser grundsätzliche Unterschied zwischen Nutzerinhalten (Contents), die als Text angezeigt werden, und nutzerspezifischen Parametern, welche nicht angezeigt werden (wie die URL des Bildes im obigen Beispiel), wird sich als ausschlaggebend für die Art erweisen, in der das System Daten speichern wird. Wir können an dieser Stelle die Prognose wagen, daß es möglich ist, diese beiden Datenarten in der gleichen Weise zu betrachten und damit in einer homogenen Datenstruktur zu speichern.

---

<sup>3</sup> Das Zeichen &Uuml ; ist ein standardisierter Umlaut-Tag, der vom Browser als „Ü“ dargestellt wird.

Es ist ersichtlich, daß sich demnach gutes HTML aus den vom Nutzer darzustellenden Inhalten, und deren logisch-struktureller Beschreibung zusammensetzt. Die tatsächliche Art der Darstellung, z.B. deren optische Erscheinung liegen beim Browser des Clients.

Im Laufe der Weiterentwicklung von HTML wurde von diesem Konzept oftmals abgewichen. Viele absolute Stilelemente wurden eingebracht um Text zum Beispiel fettgedruckt (<b>) oder kursiv (<i>) erscheinen zu lassen. Die Sprache hatte dabei sehr unter dem Konkurrenzkampf zwischen den Browserherstellern zu leiden. Viele Tags wurden einseitig und außerhalb der Spezifikationen des W3C (World Wide Web Consortium) <sup>4</sup> eingeführt. Oftmals waren diese Gegenstand heftiger Kontroversen und wurden von verschiedenen Systemen falsch oder gar nicht interpretiert. Sogenannte Cross-Browser-Probleme waren die Folge und bestimmten fortan einen beträchtlichen und vollkommen unnötigen Teil der Arbeit von Webdesignern, die plötzlich Seiten mehrmals und unter Berücksichtigung dieser Unterschiede programmieren mußten. Mit der Einführung der optischen Stildefinitionen mittels Style-Sheet-Sprachen 1996 waren diese Dinge überflüssig. CSS (Cascading Style Sheets) machte die freie Definition des Aussehens vieler Tags möglich und die immer konsequentere Durchsetzung der Konventionen des World Wide Web Consortiums brachte einen gefestigten Standard.

### **2.1.3. Metadaten**

Diese Informationen haben im Grunde beschreibenden Charakter. Über die präsentierten Daten werden Stichworte zusammengefaßt, die es zum Beispiel Suchmaschinen ermöglichen sollten, die Inhalte zu katalogisieren und einzuordnen und damit dem Publikum leichter zugänglich zu machen. Eine Arbeitsgemeinschaft namens Dublin Core<sup>5</sup> hat es sich zum Ziel gemacht, einen allgemeinen Standard für Metaangaben zu entwickeln. Werden diese Konventionen eingehalten, können zum Beispiel Suchmaschinen leichter eine Katalogisierung der Seite vornehmen oder unterlassen gegebenenfalls auch diese Einordnung auf Wunsch des Autors. Leider ist die Akzeptanz und Einhaltung dieser Konventionen von Seiten der Suchmaschinen sehr gering.

Weiterhin werden Metadaten zur Steuerung von Softwarekomponenten im Übertragungsweg des Dokumentes genutzt. Es ist möglich, über spezielle Angaben eine automatische, zeitgesteuerte Weiterleitung zu einer anderen URL zu erreichen oder die

---

<sup>4</sup> <http://www.w3.org/Consortium/>

<sup>5</sup> <http://purl.oclc.org/dc/>

Zwischenspeicherung der Seite in Proxy-Servern zu unterbinden um so die Aktualität der übertragenen Daten zu erzwingen.

## **2.2. Aufgaben und Anforderungen**

Das geforderte System muß zu verschiedensten Arten des Umganges mit diesen Daten in der Lage sein.

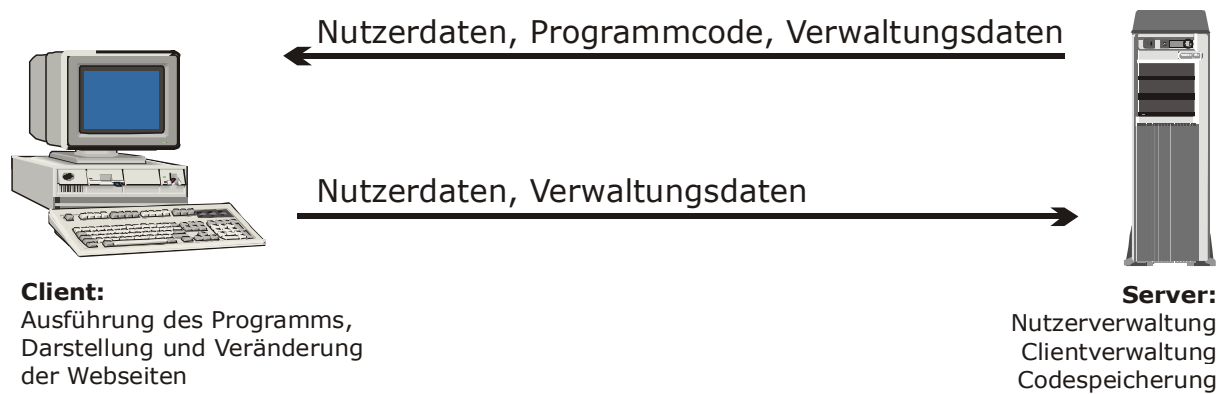
1. Die Nutzer- und Metadaten vom Anwender in komfortabler Weise zu akquirieren
2. Diese mit den Strukturdaten in eine sinnvolle Verbindung zu bringen, d.h. die vom Anwender gewollte, logische Struktur des Dokumentes muß vom System erkannt werden.
3. Speicherung, Verwaltung und Konsistenzsicherung der Daten.
4. Anbieten des Datenbestandes zur Veränderung durch den Nutzer (Editierbarkeit)
5. Erstellung des HTML-Dokumentes aus den Daten

Verschiedene Systementwürfe sind für die Lösung der Aufgabe möglich. Sie unterscheiden sich grundsätzlich im Hinblick auf Ressourcenbedarf (sowohl auf Seiten des Anwenders als auch auf Anbieterseite), Programmieraufwand und zu erwartender Performance.

## **2.3. Lösungsmodelle**

### **2.3.1. Clientbasiertes System**

In diesem Modell übernimmt der Clientrechner den Hauptteil der Arbeit. Lediglich Datenspeicherung und –verwaltung werden auf dem Server ausgeführt. Das hat verschiedene Auswirkungen. Zum einen ist recht hoher Datenverkehr und damit Netzlast zu erwarten. In den Spezifikationen ist gefordert, daß das System ohne Installationsaufwand auf der Nutzerseite arbeiten kann. Das erfordert eine komplexe Javaapplikation, deren Bytecode übermittelt werden muß. Derartige Software ist oft wenig performant und Inkompatibilitäten sind leicht möglich. Die sofortige Lauffähigkeit auf den gängigsten Plattformen ist nicht gewährleistet und der Nutzer muß, abhängig von der Bandbreite seiner Netzverbindung mit hohen Ladezeiten rechnen. Vorteilhaft an dieser Lösung ist, daß die vom Server zu bewältigende Arbeit minimal bleibt.

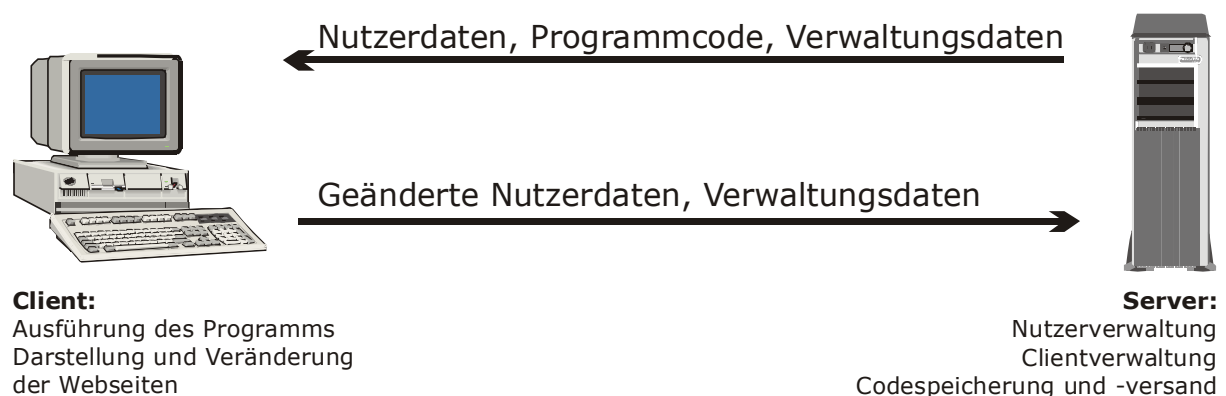


**Abbildung 1 - Clientbasiertes System**

Alternativ ist die Speicherung der Daten in diesem Modell auch auf dem Rechner des Nutzers möglich, was allerdings aus Sicherheitsaspekten heraus wenig vorteilhaft erscheint. Manipulationen der Daten durch Dritte und daraus resultierender Integritätsverlust können nicht ausgeschlossen werden.

### 2.3.2. Serverbasiertes System

Eine weitere Realisierungsmöglichkeit ist ein Modell, in dem die gesamte Arbeitslast auf Seiten des Servers liegt und dieser lediglich reines HTML an den Clienten schickt, welcher dann mit normalen Requests antwortet. Eine derartige Architektur hat den Vorteil der größtmöglichen Kompatibilität zu den verschiedenen Nutzerplattformen. Der Einsatz von Cookies oder Javascript ist sehr wahrscheinlich erforderlich und schränkt die Browserauswahl ein, doch dieser Nachteil ist vergleichsweise gering, ist doch der potentielle Nutzerkreis am heimischen Rechner meist mit Browsern ausgestattet, welche diese Funktionalitäten bieten.



**Abbildung 2 - Serverbasiertes System**

Doch ein solches System bringt auch einige wichtige Nachteile mit sich. Zum einen ist die Arbeitslast fast vollständig auf Seiten des Servers, was diesen stark auslastet und eventuell größere Investitionen in Rechnerkapazität erfordert. Das wiederum beantwortet allerdings auch die Frage nach der Leistungsfähigkeit und Geschwindigkeit. Sie ist allein von diesem Faktor abhängig und somit unter der Kontrolle des Anbieters, respektive Providers (ISP). Optional bietet sich in diesem Modell die Nutzung einer externen Datenbank an.

Zum anderen wird an dieser Stelle deutlich, daß die gestalterischen Elemente vollkommen auf die Möglichkeiten von HTML beschränkt bleiben, was Einschnitte in die Nutzerfreundlichkeit erwarten läßt.

Dennoch setzen sich heute durch immer hochentwickeltere Stylesheets und Multimediasprachen wie Flash derartige Systeme immer mehr durch. Ein gutes Beispiel sind die sehr populären LAMP-Systeme<sup>6</sup>.

### 2.3.3. Eine komplexe Client-Server-Struktur

Ein solches System zeichnet sich durch Programme sowohl auf Client- als auch auf Serverseite aus, die über einen speziellen Socket miteinander kommunizieren. Praktikablerweise sollte eine derartige Struktur in Java realisiert werden. Im Webbrowser des Anwenders läuft ein Java-Applet, welches die Darstellung der Seiteninhalte und deren Editierung übernimmt. Auf Seiten eines Servers ist in diesem Modell ein Java-Servlet oder vergleichbares in Kooperation mit einer Datenquelle eingesetzt. Seine Aufgabe ist es, die Anwender- und Inhaltsdaten aufzubereiten und an den Clienten zu schicken, wo sie verarbeitet und zurückgeschickt werden.

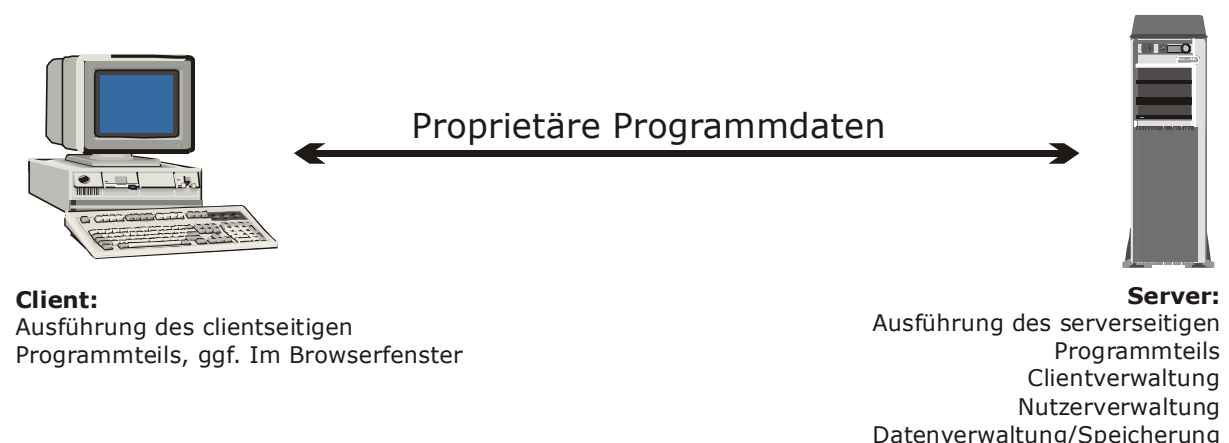


Abbildung 3 - komplexe Client/Server-Struktur

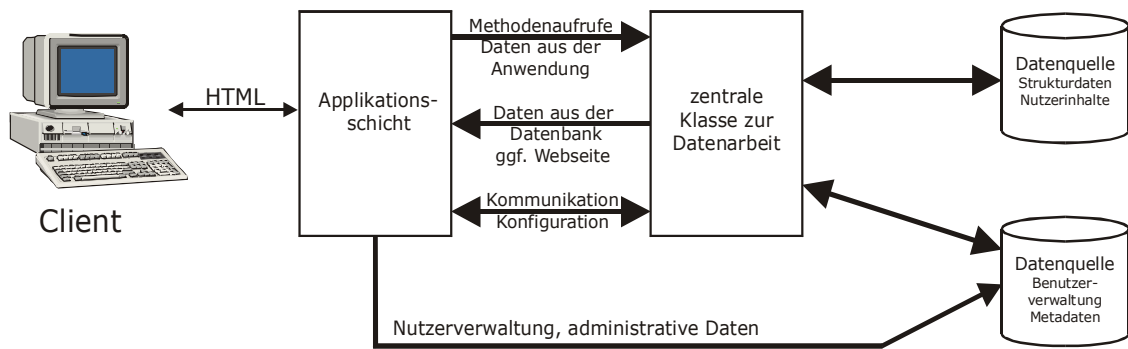
<sup>6</sup> Linux, Apache, MySQL, PHP, umgangssprachliche Abkürzung für eine Serversoftwarekombination

## 2.4. Entwurf

Betriebstechnische Gesichtspunkte ließen im konkreten Beispiel die Wahl auf ein serverbasiertes System unter Verwendung der Applicationserver „Tango2000“ fallen. Das hatte aufgrund mangelnder Java-Kapazität ein fast ausschließlich serverbasiertes System zur Folge.

Das geforderte System sollte sich durch eine flexible, erweiterbare Struktur auszeichnen, die es ermöglicht, verschiedenste Elemente der HTML-Sprache zu integrieren. Auch zukünftige, noch nicht spezifizierte, Tags sollten mit vertretbarem Aufwand integrierbar sein. Aus diesem Grund mußte eine offene Struktur geschaffen werden, die eine klare Trennung zwischen den Nutzerdaten, den Strukturelementdaten und den Programmelementen ermöglicht. Die beste Lösung zu diesem Zweck stellt eine objektorientierte Klassenstruktur im Sinne der Modularisierung dar. Die eigentliche Datenspeicherung und deren Verwaltung, sowie die Generierung der HTML-Seite werden durch eine gekapselte und möglichst autonome Klasse übernommen. Auf dieser Klasse setzt eine Applikationsschicht auf, soweit diese Funktionalität durch die Plattform unterstützt wird. Diese Anwendungsschicht hat die Interaktion mit dem Nutzer zur Aufgabe und steuert die Aktionen der Klasse durch Methodenaufrufe. Das Interface der Klasse sollte möglichst einfach und flexibel sein um sich bei künftigen Erweiterungen des Sortiments an zu verarbeitenden Tags nicht ständig anpassen zu müssen. Wichtigste Aufgabe dieser Klasse ist das Generieren der HTML-Dokumente und die Datenverwaltung. Zu diesem Zweck muß ihr eine zuverlässige und schnelle Datenquelle zur Verfügung stehen. Eine SQL-Datenbank bietet zu diesem Zweck geeignete komfortable Datenverwaltungs- und Konsistenzsicherungsmöglichkeiten.

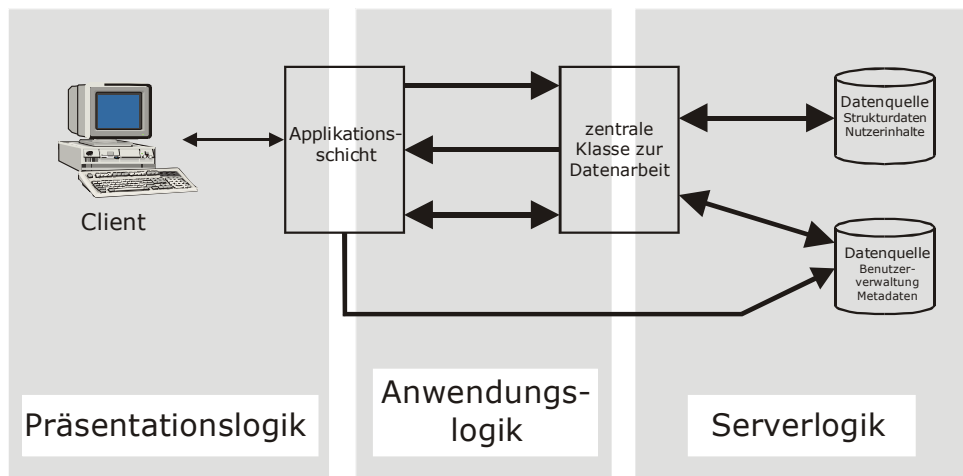
Weiterhin soll es die Aufgabe der Klasse, die den Namen `Seite.tcf` (und die im folgenden Text auch oft der Kürze wegen so bezeichnet ist) erhalten wird sein, wichtige Daten zum Nutzer und dessen Verwaltung zu kapseln. Für jede vom System aktuell bearbeitete Seite muß ein Objekt dieser Klasse instantiiert werden, welches eine Reihe von Initialroutinen durchläuft, in denen es sich auf die kommende Aufgabe vorbereitet und verfügbare Daten aus der Nutzerdatenbank requiriert. So wird es zum Beispiel zweckmäßig sein, den Zielpfad für die zu generierenden HTML-Dateien zu ermitteln und zu speichern. Weiterhin ist es unter Umständen möglich, in der Nutzerdatenbank einen URL-Präfix des betreffenden Nutzers zwischenzuspeichern, falls jener über eine eigene DNS-Domäne verfügt und nicht im Subpfad des ISP gelistet sein möchte. Anschließend wird sie die zu bearbeitende Seite vor der Applikationsschicht repräsentieren und sämtliche notwendigen Funktionalitäten bieten, welche diese zum Editieren benötigt.



**Abbildung 4 - Kommunikationsfluß**

Der Applikationsschicht unterliegt die Kommunikation mit dem Nutzer, respektive dessen Webbrowser, und der eben genannten Klasse. Darüber hinaus wird sie über einen intelligenten Mechanismus verfügen müssen, um mit dem, wie schon erwähnt recht beschränkten, Interface der Klasse agieren zu können. Dabei sind auch mögliches Fehlverhalten und Fehlbedienung von Seiten des Nutzers abzufangen.

Auch wesentliche Teile der Anwendungslogik müssen auf Grund des vereinfachten Interface in der Applikationsschicht untergebracht werden. Im Sinne des Multi Tiered Architecture Modells für im Netz verteilte Anwendungen ergibt sich daraus eine an dem gleichen Schema wie folgt aufgebaute Struktur:



**Abbildung 5 - Schichtenmodell**

Es ist deutlich ersichtlich, daß den einzelnen Schichten des geplanten Systems in dieser Aufteilung meist eine Mittlerrolle zukommt. So ist zum Beispiel die zentrale Seitenrepräsentationsklasse `Seite.tcf` zwar im Wesentlichen für die Anwendungs- und Programmlogik verantwortlich, hat aber gleichzeitig die Aufgabe der intelligenten Datenspeicherung, welche der Datenbankserver nicht allein wahrnehmen kann.

# 3. Verwendete Mittel und Ressourcen

## 3.1. Middleware und Applicationserver

Seitdem Webauftritte in praktikabler und effektiver Weise dynamisch mit Daten aus dem Inneren der Datenverarbeitungsstruktur einer Organisation versorgt werden sollten, wurden Konzepte für die Kommunikation der verschiedenen Softwarekomponenten benötigt.

Der Begriff Middleware ist nicht eindeutig definiert. Gewöhnlich werden darunter lediglich Komponenten niederer Schichten wie z.B. ODBC bezeichnet. Auch CORBA wird als Middleware angesehen. Das definiert die Middleware als eine Softwareebene, welche diesen Datenaustausch in verteilten Anwendungen zum Zweck hat. Eine Middleware zeichnen zum Beispiel folgende Merkmale aus:

- Einsetzbar in offenen Client/Server-Umgebungen
- Netzwerkfähig, ohne Beachtung des dabei verwendeten Protokolls. Im OSI 7-Schichten-Modell rangiert Middleware auf den Schichten 6 und 7
- Betriebssystemunabhängig

Die Programmierung einer solchen Middleware erfolgt über ein genau spezifiziertes API, welches im Sinne der plattformübergreifenden Funktionalität meist möglichst einfach gehalten ist.

Auf ihr aufsetzend kann es ein Applicationserver sein, der die Funktionalitäten der Middleware nutzt, um die verschiedenen Komponenten zu einem funktionalen Ganzen zu verbinden. So kann es zum Beispiel ein solcher Applicationserver sein, der einem Webserver auf Anfrage eines Clients hin die speziell zusammengestellten und formatierten Daten aus einer Datenbank zur Verfügung stellt (z.B. unter Zuhilfenahme der Middleware ODBC), damit dieser sie dann, in eine Webseite eingebettet an den Client übermittelt. Dadurch kann eine höhere Dezentralisierung der Datenverarbeitung und –bereitstellung erreicht werden.

Die verschiedenen Aufgaben in einem Unternehmen können auf speziell entworfene Rechner ausgelagert und verteilt werden, was bessere Performance, Wartung und Ausfallsicherheit zur Folge hat.

## 3.2. Der Applicationserver Tango2000

Der von Pervasive Software<sup>7</sup> angebotene Applicationserver Tango2000 zeichnet sich durch hohe Flexibilität in der Programmierung und Anwendung aus. Einsatzgebiet ist die datenbankgestützte Entwicklung dynamischer Webseiten. Die Programmierung erfolgt über sogenannte Tango Application Files (im folgenden auch schlicht „TAFs“ genannt), welche eine XML-Anwendung sind.

Das Softwarepaket setzt sich aus 3 Komponenten zusammen, auf die im Folgenden näher eingegangen werden soll:

- Tango Editor
- Tango Server
- Tango Web Analyzer

### 3.2.1. Tango Server

Der Tango Server ist das eigentliche Kernstück von Tango. Er fungiert als eine Vermittlerschicht zwischen einem eingesetzten Webserver bzw. dessen Client und den im Unternehmen gespeicherten Datenbankinhalten, dargeboten durch ein Datenbanksystem. Die Funktionsweise läßt sich am besten am konkreten Beispiel der einzelnen Arbeitsschritte erläutern.

1. Ein HTTP-Request eines Clients erreicht den Webserver des Unternehmens. Die URL enthält den Namen eines TAFs und gegebenenfalls Parameter zu dessen Ausführung.
2. Der Webserver erkennt an der Struktur der URL, daß es sich um eine Tango-Applikation handelt. Normalerweise geschieht dies am vorhandenen Suffix `*.taf`
3. Der Server gibt die Anfrage an den Tango Server weiter. Das geschieht entweder durch den Einsatz eines Server Plugins oder durch das Common Gateway Interface (CGI)
4. Der Tango Server nimmt den Request entgegen und führt ein oder mehrere TAFs aus.
5. Datenbankzugriffe über ODBC oder direkt (Oracle) werden ausgeführt, um an die zur Erstellung der dynamischen Seite notwendigen Daten zu gelangen.
6. Die Resultate aus diesen Operationen werden vom Tango Server verarbeitet und in die zu den Actions gehörenden HTML-Stücke (Results HTML) eingefügt. Dabei werden die Daten in vorher vom Programmierer definierte Platzhalter (MetaTags) integriert.

---

<sup>7</sup> <http://www.pervasive.com/>

7. Die verschiedenen HTML-Resultate der Actions werden zu einem einzigen Dokument zusammengefügt und an den Webserver übermittelt.
8. Der Webserver gibt die generierte Seite an den Browser des Nutzers weiter, womit der Arbeitszyklus beendet ist.

Es ist ersichtlich, daß von der Anfrage des Nutzers bis zur Beantwortung zahlreiche Arbeitsschritte getan werden müssen, die multiple Kommunikationsstufen und -schichten, Rechenschritte und damit auch Engpässe enthalten können.

Als zusätzliche Eigenschaft ist Tango mit serverseitigem Javascript ausgestattet, was für viele Operationen von großem Vorteil sein kann.

Der Tango Server ist mit einigen Basisfunktionen ausgestattet, die eine genaue Nutzerabgrenzung (Usertracking) ermöglichen. So wird zum Beispiel für jeden Nutzer ein befristet gültiges Cookie mit einer eindeutigen Identifikatornummer eingerichtet, was zum Beispiel für Datensicherheit erforderlich ist und es dem Server ermöglicht, einem ganz bestimmten Client Variablen in verschiedenen „Scopes“<sup>8</sup> zuzuordnen und bereitzustellen. Dadurch erst kann eine Webapplikation dieser Art ganz konkret auf einen Nutzer reagieren und auf seine Bedürfnisse zugeschnitten werden.

Die Kommunikation mit dem Webserver ist, wie schon erwähnt, auf zwei Wegen möglich. Mittels CGI oder als Plugin des Webserver. Der wichtigste Unterschied dabei ist, daß CGI als eigenständige Vermittlerschicht zwischen Tango und Webserver agiert, während das Plugin zu einem Teil des Webserver selbst wird<sup>9</sup>. Das hat, gegenüber der CGI-Lösung erhebliche Performancevorteile, da die Kommunikation deutlich schneller möglich ist. Allerdings ist laut der Nutzerdokumentation ein solches Plugin derzeit lediglich für Netscapes Webserver NSAPI und für den Internet Information Server (IIS) der Microsoft Corporation erhältlich. Wesentlich gebräuchlichere Server, allen voran der populäre „Apache“<sup>10</sup>, sind nicht auf diese Weise ansteuerbar, was schwer nachzuvollziehen ist, da eben der Apache für diese Art der Anwendung konzipiert wurde und Pervasive Software offenbar sehr wohl über das nötige Know How verfügt, da der Tango Webanalyzer, auf den später näher eingegangen werden soll, als Plugin realisiert ist.

---

<sup>8</sup> siehe dazu auch Kapitel 3.2.1. „Der Tango Editor“

<sup>9</sup> Nutzerdokumentation „T2KUsersGuide.pdf“, Seite 6

<sup>10</sup> <http://www.apache.org/>

An dieser Stelle muß ich erwähnen, daß für den Apache sehr wohl ein derartiges Plugin vorhanden ist und ganz normal bei der Installation des Servers eingebunden wird. Warum die Dokumentation in einem derart relevanten Punkt nachlässig ist, ist mir unbekannt.

Die Konfiguration des Servers erfolgt einfach, schnell und arbeitsplatzungebunden über eine Anwendung desselben. Die sogenannte `config.taf` ermöglicht es dem Administrator, sämtliche Parameter über ein Browserfenster festzulegen und zu ändern.

Darüber hinaus verfügt Tango seit seiner letzten Erweiterung zu Tango2000 über einen Server Watcher Process, einen eigenständigen Task, der die Funktionsweise des Servers überwacht, und ihn im Falle einer Fehlfunktion neu startet.

Ein weiterer Bonus ist die Fähigkeit zum Ausführen von sogenannten Cron-Jobs. Wie der vom dem UNIX-Taskplaner `cron` abgeleitete Name vermuten läßt, ist es damit möglich, zu frei wählbaren Zeitpunkten einzelne TAFs ausführen zu lassen. Damit ist zum Beispiel das tägliche Warten einer Datenbank oder das automatische Versenden von Statusmeldungen an den Administrator über eMail möglich.

### **3.2.2. Tango Editor**

Der Tango Editor ist eine Entwicklungsumgebung zur Erstellung und Verwaltung der Tango Application Files, die den Tango Server programmieren. Der Editor ist eine Anwendung im Sinne von RAD (Rapid Application Development). Er verfügt über eine komfortable Oberfläche, die es ermöglicht, durch schlichtes Drag-and-Drop und den Einsatz vorgefertigter Komponenten schnell zu einer funktionsfähigen Webapplikation zu kommen. Dieser Komfort geht allerdings zu Lasten der auf diese Weise erreichbaren Funktionalität.

Alle Funktionen des Servers werden als Actions bezeichnet. Jede dieser Actions kann mit Resultierendem HTML-Code (Results) versehen werden, der für die genaue Antwort der Action entscheidend ist. Dabei wird zwischen 3 Arten dieser Results unterschieden.

#### **1. Result HTML**

Wird erzeugt, nachdem die Datenbank die gewünschten Ergebnisse lieferte.

#### **2. NoResults HTML**

Dieser Code wird bei ergebnislosen Abfragen eingefügt

#### **3. Error HTML**

Gibt die eventuellen Fehlermeldung der ODBC-Treiber formatiert aus.

Es werden die meisten der üblichen Basisfunktionen eines derartigen Systems über diese Oberfläche angeboten.

- Datenbankoperationen
- Sogenannte Record Builder – Zusammenstellungen und Vereinfachungen mehrerer üblicher Datenbankoperationen (Datensatz unter bestimmten Gesichtspunkten einfügen oder extrahieren). Ohne SQL-Kenntnisse bedienbar.
- Results
- Mailoperation
- Dateioperationen
- Programmstrukturen, Verzweigungen, Schleifen
- Integration externen Codes über Java Beans, Javascript, Tango Class Files, Perl und Shell-Scripte etc.

Viele der vom Tango Server verstandenen Kommandos sind allerdings nur über sogenannte Meta Tags erreichbar. Das sind spezielle, in den HTML-Code eingebettete Tags, die vom Server ausgewertet und durch konkrete Ergebnisse, z.B. Datenbankinhalte oder Variable, ersetzt werden. Durch diese Beschränkungen kommt es leicht zu Inkonsistenzen im resultierenden Programm, das die meisten Lösungen auf völlig verschiedenen Wegen mit oft sehr unterschiedlicher Effizienz erzielt werden können. Oft ist es nur recht umständlich möglich, alles über die graphischen Elemente des Editors zu implementieren. Wenn dann jedoch für einzelne Aktionen nur MetaTags geeignet sind, findet das Kodieren quasi auf zwei verschiedenen Ebenen statt, was der Transparenz des Ergebnisses sehr im Wege steht.

Der Editor zeichnet sich durch eine genaue Abbildungsmöglichkeit der dem Server zur Verfügung stehenden ODBC-Datenquellen aus. Genau wie dem Tango Server an sich, stehen auch dem Editor Datenquellen zur Verfügung. Es ist auch möglich, diese Quellen während der Entwicklungszeit umzuleiten, das heißt, daß eine Action, die während des Einsatzes mit einer ganz bestimmten Datenquelle korrespondieren soll, in der Entwicklungszeit auf eine andere Quelle zeigt, was die Datensicherheit im Unternehmen bei Testeinsätzen sicherstellt. Auch die direkte Eingabe von SQL-Statements ist möglich, wobei Transaktionen simuliert werden können.

Weiterhin bietet der Editor objektorientierte Programmieransätze. Es ist möglich, Klassen zu bilden, sogenannte Tango Class Files (TCFs). Diese simplen Klassen erlauben Instantiierung multipler Objekte mit getrennten, gekapselten Daten und unabhängigen Methodenaufrufen.

Auch diese Klassen und Ihre Methoden können via Drag-and-Drop in die TAFs eingebunden und genutzt werden. Datenkapselung im Sinne der objektorientierten Programmierung ist in diesem System nicht nötig, da zwar Membervariablen existieren, diese aber in keinem Falle von außen eingesehen werden können. Lediglich Methodenaufrufe sind möglich. Alle der Klasse bekannten Daten können also, ins Objektmodell übertragen dargestellt, als `private` betrachtet werden.

Datenkapselung ist weiterhin über sogenannte Scopes möglich. Das sind Geltungsbereiche von Variablen, die von global und zeitlich unbegrenzt bis auf das einzelne TAF oder die Methode eines Objektes begrenzt reichen. Ein Beispiel: Auf eine Variable, die in einem TAF als „`local`“ definiert wird, kann in einer weiteren Instanz des gleichen TAFs nicht zugegriffen werden. Wird sie als „`user`“ definiert, ist sie von jedem TAF, das im Browser eines bestimmten Nutzers ausgeführt wird, zugänglich.

Alternativ ist es mit Tango2000 auch möglich, eine Variable im Scope „`cookie`“ zu speichern. Das hat den Effekt, daß vom Server im Browser des Client ein Cookie<sup>11</sup> angelegt wird, welches aus den TAFs heraus wie eine Variable behandelt werden kann. Der ganze Vorgang des Abspeicherns und Zurückladens wird vor dem Programmierer komplett verborgen. Natürlich sind in dieser Weise abgelegte Daten alles andere als sicher. Nutzer können Cookies löschen oder deren Aufnahme ganz verweigern, sofern Ihr Browser überhaupt zu derartiger Funktionalität in der Lage ist, was auch heute noch nicht als selbstverständlich gelten sollte. Dennoch ist es eine einfache und praxistaugliche Methode, schnell und unkompliziert Nutzer wiederzuerkennen oder deren Verhalten auf der Site zu analysieren, was dem Unternehmen unter Umständen helfen kann, genauer auf die Wünsche der Nutzer einzugehen. Zu diesem Zwecke ist es natürlich erforderlich, diese Intention in der Webapplication zu reflektieren. Das heißt, daß zusätzliche dynamische Komponenten erforderlich sind, ohne die Konsistenz der Site zu gefährden, die nach wie vor einen in sich geschlossenen Eindruck machen muß. Doch zurück zu Tango.

Die vom Tango Editor erzeugten Dateien (TAF und TCF) sind XML-Anwendungen, das heißt, sie stellen eine eigenständige, nach den XML-Spezifikationen des W3C<sup>12</sup> entwickelte Sprache dar und sind somit, genau wie HTML eine SGML-Anwendung. Ein wesentlicher Vorteil dieses Konzeptes ist die einfache Erweiterbarkeit der resultierenden Sprache durch eigene Meta Tags (Custom Meta Tags). Ein weiterer Bonus ist die Möglichkeit, durch diese

---

<sup>11</sup> Begriffserklärung „Cookie“ siehe Anhang

<sup>12</sup> <http://www.w3.org/TR/REC-xml>

Struktur TAFs auf verschiedenen Plattformen laufen zu lassen, ohne irgend etwas neu kompilieren oder anpassen zu müssen. In der Praxis allerdings trifft man dabei auf gewisse Hindernisse.

Das erste dabei ist die Einbindung externen Codes, was nur in bestimmten Abstufungen möglich ist. Theoretisch können z.B. Java-Beans auf allen Tango-Plattformen gleich eingebunden werden, wohingegen das Integrieren von DLLs<sup>13</sup> oder COMs auf Windows-Maschinen beschränkt bleibt, und nur Mac-Nutzer ihre Programme mit Macintosh Events bereichern können.

Ein weiteres Problem ist das Nutzen von Actions, die auf Betriebssystemfunktionalität zurückgreifen, wie die File-Action. Hier müssen Dateipfade und -namen so angegeben werden, wie es für das betreffende OS spezifiziert ist. Daher müssen auch solche Angaben beim Portieren angepaßt werden.

Generell problematisch beim Entwickeln serverbasierter Anwendungen ist die Fehlersuche. Debugging. Tango löst dieses Problem durch einen Debug-Modus im einzelnen TAF oder TCF. Ist dieser Modus aktiviert, werden zusätzlich zum den normalen HTML-Results alle ausgeführten Actions und die Antwort des Servers darauf aufgelistet, eingeteilt nach den Kategorien „Erfolgreich“, „Warnung“ und „Fehler“. Dieses System ist durchaus zur Fehlersuche geeignet, aber nicht mit den komfortablen Debug-Optionen einer integrierten Entwicklungsumgebung zu vergleichen. Besonders bei Verzweigungen unter den TAFs, ein Fall in dem bei jedem einzelnen die Debug-Funktion aktiviert sein muß, ist es sehr schwer, den Überblick zu wahren.

### **3.2.3. Der Tango Web Analyzer**

Die Dritte Komponente der Tango2000-Software ist der Web Analyzer (im folgenden Text als Tango WA oder schlicht WA bezeichnet), in früheren Releases auch „Bolero“ genannt. Obwohl der Web Analyzer im vorliegenden konkreten Beispiel keine Rolle spielt, soll die Funktionsweise umrissen werden, da ein solches System bedeutend für den Einsatz von Application Servern für große Anwendungen ist.

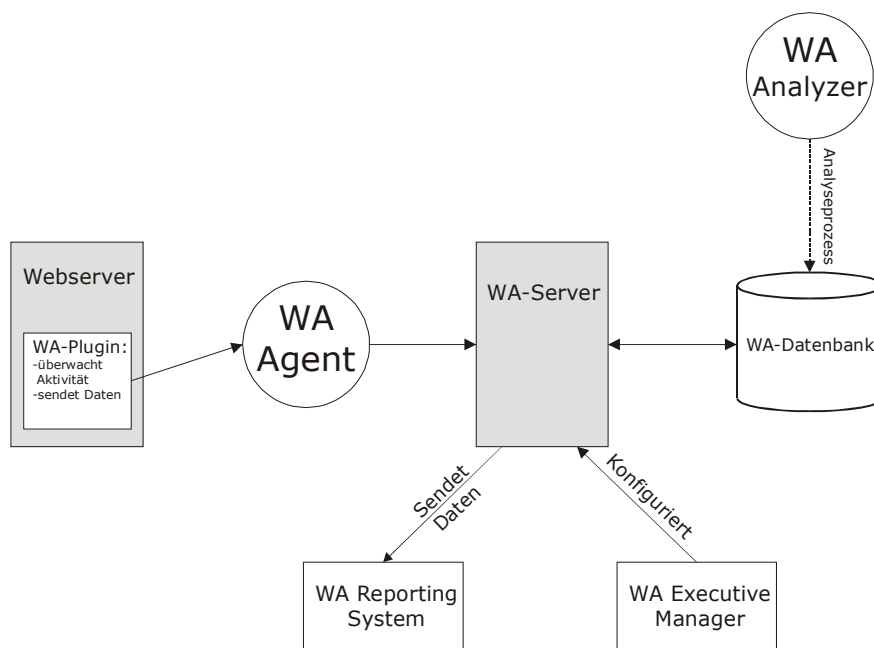
Tango WA ist eine Hilfestellung für den Systemadministrator. Es sammelt ständig Daten über Netzlast und Auslastung eines Webservers und stellt sie in einer SQL-basierten Datenbank zur Verfügung. Auch das funktioniert mittels einer Client-Server-Architektur. Es folgt der Modellvorstellung eines Data-Warehouse-Systems. Das Programm besteht aus folgenden Komponenten:

---

<sup>13</sup> Dynamic Link Library – siehe Glossar

- Tango WA Plugin
- Tango WA Agent
- Tango WA Server
- Tango WA Analyzer
- Tango WA Importer
- Tango WA database scheme scripts
- Tango WA Executive Manager
- Tango WA Reporting System

Wie schon im Abschnitt 3.2.1. erwähnt, ist ein Teil des WA als Plugin in den Webserver integriert. Dieser Teil agiert im Sinne von WA als Client. Ein WA-Server läuft ebenfalls (ggf. auf einer anderen Maschine) und erhält vom Client, der als Plugin direkt mit dem zu überwachenden Webserver verbunden ist, Daten, die er dann wiederum in der SQL-Datenbank ablegt.



**Abbildung 6 - WebAnalyzer-Struktur**

Diese Daten können auch auf einem Umweg gesammelt werden. Durch den WA Agent. Dieses Programm ist eine Anwendung im Sinne des Agentenkonzeptes. Anpassbar und fehlertolerant stellt er die sichere Verbindung zwischen Client und Server dar. Überwacht den Informationsfluß und formatiert bzw. filtert die Daten. Der WA-Server nimmt sie entgegen und legt sie in einem genauen Schema in der Datenbank ab. Er fungiert als der

Transaktionsmanager dieses Data-Warehouse-Systems. Dort gesichert, nimmt in regelmäßigen Intervallen der WA Analyzer seine Arbeit auf. Seine Aufgabe ist es, diese Daten zu analysieren und in aufbereiteter Form, zusammen mit den Ergebnissen der Analyse zur Verfügung zu stellen. Die Informationen können wahlweise nach Nutzern, aufgerufenen Seiten, Parametern, Aktivitäten und sogar geographischer Herkunft gefiltert werden. Diese Informationen werden dann wiederum in der Datenbank abgelegt.

Nützlich für den Anwender des Systems werden sie durch das WA Reporting System. Dieses Programm ermöglicht den Zugriff auf die gesammelten Werke des WA Analyzer über einen Webbrowser, wo sie visualisiert dargestellt werden. Ähnlich funktioniert auch die Konfiguration des WA Servers, eine Aufgabe, die dem Tango WA Executive Manager zufällt. Dieses Konfigurationstool ermöglicht es dem Administrator – vergleichbar mit der `config.taf` beim Tango Server - die gesamten Einstellungen über das Netzwerk vorzunehmen.

Doch auch die Analyse und Visualisierung von Serverlogbüchern, die nicht vom WA Plugin gesammelt wurden, ist möglich. Diese Importierung nimmt der Tango WA Importer vor. Er unterstützt Logfiles aller gängigen Formate. (IIS-Standard<sup>14</sup>, W3C, CLF<sup>15</sup>, Netscape Log Format).

### **3.3. Pervasive.SQL**

Der zusammen mit Tango von Pervasive Software angebotene relationale Datenbankserver „Pervasive.SQL“ (ja, der Punkt ist beabsichtigt), früher bekannt unter dem Namen „Btrieve“, ist für die Zusammenarbeit mit Tango konzipiert und wird auch zusammen damit vertrieben. Ohne einen solchen Server ist die Arbeit mit Tango zur Bedeutungslosigkeit degradiert. Alle wichtigen Funktionen beruhen einzig auf der Zusammenarbeit mit einem Datenbanksystem. Das bedeutet allerdings nicht, daß Tango nicht auch mit anderen Datenbanksystemen kooperieren könnte. Nur ist die Integration der Komponenten bei Verwendung von Pervasive.SQL deutlich weiter fortgeschritten. Werkzeuge wie zum Beispiel das Pervasive Control Center sind einfach vom Editor aus aufzurufen

Der Kern des Datenbankservers setzt sich aus 2 wesentlichen Komponenten zusammen:

---

<sup>14</sup> Internet Information Server – Standard (Microsoft Corporation)

<sup>15</sup> Common Log Format, ein Standardformat für Serverlogdateien

### **3.3.1. MicroKernel Database Engine (MKDE)**

Dieser Teil von Pervasive.SQL ist für die Nutzung über MicroKernel Module zuständig. Er erlaubt es, Btrieve und MicroKernel API Zugriffe auszuführen und ist je nach Softwareversion in verschiedenen Ausführungen vorhanden. Er ermöglicht einfache lokale Zugriffe und Applikationen auf einem Server, der sich auf dem selben Computer befindet.

Die MKDE stellt die unterste Schicht der Anwendung dar. Auch die SRDE setzt auf dieser Schicht auf. ODBC-Routinen verbinden sich mit der MicroKernel-Architektur und ermöglichen Zugriff auf die Netzwerkschichten.

### **3.3.2. SQL Relational Database Engine (SRDE)**

Die Teil des Servers ist der im vorliegenden Beispiel relevante. Er erlaubt Zugriffe über die ODBC-Middleware und stellt so die Grundlage für die Entwicklung einer datenbankgestützten Applikation mit Tango dar. Datenbankzugriffe müssen nicht länger lokal erfolgen, sondern von irgendeinem Punkt im Netzwerk aus. Die Daten stehen praktisch internetweit zur Verfügung. In der Praxis wird diese Verfügbarkeit aus Sicherheitsgründen selbstverständlich an einer Firewall enden, doch in der Theorie ist es möglich. Clients für die ODBC-Arbeit sind für Windows 95, 98 und NT, sowie für Novell NetWare verfügbar. Diese Clients ermöglichen den Zugriff auf die Datenbank von einem Remoterechner aus. Lokale Zugriffe sind dagegen immer möglich, das heißt auch auf Linux und Macintosh.

### **3.3.3. Pervasive Database Utilities**

Zusammen mit dem Server werden zahlreiche Utilities installiert, allen voran das Pervasive Control Center (PCC ). Dabei handelt es sich um ein Warehouse-Tool, welches transparenten und unkomplizierten Zugriff auf alle registrierten Pervasive SQL-Server im Netz ermöglicht. Alle wichtigen Operationen sind damit möglich, vom Einstellen der Sicherheitskonventionen bis hin zum Design oder auch dem Löschen von Tabellen. Leider hat sich diese Art des Zugriffs in der Praxis nicht immer als stabil und zuverlässig erwiesen. Oft werden Befehle vom PCC nicht oder nicht wie gewünscht ausgeführt und lassen die Datenbank nicht selten in einem defekten Zustand zurück, was während der Entwicklungsphase zahlreiche Neustarts und Wartungsarbeiten des Servers erforderte. So bietet PCC zum Beispiel die Möglichkeit an, die Struktur von Tabellen nachträglich zu ändern, selbst wenn schon Datensätze enthalten sind. Diese Funktion wurde unter verschiedenen Konfigurationen getestet und war nie erfolgreich. Ein Absturz des PCC war stets die unangenehme Folge.

Ein im PCC integriertes Tool namens SQL Data Manager ist das wichtigste für die praktische Arbeit. Es ermöglicht Zugriff auf die Tabellen und die darin enthaltenen Daten. Sie können manipuliert, gelöscht und ergänzt werden. Die direkte Eingabe von SQL-Queries ist ebenfalls möglich, sowie die verschieden formatierte Ausgabe von deren Ergebnissen.

Das PCC ist in der Lage, Wartungsoperationen an der Datenbank vorzunehmen. Die Datenintegrität von Tabellen und die referentielle Konsistenz der Schlüssel kann jederzeit nach verschiedenen Parametern überprüft werden.

### 3.4. Die eingesetzte Hardware

Im praktischen Beispiel dieser Arbeit waren Macintosh-Rechner PPC 8600 von Apple<sup>16</sup> im Einsatz, ausgestattet mit dem Betriebssystem Mac OS 9. Diese sind bestückt mit 250MHz PowerPC Prozessoren (604eV) und 320MB RAM. Auf ihnen laufen diverse Internetdienste des ISP<sup>17</sup> wie DNS, ftp und Mailserver und natürlich der Tango2000-Server. Als Webserver ist die WebStar® Server Suite von Starnine<sup>18</sup> in der Version 4.1 installiert.

Der Einsatz von Mac OS brachte leider diverse Einschränkungen bezüglich der realisierbaren Technologien mit sich. Die an sich wünschenswerte Java-Kapazität scheiterte an der Tatsache, daß für Mac OS nur eine Java 1 Runtime Engine ohne SDK<sup>19</sup> erhältlich war, was aus heutiger Sicht ein wenig unangemessen scheint. Auch Java-Servlet-fähige Webserver bzw. Plugins, wie zum Beispiel „Tomcat“<sup>20</sup> oder der mit entsprechenden Modulen bestückte „Apache“ waren nicht einsetzbar. Diese Umstände hatten entscheidenden Einfluß auf die Wahl des rein serverbasierten Architekturmodells für die Entwicklung des Systems.

Mittlerweile hat Apple diesen Mißstand nach eigenen Angaben in der neuesten Version des Macintosh-Betriebssystems „Mac OS X“ behoben. Eine Java2 Runtime Engine ist nach Angaben des Herstellers direkt im Betriebssystem integriert.

---

<sup>16</sup> <http://www.apple.com>

<sup>17</sup> Internet Service Provider, siehe Glossar

<sup>18</sup> <http://www.starnine.com/>

<sup>19</sup> Software Developer Kit – Im Gegensatz zur reinen Runtime-Implementierung von Java ermöglicht erst dieser Satz von Entwicklerwerkzeugen die tatsächliche Erstellung neuer Java-Klassen.

<sup>20</sup> <http://jakarta.apache.org/tomcat/index.html>

Die Entwicklungsarbeit fand, über das Internet verbunden, wechselweise an Windows NT 4 und Macintosh-Rechnern statt. Weiterhin befindet sich ein Linux-Rechner (SuSE 6.2) im LAN, auf welchem Pervasive.SQL installiert ist. Dieser sollte während der Entwicklungszeit als Datenbankserver eingesetzt werden.

# 4. Realisierung

## 4.1. Nutzerverwaltung

Ogleich diese Daten nicht direkt der Dokumentenstruktur zugeordnet werden können, sind sie doch Teil des Gesamtsystems und müssen in der Datenbank abgelegt werden. Dieser Umstand findet schon im Kapitel 2.4. „Systementwurf“ Erwähnung, da auch die zur Arbeit mit der Datenbank verantwortliche Klasse mit den Daten der Benutzerverwaltung korrespondiert. Die Benutzerverwaltung enthält sämtliche erforderlichen Informationen zur Identifikation und Behandlung eines einzelnen Nutzers. So zum Beispiel seine Adresse, seinen Nutzernamen, sein Paßwort und alle von ihm angelegten Seiten. An dieser Stelle ist es nötig, festzulegen, wie die eigentliche Art aussieht, in der das System mit Seiten als solche umgeht, wie also die Datenhierarchie aufgebaut ist.

Die SQL-Tabelle, welche dieses Ziel im vorliegenden System realisiert, trägt den zweckdienlichen Namen „Nutzer“ und hat den folgenden Aufbau:

<b>Datentyp</b>	<b>Bezeichnung</b>
identity	UserKey
varchar(10)	Nutzer
varchar(10)	Passwort
varchar(30)	Vorname
varchar(30)	Nachname
varchar(50)	Email
varchar(200)	Pfad
varchar(200)	URLPrefix

Der Datentyp „identity“ ist eine nicht in der SQL-Norm spezifizierte Implementierung von Pervasive.SQL. Er beinhaltet gewissermaßen ein Makro für einen Primärschlüssel, bestehend aus einem Integerwert und dem damit verbundenen eindeutigen Schlüssel. Anhand dieses UserKeys wird der Benutzer fortan vom System identifiziert. Die anderen Daten erklären sich weitgehend von selbst.

Aufgrund der relativ geringen Sicherheitsrelevanz des Systems wird auch das Paßwort des Nutzers im Klartext in der Datenbank gespeichert. Ein potentieller unerwünschter Eindringling, der bis hierher, in die Datenbank, vordringt, hat bedauerlicherweise auch Zugriff zu den Nutzerinhalten und Konstruktionsdaten und benötigt das Paßwort nicht. Allerdings ist dieser Fall sehr unwahrscheinlich, denn normalerweise verfügt der SQL-Server über keine direkte Verbindung in das Internet und kann von außerhalb des Unternehmens-LANs nicht erreicht werden.

Der abgespeicherte Pfad entspricht dem Verzeichnis auf dem Server, in dem die fertig generierten Seiten des Nutzers abgelegt werden sollen. Dabei ist natürlich zu beachten, daß dieser Pfad innerhalb des vom Webserver erreichten Pfades liegen muß, damit er für jenen zur Anzeige der fertigen Seite zugreifbar ist.

An dieser Stelle treten bedauerliche Plattformeinschränkungen auf, denn hier muß der Pfad in der betriebssystemspezifischen Weise (z.B. mit Backslashes (\) für Windows- oder trennenden : für Macintosh-Server) angegeben werden, um dem Tango Server das Schreiben zu ermöglichen. Dieser Umstand macht es leider unmöglich, in dieser Art leicht plattformübergreifend zu entwickeln, da diese Informationen ständig angepaßt werden müssen.

## 4.2. Seitenverwaltung und Metadaten

„Athen“ wird eine Anzahl von Seiten zu verwalten haben, und aus jenen HTML-Dokumente generieren. Die Konstruktionsdaten an sich werden aus Gründen der Übersichtlichkeit in separaten Tabellen gespeichert.

Ihre Verwaltung hingegen erfolgt in einer gemeinsamen Tabelle „Seiten“, welche den folgenden Aufbau hat:

<b>Datentyp</b>	<b>Bezeichnung</b>
identity	SeitenID
varchar(60)	Titel
integer	Eigner
varchar(60)	Dateiname
varchar(60)	Stylesheet
varchar(60)	Backpic

varchar(20)	Meta1Name
varchar(100)	Meta1Wert
varchar(20)	Meta2Name
varchar(100)	Meta2Wert
varchar(20)	Meta3Name
varchar(100)	Meta3Wert
varchar(20)	Meta4Name
varchar(100)	Meta4Wert

Wie im Kapitel 2.1. festgestellt wurde, lassen sich die in HTML-Dateien vorhandenen Informationen in 3 Kategorien aufteilen. Nutzerdaten, Strukturdaten und Metadaten. Alle diese Informationen müssen von dem geforderten System gespeichert und wiedergegeben werden können. Um dies zu erreichen, ist es zunächst erforderlich, die Konstellationen, in denen eben diese Daten in HTML-Dokumenten vorkommen können, etwas näher zu beleuchten.

Die Einordnung der Metadaten ist dabei die simpelste. Sie sind laut Spezifikation des W3C im Header der Datei untergebracht<sup>21</sup>, das heißt, sie sind an genau bestimmter Stelle zwischen den Tags <HEAD> und </HEAD>. Da umfangreiche Metaangaben für die Zielgruppe des Systems nur von bedingter Relevanz scheinen, wurde entschieden, diese in festgelegter Anzahl in der Datenbank zu verankern. Der Nutzer hat für jede Seite die Möglichkeit, maximal 4 Metaangaben zu bestimmen, die von dem Programm in die Seite zu integrieren sind. Diese Daten werden in den Feldern Meta#Name und Meta#Wert gespeichert.

Diese starre Lösung befreit den Server von langwierigen Verwaltungsaufgaben für Metaangaben und spart wertvolle Kapazitäten. Darüber hinaus kann gegebenenfalls in späterer Entwicklungsphase dieses System angepaßt werden, um auch mehr Metadaten erfassen zu können, wenn dies nötig werden sollte.

Das Feld „SeitenID“ ist wiederum vom Typ „identity“ und ist der primäre Bezeichnerschlüssel der Seite.

Das Element mit dem Namen „Eigner“ ist ein Verweis auf den Identifikator des Nutzers, von dem die betreffende Seite angelegt wurde, gewissermaßen seine Nutzernummer.

„Titel“ schließlich bezeichnet, wie der Name schon andeutet, den Titel der Seite. Dabei handelt es sich um denjenigen Titel, der im Inneren des <TITLE> Tags im Kopf (<HEAD>)

---

<sup>21</sup> <http://www.w3.org/TR/html4/struct/global.html#h-7.4.4>

der Seite angegeben und in der Kopfleiste des Browserfensters erscheinen wird und nicht um eine Überschrift oder ähnliches.

Eine weitere für die Seite allgemein relevante Information ist der Name eines eventuell vom Nutzer gewählten Hintergrundbildes. Dieser Name wird im Feld `Backpic` abgelegt.

Die Spalte „Dateiname“ enthält sinnigerweise den Dateinamen der zu generierenden Seite. Diese Angabe erfolgt ohne Pfad, da dieser ja schon in der Nutzertabelle angegeben ist, wo er vom ISP spezifisch für jeden Anwender festgelegt wird. Jeder Nutzer wird also alle seine Seiten in einem Pfad auf dem Laufwerk des Servers haben. Unterverzeichnisse sind dabei nicht vorgesehen. Es wird die Aufgabe der generierenden Seitenrepräsentationsklasse sein, aus dem Pfad und dem Dateinamen den vollständig qualifizierten Namen zusammenzustellen. Für eine Erklärung der Spalte `stylesheet` nehme ich mir die Freiheit, den geneigten Leser auf das Kapitel „4.5. Stildefinitionen“ zu verweisen.

## **4.3. Die Möglichkeiten zur Abbildung der Dokumentenstruktur in der Datenbank**

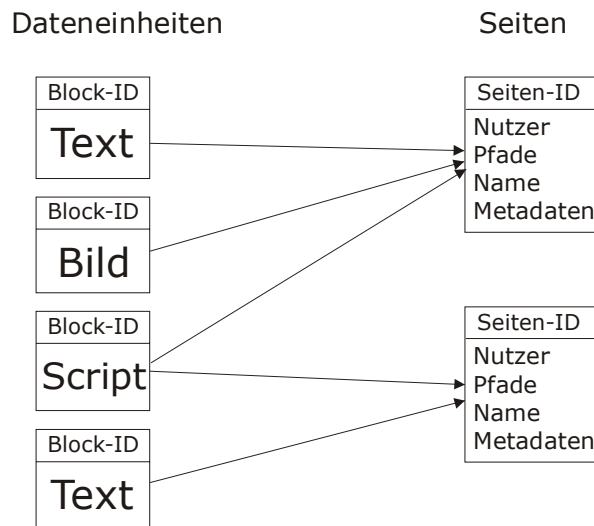
### **4.3.1. Blockbasierte Datenhierarchie**

Zunächst muß bestimmt werden, nach welchem Konstruktionsprinzip die Seiten erstellt werden, wie die Datenhierarchie aussieht und in welcher Reihenfolge was generiert wird.

Vorstellbar ist zum Beispiel ein Aufbau, der jeden einzelnen Datenblock eindeutig identifiziert und im nachhinein zu Seiten zusammenfaßt. Eine solche Struktur ermöglicht mehrfach verwendbare Blöcke. Bilder, Werbetexte oder Scripts wie die beliebten Seitenbesuchszähler können mehrfach von verschiedenen Seiten genutzt werden und müssen dennoch nur ein einziges Mal in der Datenbank abgelegt werden. Primärschlüssel<sup>22</sup> ist also der Bezeichner des einzelnen Blocks (`Block-ID`), welcher eine einzelne Dateneinheit darstellt, aus deren Zusammenstellung sich schließlich eine Seite ergibt.

---

<sup>22</sup> Primäres Identifikationsmerkmal innerhalb einer Datenbankstruktur, zum Beispiel eine eindeutige ID-Nummer.



**Abbildung 7 - Blockbasierte Datenstruktur**

Auch eine leichte Kopierbarkeit einzelner Blöcke in andere Seiten erfordert in einem solchen System nicht mehr als das Einfügen eines neuen Zuordnungsschlüssels. Dennoch ist die praktische Umsetzung eines solchen Konzeptes mit zahlreichen Unwägbarkeiten verbunden. Zum einen ist es erforderlich, zahlreiche wiederverwendete Elemente mit Parametern zu versehen, welche auf den jeweiligen Nutzer und Besitzer der Seite zugeschnitten sein müssen. Das erfordert, ebenso wie die Wiederverwendung an sich, zusätzlichen Verwaltungsaufwand und unterwandert obendrein das Konzept, gleiches mehrfach zu nutzen. Eine strikte (logische) Trennung zwischen den Nutzerdaten und den Strukturinformationen innerhalb des Blocks würde das Problem zumindest teilweise lösen, stellt aber seinerseits einen enormen rechentechnischen Aufwand dar. Ob diesen ein Application Server dieses Typs zu bewältigen vermag, muß fraglich bleiben. Schließlich müssen die Seitenelemente aus zwei logischen Datenbanken requiriert, zusammengestellt und anschließend noch zu einer Seite verknüpft werden. Auf Grund der verschachtelten Struktur von HTML-Seiten ist dieser Vorgang idealerweise rekursiv zu gestalten, was noch mehr Aufwand mit sich bringt. Erfahrungsgemäß kann einem Server, dessen Performance auf kurze, interaktionsreiche Anfragen hin ausgelegt ist, ein derart komplexer Programmablauf kaum zugemutet werden.

Es folgt ein einfaches Schema der Konstruktion einer derartigen Seite.

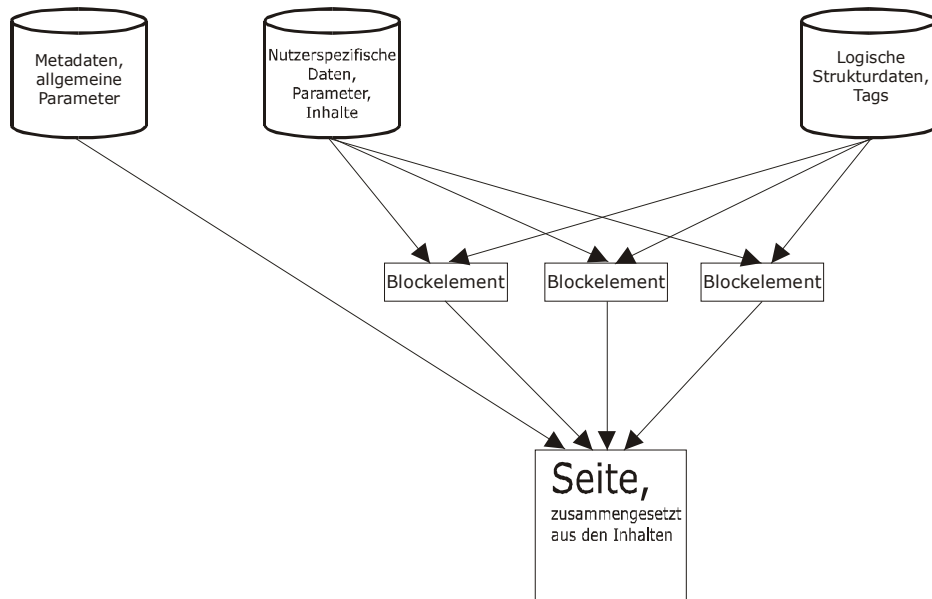


Abbildung 8 - Blockbasierte Seitenkonstruktion

### 4.3.2. Seitenbasierte Datenhierarchie

Eine naheliegende Alternative zu dem oben genannten Modell stellt eine seitenbasierte Hierarchie dar, bei welcher der Primärschlüssel zur Verwaltung der Daten die einzelne Seite identifiziert. Von dieser aus wird dann auf ebenfalls gespeicherte Blockelemente verwiesen. Ein einfacheres, aber auch effizienteres System, für welches sich im vorliegenden Beispiel entschieden wurde.

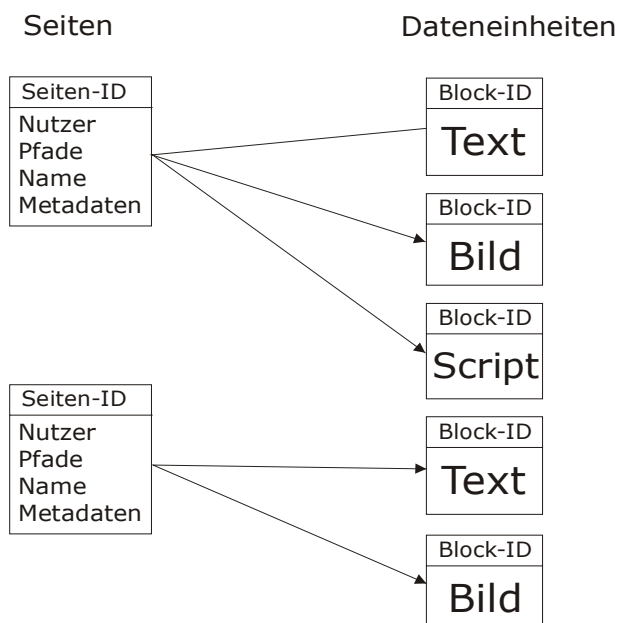
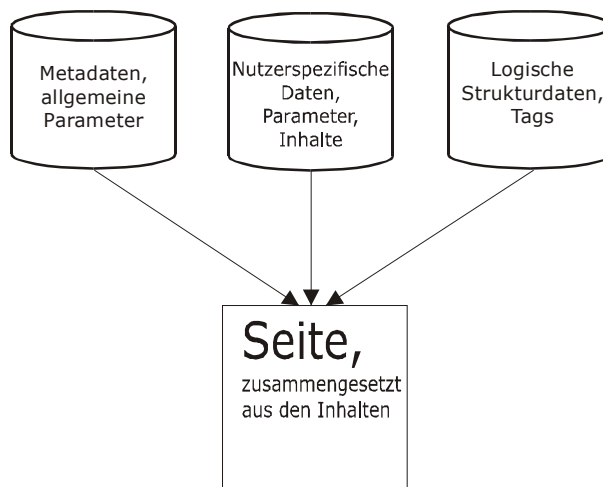


Abbildung 9 - Seitenbasierte Datenhierarchie

In diesem Modell kann auf das Erzeugen zwischengespeicherter Blockelemente verzichtet werden. Nur deren logische Struktur wird aus der Datenbank geladen und direkt in die Seite hinein generiert. Durch diese direktere Vorgehensweise kann ein Geschwindigkeitsvorteil erwartet werden. Darüber hinaus wird die Datenbank durch weniger Primärschlüssel (und höhere Redundanz) eine höhere Übersichtlichkeit und Ausfallsicherheit haben. Wenn zum Beispiel durch einen Fehler in der Ausführung der Programme ein Blockelement nicht korrekt gespeichert wird, geht es nur in der betreffenden Seite verloren und nicht in allen, in denen es Verwendung findet.



**Abbildung 10 - Seitenbasierte Seitenkonstruktion**

## 4.4. Die Tabellenstruktur der Konstruktionsdaten

Die Realisierung der rekursiven Verschachtelung der Tags ist in diesem, seitenorientierten, Modell mit weniger Aufwand realisierbar als in dem blockbasierten. Es müssen lediglich die logischen Konstruktionsanweisungen gelesen werden und nicht der komplette Block, was einen einfachen Verweis aus einen übergeordneten („Parent“) Block ermöglicht.

Eine Anpassung der Blockparameter an die einzelnen Nutzer ist nicht notwendig, da diese Logik wie in Kapitel 2.4. „Entwurf“ dargestellt, in der Anwendungsschicht enthalten ist. Gemäß dieses Konzeptes kann die Anwendungsschicht über die Speicherungs- und Generierungsfunktionalität der Hauptklasse (`Seite.tcf`) frei verfügen.

Das heißt, daß die Klasse der Applikationsschicht Methoden zur Verfügung stellen muß, die eine weitgehend freie Gestaltung der Tags und deren Parameter erlaubt.

Um die Schachtelung zu ermöglichen, muß es ebenfalls Methoden geben, um neue Tags in bereits bestehende Blöcke einzufügen oder sie daraus zu entfernen.

Im vorliegenden System soll für jede Seite eine Tabelle in der Datenbank angelegt werden, die nach einem vordefinierten Schema entworfen wurde welches für alle Seiten gleich ist. Diese Tabelle wird als `ConstructTable` bezeichnet und mit einem Namen versehen, der den Primären Seitenidentifikationsschlüssel enthält. Er lautet in diesem Beispiel wie folgt:

`seite_#_construct` <sup>23</sup>

Dieser Konstruktionsplan hat den folgenden Aufbau:

<b>Datentyp</b>	<b>Bezeichnung</b>
<code>identity</code>	<code>Komp_ID</code>
<code>char(15)</code>	<code>Typ</code>
<code>char(15)</code>	<code>EndTag</code>
<code>integer</code>	<code>Place</code>
<code>integer</code>	<code>Parent</code>
<code>bit</code>	<code>hasChilds</code>
<code>longvarchar</code>	<code>Params</code>

---

<sup>23</sup> „#“ steht für die Identifikationsnummer, also z.B. `seite_42_construct`

Auch hier ist der primäre Identifikationsschlüssel vom Typ `identity`. Er bezeichnet eine sogenannte Komponente, das heißt ein Blockelement. Alle Tags und Content-Elemente sind in dieser Tabelle enthalten.

Diese Gleichstellung zwischen diesen beiden Datenarten ist ein direktes Resultat aus der bereits in Kapitel 2.1.2. „Strukturdaten“ gemachten Feststellung, daß sich die eigentlichen Inhalte (Text) des Nutzers vom System wie Tags ohne die Möglichkeit der weiterführenden Schachtelung behandeln lassen, und damit alle relevanten Daten in einer homogenen Struktur und sogar in einer Tabelle untergebracht werden können.

Wie wird das konkret realisiert ? Schließlich handelt es sich am Ende ja doch um zwei verschiedene Dinge.

Daher hier nun die Aufstellung, wie die Spalten der `ConstructTable` in beiden Fällen genutzt werden.

#### **4.4.1. Tags als Tabellenelemente**

Ist das darzustellende Element ein Tag, dann ist die Einordnung der einzelnen Datenarten verhältnismäßig simpel.

Typ enthält den eigentlichen Bezeichner des Tags, also zum Beispiel das `H1` für eine Überschrift oder auch `IMAGE` für ein Bild. Das Feld ist für maximal 15 Zeichen Länge ausgelegt, was aber noch eine Weile reichen dürfte. Zur Zeit gibt es kaum einen Tag mit mehr als 6 Zeichen und auch die Weiterentwicklung dieser Sprache wird Namen mit 15 Buchstaben wohl nicht so bald erreichen.

Unnötige Redundanz kann an dieser Stelle vermieden werden, indem die ein- und ausleitenden Zeichen `<` und `>` weggelassen werden. Sie sind allen symbolischen Tags gemeinsam und können unkompliziert hinzugefügt werden, wenn die Seite erzeugt wird.

Wenn es sich um einen abzuschließenden Tag handelt, muß dieser im Feld `EndTag` vermerkt sein. Denn die für die Datenarbeit zuständige Klasse `Seite.tcf` kann (bei der Generierung der Seite) keine Unterscheidungen und Einordnungen dieser Angaben vornehmen. Es würde die Anwendungslogik zu sehr in die Klasse integrieren und deren Interface unnötig aufblähen. In dieses Feld ist demnach der gleiche Tag wie in das vorhergehende, nur mit einem anführenden Slash (`/`) zu verbringen. Bedauerlicherweise ist diese Redundanz erforderlich, um den Rechenaufwand bei der Zusammenstellung der Seite zu minimieren. Die weniger zeitkritische Anwendungsschicht wird diese Aufgabe übernehmen müssen. Ungeachtet dessen scheinen diese wenigen Zeichen ein recht geringer Preis für Performancegewinn bei einer kritischen Operation zu sein.

Ein weiteres sehr wichtiges Feld ist in diesem Zusammenhang `Params`. Hier werden alle Parameter des betreffenden Tags gespeichert, die nicht vorgegeben sondern nutzerbestimmbar sind. Hier handelt es sich zum Beispiel um eine ganze Reihe von sogenannten Universalattributen<sup>24</sup> wie zum Beispiel `id`, `align` oder auch `class`, die beide in sehr vielen Tags vorkommen können. Durch die Lagerung der Programmlogik in die Anwendungsschicht hinein, ist es hier allerdings der Fall, daß diese Parameter wie Rohdaten behandelt werden. Die Klasse `Seite.tcf` ist nicht in der Lage, diese Daten sinnvoll zu behandeln. Vielmehr wird sie diese als Rohdaten behandeln und ohne Prüfung in die HTML-Struktur einbauen müssen, da es andernfalls erforderlich wäre, daß sie in der Lage ist, die verschiedenen auftretenden Tagtypen zu unterscheiden. Das wurde, um diesen Umstand noch einmal kurz zu rekapitulieren, abgelehnt, weil dann beim Hinzufügen neuer Funktionalitäten in das System die Klasse und deren Interface angepaßt werden müßte. Alle Unterscheidungen und Sonderbehandlungen der Elementtypen sollen wenn irgend möglich in der Anwendungsschicht erfolgen.

Ihr wird demnach die Aufgabe zukommen, die Parameter der Blöcke zu erzeugen und der Seitenrepräsentationsklasse zu übergeben, auf daß sie in die Datenstruktur integriert werde. Natürlich muß auch das Auslesen und Aufbereiten der Daten in dieser Schicht erfolgen, damit die Werte dem Anwender zum Editieren dargeboten werden können.

So muß zum Beispiel der Terminus „`align=CENTER`“ von der Anwendung in seine Bestandteile zerlegt werden, die da der Name des Attributs (`align`) und dessen Wert (`center`) wären. Anschließend kann dem Nutzer das Ändern dieses Wertes ermöglicht werden.

Die drei verbleibenden Spalten `place`, `parent` und `hasChilds` sind für die logische Aufteilung bestimmend. Nach ihren Werten richtet sich die Seitenkonstruktionsklasse bei der Verfolgung der rekursiven Struktur der Seite.

---

<sup>24</sup> Parameter, die in den meisten Tags erlaubt sind. Siehe auch die „Selfhtml“-Dokumentation unter <http://www.netzwelt.com/selfhtml/tcma.html>

<code>place</code>	bestimmt die Position des betreffenden Elementes im Inneren des übergeordneten Tags. Es handelt sich um eine fortlaufende Nummer.
<code>parent</code>	die Identifikationsnummer ( <code>Komp_ID</code> ) des übergeordneten Tags. Da jedes Element der Seite über einen solchen elterlichen Block verfügt, kann anhand dieser und der vorhergehenden Information die genaue Position innerhalb der rekursiven Struktur bestimmt werden.
<code>hasChilds</code>	dieses Attribut ist ein logischer Wert, welcher angibt, ob das betreffende Tag seinerseits untergeordnete (Child-) Tags hat. Diese Information dient zum einen der Zeitersparnis, indem die Konstruktionsroutinen der <code>Seite.tcf</code> nicht in Blöcke verzweigen, die gar keine Childs haben und wertvolle Zeit mit sinnlosen Datenbankabfragen vergeudet wird. Zum anderen hat sie die Funktion, sicherzustellen, daß keine Tags in solchen enthalten sind, die gar keine haben können, wie zum Beispiel solche ohne abschließendes Element (z.B. <code>&lt;IMAGE&gt;</code> ).

Im Abschnitt 4.6. dieses Kapitels wird ausführlich erläutert, wie dieser Prozeß genau erfolgt. Das System erfordert es, daß jedem Tag eines übergeordnet ist. Daher wird jede `ConstructTable` als vordefinierten ersten Block mit der Nummer 1 das `<BODY>`-Element enthalten, von dem aus dann verzweigt werden kann.

#### 4.4.2. Inhalte als Tabellenelemente

Handelt es sich um ein Textelement, das in die Datenbank gespeichert wird, ist der Fall etwas anders als bei einem Tag. Hier müssen Unterscheidungen vorgenommen werden, die dennoch die homogene Struktur nicht beeinträchtigen. So muß zum Beispiel, um gutes HTML zu erzeugen und eine saubere Schachtelung der Daten zu erreichen, jeder Text innerhalb eines Tags<sup>25</sup> erscheinen. Inhalte können niemals einfach so mitten im Dokument auftauchen<sup>26</sup>, was zur Folge hat, daß sie ein übergeordnetes Element benötigen, aber ihrerseits keine Childs haben können. Sie stehen am Rande der geschachtelten Struktur.

`Komp_ID` ist demnach der gleiche Identifikator in der gleichen Tabelle. Das Feld `Typ` hingegen kann genutzt werden, um die Unterscheidung zu ermöglichen. Zu diesem Zweck

---

<sup>25</sup> z.B. `<P>`, `<H2>` oder auch `<TD>`, nicht jedoch `<IMAGE>`.

<sup>26</sup> ...obgleich zahlreiche Browser derartiges nicht bemängeln und den Text im Standardformat darstellen. Es wird aber als stilistisch gutes HTML angesehen, Text nur in Tags eingeschlossen vorkommen zu lassen.

wird in diesem Feld, wann immer es sich um Text handelt, der Wert `__text__` eingetragen. Diese Auswahl wurde willkürlich getroffen. Auch eine andere Bezeichnung wäre denkbar, solange sie sich deutlich von jedem derzeit HTML-spezifizierten Tag unterscheidet. Wann immer die Konstruktionsmaschine also auf diesen Wert (`__text__`) im Feld `Typ` trifft, ist ihr damit klar, daß es sich um Inhalte handelt, von denen aus nicht weiter verzweigt werden soll.

Doch ist damit erst klar, „daß es Text ist“. Wie selbiger nun am Ende jedoch lautet, muß aber auch bekannt sein. Zu diesem Zweck wird die in diesem Falle ohnehin nicht anders benötigte Spalte `Params` genutzt. In ihr kann der betreffende Text abgespeichert werden und ist somit schnell und ohne Zugriffe auf andere Tabellen abrufbar.

Hier nun setzt eine konstruktionsbedingte Schranke der Flexibilität enge Grenzen. Laut Pervasive.SQL-Spezifikation sollten Felder vom Typ `longvarchar` bis zu ca. 2147483648 Zeichen aufnehmen können<sup>27</sup>. Leider war in den Tests die aktuelle Version dieses Servers nicht annähernd so kooperativ, so daß sie bei Werten oberhalb von 255 Zeichen (der Maximalgrenze von Standard SQL `char`-Feldern) eine Fehlermeldung eher ablehnenden Inhalts präsentierte. Das hat zur Folge, daß ein solcher Block reinen Texts maximal 255 Zeichen umfassen kann. In der Praxis wird ein einziger Abschnitt dieser Größe hingegen selten auftreten, da meist Hyperlinks, Maillinks oder Absatzumbrüche ein neues Tag und damit auch einen neuen Block beginnen. Möglicherweise wird Pervasive Software in künftigen Versionen seiner SQL-Software den eigenen Spezifikationen besser gerecht werden.

Leider war dies nicht das einzige Problem bei der Umsetzung. Im Kapitel 4.6.5. „Behandlung von Umlauten“ werde ich näher auf die kaum mögliche Behandlung von Umlauten und anderen nicht-ASCII-Sonderzeichen eingehen.

### **4.4.3. Demonstration**

Nach all der Theorie wird nunmehr ein Beispiel der Anschaulichkeit sicher dienlich sein. Im folgenden ist ein Ausschnitt aus einer solchen Konstruktionstabelle dargestellt.

---

<sup>27</sup> Pervasive.SQL Engine Reference, Seite 207

Komp_ID	Typ	EndTag	place	parent	hasChilds	Params
42	P	/P	3	1	true	align="right"
43	__text__		1	42	false	Zur Homepage der
44	A	/A	2	42	true	href="www.heise.de/ct"
45	__text__		3	42	false	Geht's hier entlang
46	__text__		1	44	False	Zeitschrift c't

Dieses Beispiel stellt den folgenden Absatz dar:

Zur Homepage der [Zeitschrift c't](http://www.heise.de/ct) geht's hier entlang.

Wobei, falls sie dieses Schriftstück in der gedruckten Fassung vorliegen haben, der unterstrichene Teil einen Hyperlink (Verweis) auf eine andere Webseite darstellt.

Basiselement dieses Beispiels ist der erste Block mit der ID 42. Er hat als übergeordneten Parent-Block lediglich denjenigen mit der Nummer 1, also den <BODY> des Dokumentes. In diesem steht er an 3. Stelle, also recht weit vorn, möglicherweise direkt nach einer Überschrift und einem Bild. Einziger Parameter ist die Anordnung am rechten Rand (align="right"). Alle weiteren Elemente dieses Beispiel sind im Inneren dieses Blockes. Als erstes ist da ein Abschnitt reinen Textes (Nr. 43). Danach kommt ein weiterer Tag, ein Link auf die Webadresse [www.heise.de/ct](http://www.heise.de/ct). Damit dieser Link auch sichtbar ist, muß er mit irgend etwas hinterlegt werden, in diesem Fall mit einem kurzen Text, dem Block mit der Komp\_ID 46, der wiederum ein Child von Element 44 (dem Link) ist. Anschließend folgt noch ein weiterer Textabschnitt um den Satz zu vervollständigen und damit ist der Tag geschlossen.

Natürlich ist es mit diesem Konzept ohne weiters möglich, auch andere Formen eines Links darzustellen, zum Beispiel in Gestalt eines Bildes. In den Tag mit der Signatur A lassen sich letztlich komplette Unterstrukturen integrieren, die dann alle zusammen als Link auf die gleiche Adresse fungieren. Durch die Struktur der Datenbank wird stets dafür gesorgt sein, daß alle Tags ordnungsgemäß abgeschlossen und gültig sind.

## 4.5. Die Stildefinitionen

Doch wo bleibt bei all dieser Konstruktionsarbeit die in der Aufgabenstellung geforderte Auswahlmöglichkeit für den Nutzer zwischen verschiedenen Stilen für das Aussehen der resultierenden Seite ? Ist es nicht notwendig, auch diese Informationen zu speichern und in die generierte Seite zu integrieren ?

Diese berechtigten Fragen beantwortet im vorliegenden Modell CSS<sup>28</sup>. Die schon im Kapitel 2.1.2. „Strukturdaten“ erwähnte Style-Sheet-Definitionssprache ermöglicht eine tiefgreifende Änderung im optischen Erscheinungsbild durch das simple Einbinden einer anderen, externen Stylesheetdatei. Welche Datei der jeweilige Nutzer für seine Seite gewählt hat, wird in den seitenspezifischen Benutzerdaten in der Tabelle „Seiten“ abgespeichert, ebenso wie der Name der Seite oder deren Dateipfad. Die URL der CSS-Informationen wird anschließend beim Generieren der Seite eingefügt und danach vom Browser ausgelesen. Der Nutzer erhält also nahezu die gleiche Seite mit identischen Tags und Nutzerdaten übermittelt, und dennoch kann sie unter Umständen völlig anders aussehen, je nachdem, welche Style-Sheets er gewählt hat.

Ob und in wie weit das geschieht, hängt voll und ganz vom verwendeten Browser des Clients ab.

Natürlich können CSS-Anweisungen auch direkt im Text des Dokumentes erscheinen. In der Tat ist das die ursprüngliche Verwendung dieser Sprache. Für den genannten Anwendungszweck ist das jedoch wenig sinnvoll, da es ja gerade der gewünschte Vorteil ist, beim Zusammensetzen der Seite möglichst wenige zusätzliche Informationen aufnehmen zu müssen. Daher ist es unumgänglich, diese Daten in irgendeiner Weise extern abzulegen.

Zum Integrieren der externen CSS-Seite in das HTML-Dokument gibt es generell zwei Möglichkeiten:

### Integration innerhalb der Style-Sheet-Blocks

Bsp:

```
<style type="text/css">
    <!-- @import url(formate.css); //-->
</style>
```

---

<sup>28</sup> Cascading Style Sheets

Mittels dieser Methode lassen sich die Daten aus dem Inneren eines Style-Sheet-Anweisungsblocks heraus requirieren. Natürlich muß zu diesem Zweck ein solcher Block erst einmal begonnen werden. Das geschieht mittels des Tags `<STYLE>`. Das ist der spezifizierte Tag für diese Art von Style Sheets. `@import` ist demnach eine Style-Sheet-Anweisung.

### **Das Einbinden über den HTML-Tag `<link>`.**

Bsp:

```
<link rel=stylesheet type="text/css" href="formate.css">
```

Hier erfolgt die Einbindung direkt in den HTML-Tags. Es ist die simpelste Möglichkeit, da diese Art der Integration kaum Freiräume für Fehler läßt. Die Seiten sollten nach Möglichkeit von vielen verschiedenen Browsern gelesen werden können. Nicht CSS-fähige Systeme werden hier lediglich einen unbekanntenen Tag vorfinden, den sie spezifikationsgemäß ignorieren werden.

Aufgrund dieser Umstände, ebenso wie der Tatsache, daß die vorangegangene Möglichkeit auf einigen der getesteten Browser nicht so recht funktionierte, wurde zugunsten dieser Methode entschieden.

## **4.6. Die Seitenkonstruktionsklasse**

In diesem Abschnitt soll auf die Struktur und Funktionsweise der Seitenrepräsentationsklasse `Seite.tcf` eingegangen werden. Oftmals schon fand sie in den bisherigen Ausführungen Erwähnung und es wurde deutlich gemacht, welche zentrale Bedeutung ihr in dem System „Athen“ zukommt.

Noch einmal ganz kurz zur Erinnerung:

- Eine Instanz der Klasse `Seite.tcf` repräsentiert eine von einem bestimmten Nutzer zu bearbeitende Seite. Der Begriff Instanz ist an dieser Stelle nicht im eng gefaßten objektorientierten Sinne aufzufassen. Vielmehr bezeichnet er eine im Server ablaufende Spezifizierung eines auszuführenden Programmteiles auf einen bestimmten Client. Die Instanz ist demnach vielmehr virtueller als physischer Natur.
- Identifiziert wird sie mittels des Seitenidentifikators `SeitenID` aus der Tabelle `Seiten`.

- Sie übernimmt die gesamte Datenbankarbeit, die in Verbindung mit den Konstruktionstabellen steht.
- Dabei ist die Logik auf das Einfügen und Verwalten von Tags und Inhalten im Allgemeinen beschränkt. Die Anwendungslogik wird an anderer Stelle behandelt.

Diese Eigenschaften sind bestimmend für das Interface der Klasse. Darunter versteht man die Gesamtheit der Funktionen (Methoden) über welche die Klasse mit der Außenwelt, in diesem Falle der Applikationsschicht, kommuniziert. Öffentliche, von außen lesbare Membervariablen gibt es bei Tangoklassen nicht. Interne, gekapselte „Instance Variables“ hingegen schon. Daher will ich zunächst aus diese eingehen, auf das deutlich werde, über welche Daten die Klasse während ihres Lebenszyklus verfügt.

#### 4.6.1. Die Membervariablen

Die folgenden Variablen sind demnach allen Instanzen der Klasse eigen und werden von dieser gekapselt.

PageID	Der Identifikator der Seite.
ConstructTable	Der Name der SQL-Tabelle, welche die Konstruktionsdaten für diese Seite enthält.
Path	Der Pfad, unter dem die Dateien abgelegt werden sollen. Dieser sollte sich im Zugriffsbereich eines Webservers befinden, da sonst die fertige Seite nicht zugreifbar sein wird.
Stylesheet	Der Name der einzubindenden Stylesheetdatei
Backpic	Das einzubindende Hintergrundbild
Titel	Der Titel der Seite, wie er in der Kopfzeile des Browsers erscheint.
URLPrefix	Der den Dateinamen voranzustellende Präfix, aus dem sich dann letztlich die vollständige URLS zur fertigen Seite zusammensetzt.
Metas	Ein Array, welches alle Metaangaben, unterteilt in den Namen und den Wert des Metas. Wie schon oben erwähnt, sind zur Fertigstellung dieses Dokumentes 4 Metaangaben implementiert, was demnach an dieser Stelle ein 4x2-String-Array ergibt.
Filename	Der Dateiname der fertigen Datei als vollständige Pfadangabe.
shortFilename	Wie oben, jedoch nicht vollständig referenziert (ohne Pfad)

`editFilename` Der Dateiname der im Editieren befindlichen Datei. Das Dokument muß auch während der Phase, in welcher der Nutzer es verändern kann, abgespeichert werden. Für den Nutzer selbst ist das wenig relevant, da er keinen Zugang zu diesen temporären Dateien haben wird. Die Klasse selbst hingegen, wird meist mit diesem Namen arbeiten. Er wird aus der ID der Seite und dem Präfix „e\_“ generiert. Eine Seite mit der ID 42 hätte demnach eine temporäre `edit`-Datei mit dem Namen `e_42.html`.

`shortEditFilename` Wie oben, jedoch ohne Pfad

#### 4.6.2. Das Interface der Klasse `Seite.tcf`

Diese Zusammenstellung von Methoden sollen nun vorgestellt werden. Zur Darstellung der Parameterkonfiguration habe ich die in Tango übliche Notation gewählt. Selbige ist recht simpel. Auf den Namen der Funktion folgt die Aufstellung der Parameter getrennt nach Eingangs- und Ausgangsparametern. Diese werden mit  $\rightarrow$  bzw.  $\leftarrow$  gekennzeichnet.

Auf die Bezeichnung des Parameters folgt dessen Typ. Datentypen werden in Tango sehr flexibel gehandhabt, was in den meisten Fällen dazu führt, daß an dieser Stelle `Any` angegeben wird. In einem als `Any` definierten Parameter oder einer Variablen kann so gut wie alles gespeichert werden. Letztlich behandelt Tango ohnehin die meisten Daten als reinen Text, so nicht nur Strings sondern auch Zahlen.

Zusätzlich zu Ausgabeparametern hat jede Methode in TCFs einen Rückgabewert, den ich in der Notation nur erwähne, wenn er für die zu behandelnde Sache von Bedeutung ist.

Anschließend folgt noch eine kurze Beschreibung der Methode und ihrer Eigenheiten und Tätigkeiten. Auf einige der Methoden werde ich dabei nicht näher eingehen, da sich deren Bedeutung aus dem Namen erschließt.

`initialize`

$\rightarrow$  `SeitenID` [`Any`]

$\rightarrow$  `Pfad` [`Any`]

$\rightarrow$  `URLPrefix` [`Any`]

Diese Methode initialisiert eine neu angelegte Instanz der Seitenrepräsentationsklasse. Sie muß von der Applikationsschicht mit der ID

der Seite, ihrem Dateipfad auf dem Server und dem URL-Prefix<sup>29</sup> versorgt werden, welcher oft dem Domainnamen des Nutzers entspricht.

Anschließend führt die Klasse einige Datenbankabfragen durch und belegt all ihre Variablen mit den betreffenden Werten. Zum Beispiel wird an dieser Stelle der Name der Konstruktionstabelle generiert.

Es existieren zwar auch implizite Konstruktoren und Destruktoren namens `On_Create` bzw. `On_Destroy`, aber diese können nicht direkt aufgerufen und mit Parametern versehen werden, was ihre Verwendbarkeit für diesen Zweck ausschließt. Generell ist Tango2000 nur mit rudimentären Ansätzen der objektorientierten Entwicklung ausgestattet. Vererbung, Funktionsüberladung oder gar Polymorphie existieren nicht, wären allerdings nach der Meinung des Verfassers dieser Zeilen in einem solchen System fehl am Platze. Das gesamte Klassenkonzept dient vielmehr in erster Linie der Wiederverwendbarkeit der Komponenten.

`setStyleSheet`

→ `newstyle [Any]`

Diese Funktion weist die Klasse an, eine neue Stylesheetdatei in die betreffende Seite zu integrieren. Deren Name ist als String zu übergeben.

`getStyleSheet`

Der Name der aktuellen Stylesheetdatei kann mittels dieser Methode als Rückgabewert ermittelt werden.

`setMetas`

→ `m1Name [Any]`

→ `m1Value [Any]`

...

→ `m4Name [Any]`

→ `m4Value [Any]`

---

<sup>29</sup> siehe Kapitel 2.4. „Entwurf“. Dieser Präfix wird dem Seitennamen vorangestellt um eine vollständige URL zu erhalten, auf die dann verwiesen werden kann.

Wie der Name schon recht vielsagend andeutet, wird mittels dieser Methode ein neuer Satz Metaangaben integriert. Dabei wird jeweils getrennt der Name des Metas und dessen Wert angegeben.

#### `getMetas`

Mit dieser Funktion können die gespeicherten Metaangaben einer Klasse als String-Array ausgelesen werden. (Siehe dazu auch die Aufstellung der Membervariablen im vorherigen Abschnitt.)

#### `insertTag`

- `StartTag` [Any]
- `EndTag` [Any]
- `Parameter` [Any]
- `InsertPosition` [Any]

Hier haben wir eine der wichtigsten Methoden des Interface. Mit ihr ist es möglich, einen neuen Tag einzufügen, und zwar an genau die Stelle, die als `InsertPosition` angegeben wird. Dabei handelt es sich um eine gültige Komponenten-ID. Der neue Tag wird die Position des betreffenden Blocks im gleichen Parent einnehmen und selbigen um eins nach hinten schieben. Das gleiche geschieht natürlich auch mit den anderen Tags und Textblöcken im gleichen Parent.

`StartTag` und `EndTag` sind die beginnenden und, soweit vorhanden, abschließenden Zeichenketten des Blocks, also z.B. `H1` und `/H1`

Wird als `InsertPosition` 0 oder gar nichts angegeben, dann wird der Tag an die letzte Stelle des `<Body>`-Tags eingefügt.

Rückgabewert ist die Komponentenidentifikation des neu erzeugten Blockes.

#### `insertTagInto`

- `StartTag` [Any]
- `EndTag` [Any]
- `Parameter` [Any]
- `InsertPosition` [Any]

`insertTagInto` ermöglicht das Einfügen eines Tags *in* einen bereits bestehenden. Sollten sich in dem betreffenden Element bereits welche befinden, wird das neue an die letzte Stelle angehängt.

Dabei wird natürlich vorher geprüft, ob die angegebene Position auch kein Text-Block ist, und damit keine Childs aufnehmen kann.

Zurückgegeben wird auch hier die neue Komponenten-ID.

#### `insertText`

→ Text [Any]

→ InsertPosition [Any]

Diese Methode fügt einen Block reinen Texts ein (`__text__`). Dabei muß von der Applikationsschicht sichergestellt sein, daß als Parent ein Tag vorhanden ist, der diesen auch aufnehmen kann. Die Art des Einfügens entspricht der `insertTag`-Methode.

Primärer Anwendungsbereich dieser Funktion ist das Ergänzen von Text in Abschnitten, nachdem zum Beispiel ein Link oder ein Bild den Textfluß unterbrochen haben.

#### `insertTextInto`

→ Text [Any]

→ InsertPosition [Any]

Natürlich muß es eine Möglichkeit geben, Text ins Innere eines Tags zu plazieren. Möglicherweise ist dies sogar die häufigste Anwendung.

Wie bei der `insertTagInto`-Methode wird hier beim Einfügen verfahren. Es wird geprüft, ob es sich um einen `__text__`-Block handelt, und wenn nein, dann eingefügt, gegebenenfalls hinter bereits bestehende Blöcke.

Beachtet werden muß auch hier, daß dabei keine semantische Prüfung erfolgt. Diese muß in der Applikationsschicht angesiedelt sein, oder es ist durchaus möglich, einen Text im „Inneren“ eines `<IMAGE>`-Tags unterzubringen, wo er natürlich nicht ordnungsgemäß angezeigt werden kann.

`delete`

→ `ID [Any]`

Löscht einen Tag aus der Struktur heraus. Dabei wird nicht nur der betreffende, sondern natürlich auch alle untergeordneten Tags gelöscht, die sonst die Konsistenz der Datenstruktur zusammenbrechen lassen würden. Auch die Reihenfolgeangaben der restlichen Blöcke im gleichen Parent werden entsprechend geändert. Vorher erfolgt ein Test, ob nicht eventuell das `<BODY>`-Tag mit der Nummer 1 als Parameter gewählt wurde.

`generate`

`gen_addItem`

→ `ItemID [Any]`

Diese beiden Methoden stellen die eigentliche Kernfunktionalität der Klasse zur Verfügung. Sie generieren aus der Datenstruktur heraus die resultierende HTML-Datei. Dabei ist `generate` die aufzurufende Funktion und `gen_addItem` die Rekursivfunktion. Im Abschnitt 4.6.4. dieses Kapitels werde ich auf diese Algorithmen näher eingehen.

`Generate` ist parameterlos. Die Klasse verfügt bereits nach der Initialisierung über alle nötigen Daten um zu wissen, wohin und mit welchen Besonderheiten die Seite zu generieren ist.

`edit`

`edit_addItem`

→ `ItemID [Any]`

Natürlich muß sich die von dem Anwender veränderbare Seite vom fertigen Produkt in einigen wichtigen Punkten unterscheiden. Letztlich sind alle wesentlichen Funktionalitäten, die eben diese Bearbeitung erfordern im HTML einzubetten. Das erfordert eine andere aufzurufende Methode, die mit `edit` bezeichnet ist. Natürlich sind dabei umfangreiche Parallelen zu `generate` vorhanden, was auch hier eine Rekursivfunktion (`edit_addItem`) erforderlich macht.

setItemData

- Item [Any]
- StartTag [Any]
- EndTag [Any]
- Parameter [Any]

Um ein Element der Datenstruktur ändern zu können, ist es selbstverständlich notwendig, der Applikationsschicht eine Möglichkeit zu bieten, gezielt auf alle Werte eines Elementes zuzugreifen. Das geschieht mittels der Methode `setItemData`. Auch hier wird aus Leistungs- und Geschwindigkeitsgründen keine Plausibilitätsprüfung im Inneren der Klasse durchgeführt. Schließlich ist es nicht zu erwarten, daß andere als die Routinen der zugehörigen Applikation auf die Klasseninstanz zugreifen, deren Methoden aufrufen und damit den Datenbestand ändern. Dennoch ist es mittels dieser Funktion theoretisch möglich, zum Beispiel aus einem Element vom Typ `H2` einen Textblock zu machen, was die Integrität der Daten natürlich stark beeinträchtigen würde. Die Anwendungslogik muß dafür Sorge tragen, daß derartiges niemals passieren kann. Die Parameter sind weitestgehend identisch mit den Funktionen zum Einfügen der Blöcke. `Item` bezeichnet den Identifikator des Elementes (`komp_ID`).

Beachten Sie bitte an dieser Stelle, daß es für die Applikationsschicht nicht möglich ist, auf die strukturellen Informationen wie `place`, `parent` oder `hasChilds` zuzugreifen. Einzig die Seitenrepräsentationsklasse hat hierzu die Möglichkeit. Dieser Umstand ist eine Folge der getrennten Schichten. Die Eigenarten der Blöcke sind durchaus Sache der Anwendungslogik, wohingegen deren datenmäßige Konsistenz der Serverlogik zur Aufgabe gereichen.

getItemData

- Item [Any]
- ← StartTag [Any]
- ← EndTag [Any]
- ← Parameter [Any]

Die umgekehrte Funktionalität bietet diese Methode. Mit ihr können alle Daten eines betreffenden Elements aus der Datenbank herausgelesen werden. Einziger Eingangsparameter ist die ID des Blockes. Alle anderen Parameter geben die gewünschten Daten heraus.

`getFileName`

`setFileName`

`getEditFilename`

Diese Methoden geben den Namen der fertigen bzw. der vom Nutzer zu verändernden Datei aus. Auch das Neusetzen des Namens ist möglich. Da der Name der bearbeiteten Datei aus der ID der Seite heraus generiert wird um Doppelungen zu vermeiden, kann er nicht direkt gesetzt werden.

`setBackpic`

`getBackpic`

Jenes Funktionspaar ermöglicht das Setzen und das Auslesen des Hintergrundbildes für die Seite<sup>30</sup>.

`getPath`

`getURL`

`getURLPrefix`

Die übrigen Methoden erlauben den Zugriff auf verbleibende Daten, die für die Applikationsschicht zum ermitteln von Linkadressen und ähnlichem wichtig sind. Es handelt sich um simple Rückgabefunktionen.

Aus dieser Aufstellung ist leicht ersichtlich, wie umfangreich das Interface trotz der leicht eingeschränkten, von der Anwendungslogik getrennten Funktionalität ist (ich verweise dazu auch auf Kapitel 2.4 „Entwurf“). Im Falle einer weniger allgemeinen und dafür funktionelleren Gestaltung würde die Vielzahl der Methoden exponentiell zunehmen. Für jeden einzelnen Tagtyp wären Funktionen für `insert` und `insertInto` erforderlich. Bei jeder Änderung der Klasse müßte die Anwendungsschicht angepaßt werden und umgekehrt.

---

<sup>30</sup> Dabei handelt es sich um eine Bilddatei, die vom Browser in den Hintergrund der Seite gekachelt wird. Zumeist ein einfaches Muster oder Symbolik.

### 4.6.3. Die temporäre Bearbeitungsseite als Resultat der edit-Methode

Wie im letzten Abschnitt ersichtlich wurde, stellt `Seite.tcf` ihre Hauptfunktionalität mittels der Methoden `generate` und `edit` zur Verfügung. Im nächsten Abschnitt sollen die dahinter stehenden Funktionsweisen erläutert werden.

Zunächst aber sollte ich darauf eingehen, worin genau sich eigentlich die Produkte der beiden Funktionen unterscheiden. Im vorangegangenen Text wurde das Resultat von `edit` als das vom Anwender zu bearbeitende Zwischenstadium charakterisiert. Es muß also Bestandteile enthalten, die es innerhalb von schlichtem HTML erlauben, mit der Anwendungsschicht auf Seiten des Servers zu kommunizieren und ihr die nötigen Daten zu übermitteln. Aber wie kann das realisiert werden ?

Die Oberfläche der Anwendung „Athen“ beruht auf einem sogenannten Frameset. Zwei Teile davon sind für die Nutzerinteraktion essentiell. In einem der Frames wird eine aus Links bestehende Menüstruktur dargestellt, die jeweils die vom Anwender gewünschten Befehle enthält.

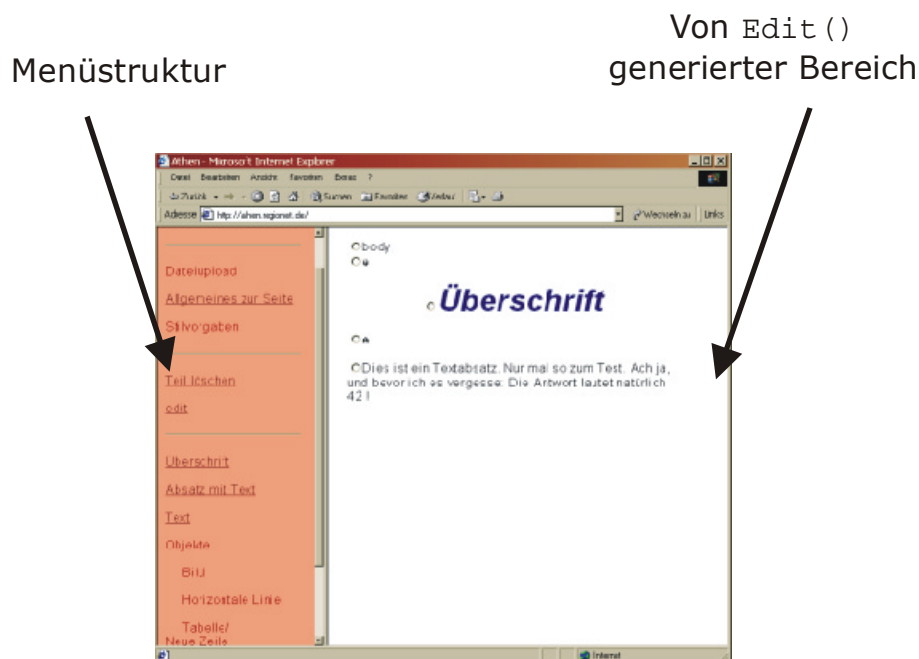


Abbildung 11 - Die Oberfläche des Systems Athen

Im Grunde ist es also lediglich erforderlich, die einzelnen Bestandteile des Dokumentes deutlich kenntlich zu machen und dem Anwender gleichzeitig eine Selektion zu ermöglichen.

Die Information, welches Element selektiert ist, muß der Applikation zugänglich sein, damit sie darauf zurückgreifen kann, wenn der Nutzer einen Menüpunkt klickt.

Die in Tango effektivste Möglichkeit, dies zu realisieren, ist ein Cookie. Es kann vom Server in TAFs und TCFs unter schlichter Verwendung des Scopes „Cookie“ wie eine gewöhnliche Variable gelesen werden und es ist auf Seiten des Edit-HTML mittels einer einfache Javascript-Funktion wie dieser hier setzbar:

```
<script Language="JavaScript">
<!--
function setzeCookie(name,wert) {
    var arg_wert = setzeCookie.arguments;
    var arg_laenge = setzeCookie.arguments.length;
    var expires = (arg_laenge > 2) ? arg_wert[2] : null;
    var path = (arg_laenge > 3) ? arg_wert[3] : null;
    var domain = (arg_laenge > 4) ? arg_wert[4] : null;
    var secure = (arg_laenge > 5) ? arg_wert[5] : null;
    document.cookie = name + "=" + escape(wert) +
        ((expires == null) ? "" : ("; expires=" +
expires.toGMTString())) +
        ((domain == null) ? "" : ("; domain=" + domain)) +
        ((secure == true) ? "; secure" : "");}
//-->
```

Aufgerufen wird die Funktion beim Klick auf einen Radiobutton.

```
<input type=radio name="Item" value="1"
onclick="setzeCookie('WorkItem','0')">
```

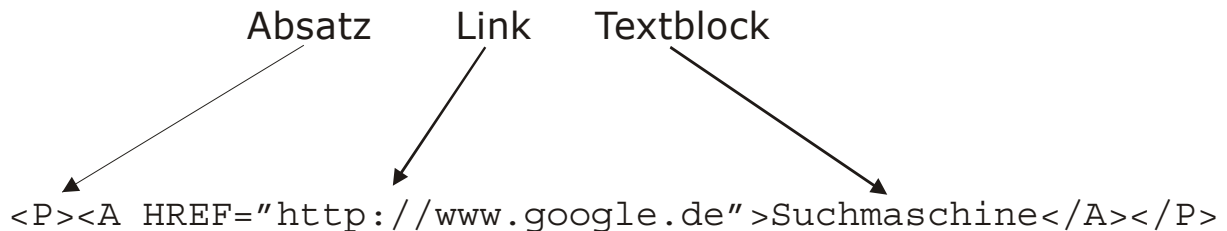
Nun muß dem Nutzer erst einmal etwas dargeboten werden, was ihm der Browser als selektierbare Komponente darstellt. Da nur jeweils ein einzelnes Element angewählt sein kann, sollten sich diese Selektionsmöglichkeiten idealerweise gegenseitig ausschließen. HTML bietet diese Funktionalität in Formularen in Gestalt von gruppierten Radiobuttons<sup>31</sup>

---

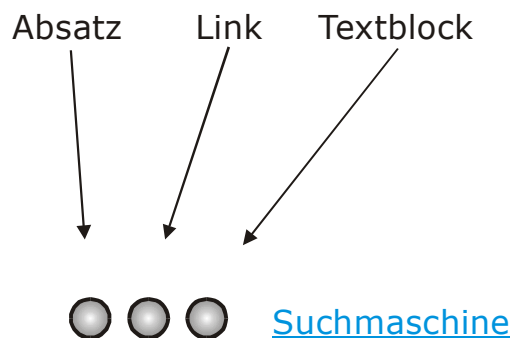
<sup>31</sup> Radiobuttons sind umgangssprachlich kleine Auswahlfelder, die meist gruppiert auftreten. Sie zeichnen sich dadurch aus, das immer nur eines gewählt sein kann. Bei der Anwahl eines anderen erlischt die letzte Selektion.

an. Diese Buttons können wie oben gezeigt mit Befehlszeilen versehen werden und treten beim Klick des Nutzers in Aktion. Natürlich ist die exklusive Selektion dabei lediglich optischer Natur. Es gibt *ein* zu setzendes Cookie und die Radiobuttons wurden nur gewählt, weil hierbei das programmtechnische Darstellen der automatischen Deselektion entfallen kann. Auch die Gruppierung muß implementiert werden, was durch die Einbettung der betreffenden Tags in ein einziges Formular erfolgt. Das Attribut name im input-Tag markiert die Radiobuttons als zusammengehörig.

Natürlich müssen dabei auch Informationen für den Nutzer vorhanden sein, die ihm deutlich machen, *was* er denn nun eigentlich anklickt. Schließlich können unter Umständen mehrere Tags geschachtelt auftreten und es kann zu einer Aneinanderreihung von Radiobuttons kommen, wenn jeder Tag über eine solche Auswahlmöglichkeit verfügt. Im ungünstigsten Fall ist der betreffende Tag dann auch noch für den Betrachter unsichtbar. Beispielhafterweise würde ein solcher Fall auftreten, wenn sich ein Link am Anfang eines Absatzes befindet. Jedes der 3 beteiligten Elemente muß wähl- und veränderbar sein. Der Absatz, der Link als solcher mit seiner Zieladresse und natürlich auch der Text im Inneren des Links, durch welchen er erst sichtbar wird.



Hier sind 3 logisch wählbare Elemente auf engem Raum vereint und würden nur mit Radiobuttons versehen im Browser des Nutzers etwa ein derartiges Bild ergeben:



Die genaue Darstellung hängt auch hier vom verwendeten Browser und Betriebssystem bzw. Windowmanager ab, aber es ist zu erkennen, daß sich vor dem einfachen kleinen Link 3 Buttons befinden.

In Athen wird dies durch kleine Piktogramme gelöst, die dem jeweiligen Element so es unsichtbar ist, vorangestellt werden.



Wählt der Anwender nun den mit A markierten Button, kann er die Eigenschaften des Absatzes ändern, während L den Link bezeichnet.

Hier steht jeweils ein kleiner Buchstabe für die Art des Tags, jedoch können die Bilder beliebig angepaßt werden, um dem unbedarften Nutzer ein wenig besser entgegenzukommen.

Wird einer diese Buttons geklickt, erfolgt ein Aufruf der Javascript-Funktion `setzeCookie()`, die ein Cookie namens „WorkItem“ auf den Wert des angewählten Items setzt. Anschließend kann ein Klick auf einen der Menüpunkte das entsprechende TAF mit genau diesem Wert als Argument starten.

Es ist zusammenfassend feststellbar, daß also die `edit`-Funktion eine Seite kreieren muß, die sich von der gewöhnlichen Seite in 3 Punkten unterscheidet:

- alle editierbaren Blöcke sind mit einen Radiobutton versehen
- diese Buttons lösen einen Aufruf der Javascript-Funktion zum Setzen des Cookies aus
- „unsichtbaren“ Tags ist ein Bild vorangestellt
- die Integration von Metainformationen ist unnötig
- angepaßtes Stylesheet

Der letzte Punkt ist dem Umstand geschuldet, daß Probleme bei der Darstellung der Seite in unterschiedlichen Browsern auftreten. Um die optische Zusammengehörigkeit von Bild, Radiobutton und betreffendem Element zu gewährleisten, ist es nötig, daß sie soweit wie möglich auf einer Zeile und günstigstenfalls eng zusammen liegend dargestellt werden. Leider ist meist nach einem derartigen Element ein Zeilenumbruch üblich. Aus diesem Grunde

wurde in der `edit`-Methode ein speziell zu diesem Zwecke modifiziertes Stylesheet verwendet. Es enthält unter anderem die folgenden Anweisungen zur `inline`-Darstellung:

```
input[type="radio"] + h1, h2, h3, h4, p, blockquote
{display:inline}
image + h1, h2, h3, h4, p, blockquote {display:inline}
image {display:inline}
image + tr, td, table {display:inline}
```

Leider hatten diese Anweisungen nicht auf jedem der getesteten Browser den gleichen gewünschten Effekt. Einige stellen den Zeilenvorschub dennoch dar, oder ignorierten das Stylesheet ganz. Diese oben schon erwähnten Cross-Browser-Probleme waren und sind stets ein großes Hindernis bei der Entwicklung von Webseiten.

#### 4.6.4. Die Funktionsweise der Seitengenerierung

Wie im letzten Abschnitt ersichtlich wurde, gibt es Unterschiede in der Art, wie die Methoden `edit` und `generate` ihre Ergebnisse erzeugen. Dennoch arbeiten sie mit der gleichen Datenbasis und haben wie in der Interfacebeschreibung deutlich wurde auch einen ähnlichen rekursiven Aufbau, auf den ich nunmehr näher eingehen möchte. Den besten Überblick bietet vermutlich ein schematischer Programmablaufplan der beiden Methoden.

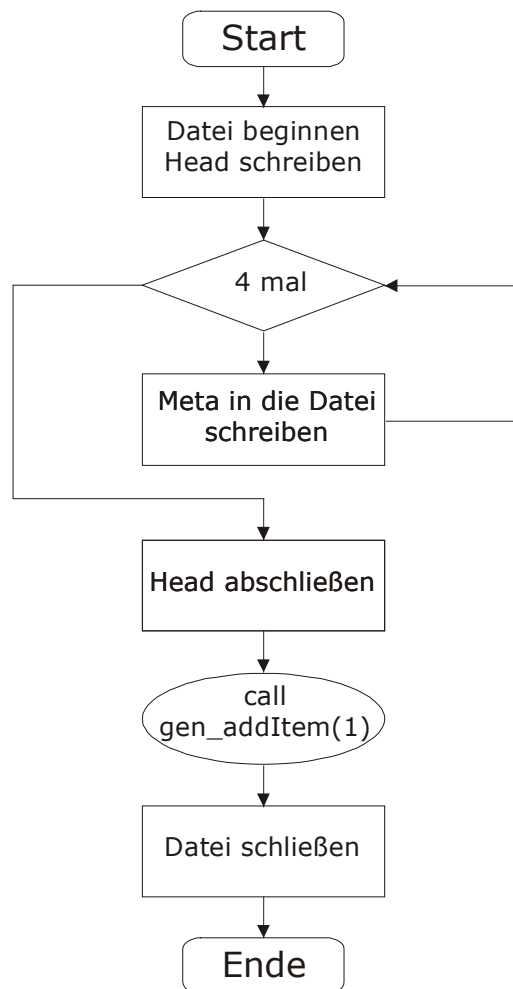
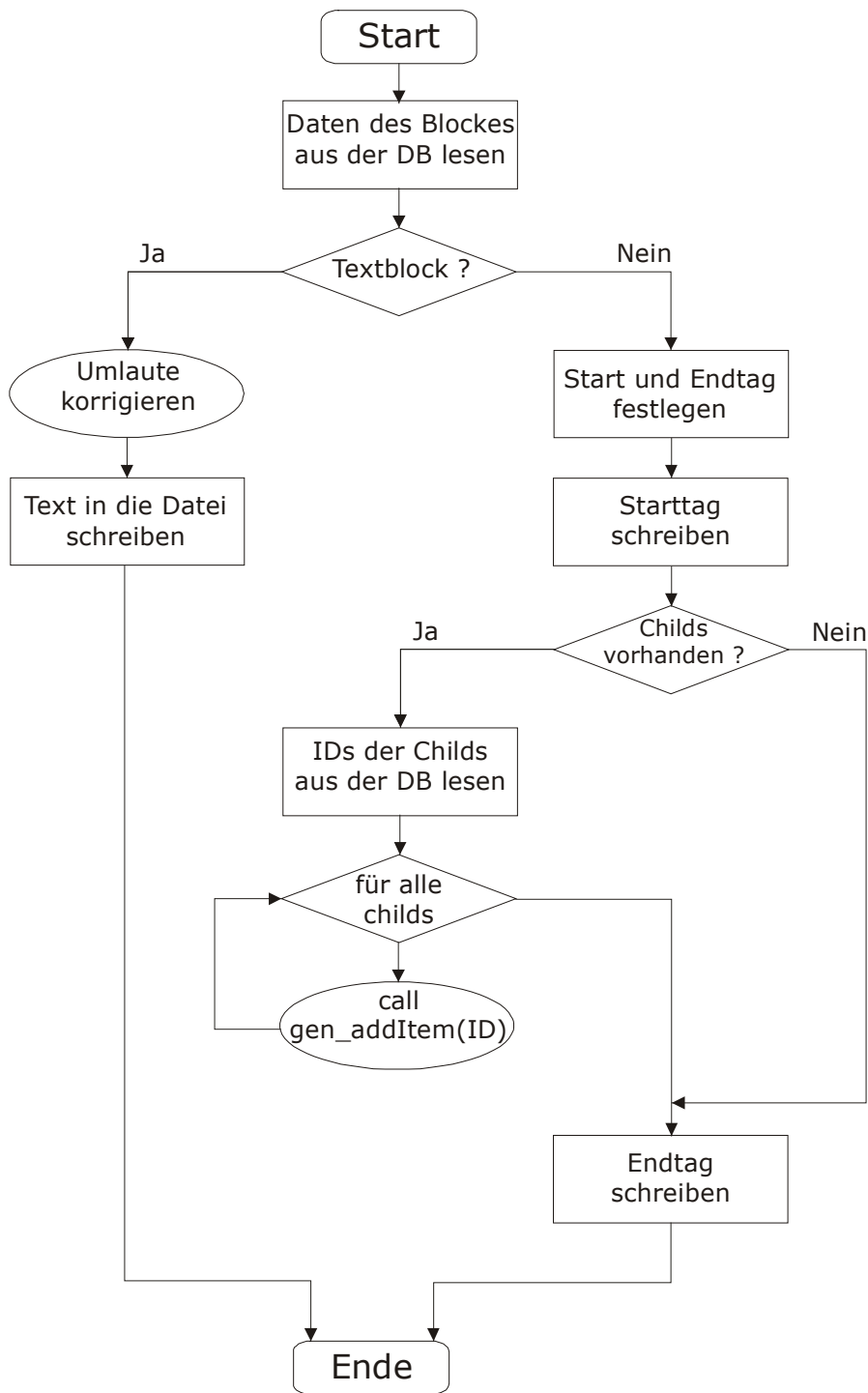


Abbildung 12 - Die Methode `generate`



**Abbildung 13 - Die Methode `gen_addItem`**

Aus diesen beiden Plänen ist die Grundlage der rekursiven Verarbeitung zu erkennen. Wenn `gen_addItem` mit der ID eines Blockes als Argument aufgerufen wird, liest die Funktion als erstes die betreffenden Daten aus der Datenbank. Diese schreibt sie nacheinander in aufbereiteter Form in die Datei. Wenn es sich nicht um einen Textblock handelt, wird ermittelt, ob der Block seinerseits weitere Blöcke enthält. Ist dies der Fall, wird die Methode `gen_addItem` für jedes dieser Elemente aufgerufen.

Der erste Aufruf, und damit der Start der Rekursion erfolgt in der `generate`-Funktion, und zwar immer durch den Aufruf `gen_addItem` mit dem Argument 1, der stets vorhandenen ID des BODY-Tags.

Die Algorithmen der Methoden `edit` und `edit_addItem` sind nahezu identisch zu den eben genannten. Die einzigen Unterschiede bestehen im Hinzufügen der im letzten Abschnitt (4.6.3) genannten Besonderheiten der Bearbeitungsseite und den Dateinamen und -pfaden.

#### 4.6.5. Behandlung von Umlauten

Der Application Server Tango2000 ist mit einer Möglichkeit ausgestattet, mit Umlauten und anderen Sonderzeichen des nichtangelsächsischen Sprachraumes umzugehen. Sie werden innerhalb von GET und POST Abfragen in einem speziellen Format transportiert und können sogar aus Datenbanken oder Parametern heraus in Seiten integriert werden. Allerdings werden sie das leider nicht in Form der Standard HTML-Formate wie `&Uuml;` für "Ü", sondern in der weitaus weniger verbreiteten Darstellung ihres Unicode (`&#220`). Diese Darstellung kann allerdings nur ordnungsgemäß umgesetzt werden, wenn die betreffende Seite auch vom Tango Server an den Browser übermittelt wird. Und zwar in Form eines TAF. Wird eine HTML-Seite vom Server erzeugt, wie es im vorliegenden System geschieht, wird diese jedoch nicht mehr länger von Tango, sondern vom Webserver angezeigt. Schließlich ruft dieser ja den Tango-Server nur zur Darstellung über das Plugin auf, wenn er auf die Endung `*.taf` in der URL trifft<sup>32</sup>. Das ist bei der fertig generierten Seite nicht der Fall. Sie wird ganz normal wie eine gewöhnliche HTML-Seite angezeigt, was sie ja auch sein soll. Das hat zur Folge, daß derartige Sonderzeichen verstümmelt oder gar nicht angezeigt werden.

Einzig praktischer Ausweg aus diesem Problem war das Ersetzen der betreffenden Zeichen im Text *bevor* diese in die Datenbasis transferiert werden. Aus „Ü“ muß `&Uuml;` werden. Diese Zeichen werden von Tango nicht als Besonderheiten betrachtet und ganz normal ins HTML integriert. Zu diesem Zwecke mußte ein serverseitiges Javascript geschrieben werden, welches eben diesen Austausch vornimmt. Es ist recht simpel und besteht aus wenigen Kommandos zum Ersetzen der Zeichen.

```
var str=server.getVariable("txt","method");
str=str.replace(/§/, "&szlig;");
```

---

<sup>32</sup> vergleiche dazu Kapitel 3.2.1. „Der Tango Server“

...weiterer Austausch...

```
server.setVariable("txt",str,"method");
```

Das Objekt `server` ist bei serverseitigen Javascripts seit der ersten LiveWire-Spezifikation vordefiniert und stellt eine ganze Reihe von Methoden zur Verfügung, die im Umgang mit serverspezifischen Algorithmen nötig sind, wie zum Beispiel das Ermitteln der IP des Clients und Portadressen und ähnlichem. Hier handelt es sich um ein Modell mit erweitertem Interface, welches es ebenfalls erlaubt, Variablen zu setzen und auszulesen.

Es ist natürlich auch denkbar, diesen Austausch erst vorzunehmen, wenn die Konstruktion der Seite erfolgt, was allerdings während dieser kritischen Phase noch mehr Rechenkapazität in Anspruch nehmen würde als ohnehin schon gebraucht wird.

Unangenehmer Seiteneffekt ist allerdings die Tatsache, daß diese umgewandelten Sonderzeichen in genau dieser Form wieder zum Anwender gelangen wenn dieser einen von ihm verfaßten Text bearbeiten möchte. Er bekommt also statt der gewohnten Umlaute HTML-Sonderzeichen zu sehen. Eventuell sollte in einer späteren Version von Athen über eine Rückkonvertierung der Zeichen vor dem Integrieren in das Editiertextfeld der Bearbeitungsseite nachgedacht werden.

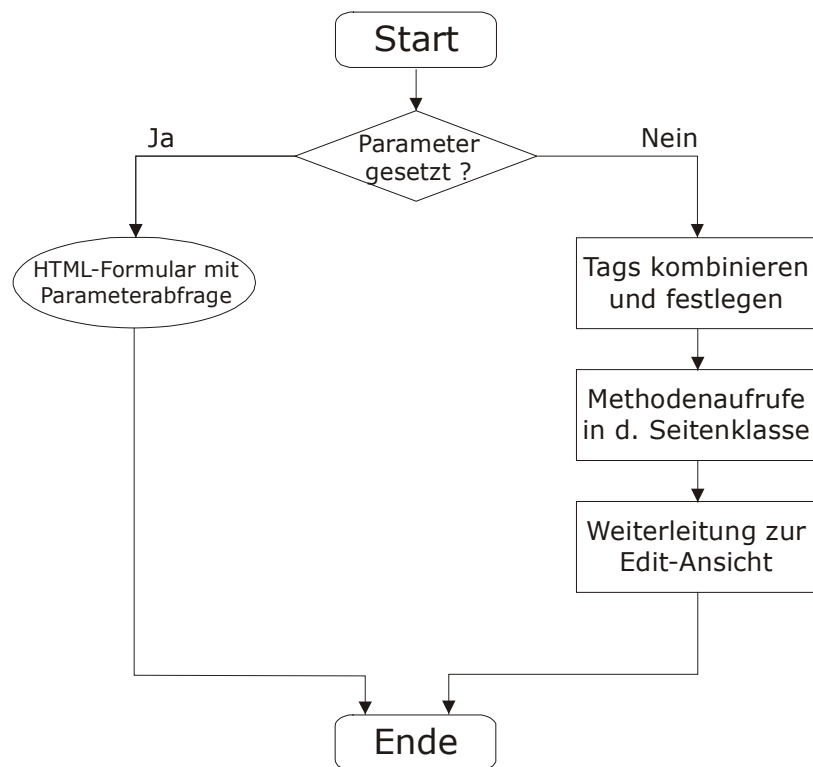
## **4.7. Die Anwendungsschicht**

Die Oberfläche des Redaktionssystems Athen hat zum Zeitpunkt der Erstellung dieser Arbeit lediglich Testcharakter. Sie entbehrt noch einer einheitlichen und einfachen Nutzerführung, sowie auch einer ansprechenden Optik. Prinzipiell kann sie auch verschiedene Gestalten annehmen, solange sie mit der zentralen Klasse `Seite.tcf` zusammenarbeitet. Dennoch möchte ich an dieser Stelle die im Rahmen des Projektes entwickelte und bewährte Struktur der verschiedenen Komponenten der Anwendung Athen erläutern.

Prinzipiell ist festzustellen, daß sich eine Aufteilung in kleine Dateien anbietet um die unterschiedlichen Elementtypen und Aktionen behandeln zu können. Auf die wichtigsten davon verschiedene werde ich nun einzeln eingehen.

### **4.7.1. Ein Element hinzufügen**

Diese Menüpunkte sind sicher diejenigen, die am häufigsten benutzt werden und lassen sich unkompliziert mittels einfacher kleiner TAFs behandeln, die alle vereinfachterweise die folgende Struktur haben:



**Abbildung 14 - Einfaches TAF zum Hinzufügen eines neuen Elements**

Beim Aufruf einer solchen Datei wird zunächst überprüft, ob Eingangsparameter vorhanden sind. Ist dies nicht der Fall, wie es bei einem neuen Aufruf üblicherweise sein wird (GET), dann liefert Tango dem Webserver<sup>33</sup> eine HTML-Seite, die ein Formular enthält, welches dem Nutzer die Möglichkeit bietet, die Parameter für das Objekt seiner Wahl festzulegen. Zum Beispiel einen Dateinamen und Ausrichtung für ein Bild oder auch die Adresse und den Text für einen Link. Anschließend bestätigt der Nutzer seine Eingaben und fordert also das gleiche Dokument mittels POST noch einmal vom Server an, diesmal jedoch mit der Übergabe der eben in das Formular eingegebenen Parameter. Dieses kann nun die Daten in HTML-Tags und Parameter umsetzen und die entsprechenden Kommandos an die Seitenrepräsentationsklasse in Form von Methodenaufrufen übergeben. Dies werden für gewöhnlich mehrere Aufrufe sein. So muß zum Beispiel für eine einfache Überschrift nicht nur das HTML-Element (H3) sondern auch der enthaltene Text (`__text__`) eingefügt werden.

Ein solcher Programm rumpf läßt sich leicht entwickeln, wiederverwerten und auch warten. Eine Änderung der unterstützten Tags oder Parameter ist problemlos möglich, indem das Formular geändert und die Methodenaufrufe angepaßt werden.

<sup>33</sup> vgl. dazu die Übersicht über die Funktionsweise des Servers in Kapitel 3.2.1 „Der Tango Server“

An welche Stelle das betreffende Element eingefügt wird, bestimmt der Nutzer durch einen Klick auf den Radiobutton eines der Elemente und das damit gesetzte Cookie. Dabei treten unter Umständen Sonderfälle auf, die vor dem Aufruf abgefangen werden sollten. Dazu gehören etwa angewählte Blöcke bei denen wohl in ein das Element hinein, als auch an dessen Stelle das neue treten kann. Auch das Anwählen eines solchen Elements ohne vorher angewähltes Item ist denkbar. In diesem Falle ist von Athen der BODY-Tag vorgegeben.

#### **4.7.2. Das Aufrufen der Bearbeitungsseite**

Wie im Abschnitt 4.6.3 beschrieben, ist die `edit`-Methode eine der wichtigsten Funktionen der Klasse `Seite.tcf`. Allerdings kann sie nicht einfach aus dem Browser heraus aufgerufen werden, noch sollte dies wahllos geschehen, da auch hier Sonderfälle auftreten können. So ist es zum Beispiel erforderlich, erst einmal zu ermitteln, ob bereits eine Instanz der Seitenrepräsentationsklasse besteht, die mittels `edit` angesprochen werden kann. Gegebenenfalls ist diese zu erzeugen und zu initialisieren, wozu natürlich Datenbankabfragen und Wertübergaben an die `initialize`-Methode gehören.

All dies übernimmt ein TAF namens `Edit_Page.taf`, welches die nötigen Dinge prüft und das Resultat der von ihm aufgerufenen Methode an den Browser schickt. Das geschieht nicht direkt, sondern indem die von `edit` erzeugte HTML-Datei über eine `Presentation`-Action aufgerufen wird.

Die meisten der TAFs, die eine Seite verändern, haben als letzte Anweisung den Aufruf dieser Datei. Damit wird gewährleistet, daß die angezeigte Seite stets aktuell ist.

#### **4.7.3. Das gemeinsame Bearbeitungs-TAF**

Wann immer der Nutzer eines der Elemente auswählt, sollte ihm die Möglichkeit geboten werden, dessen Eigenschaften zu bearbeiten. Da allerdings die einzige durch das Anwählen übermittelte Information das Cookie mit der ID ist, wird durch den entsprechenden Menüpunkt stets das gleiche TAF aufgerufen werden müssen. Ihm obliegt es, zunächst erst einmal die Daten des Elements mittels `getItemData`<sup>34</sup> zu erfragen, um diese gleich darauf dem Nutzer in aufbereiteter Form, als Standardwerte in ein Formular eingetragen, zur Veränderung darzubieten. Wird dieses Formular dann ausgefüllt bestätigt, können die

---

<sup>34</sup> siehe Abschnitt 4.6.2 „Das Interface der Klasse `Seite.tcf`“

Informationen mittels `setItemData` wieder zurückgeschrieben werden. Vorher erfolgt die Rückkonvertierung in die entsprechenden HTML-Attribute.

#### **4.7.4. Die Menüstruktur**

Die in der linken Hälfte des Framesets<sup>35</sup> befindliche Menüstruktur ist Ergebnis eines einzigen TAFs (`Navigation.taf`), welches je nach Parameter beim Aufruf unterschiedliche Kaskadierungsebenen erlaubt. Ein Klick auf einen der Menüpunkte löst entweder den Aufruf eines TAFs oder den Aufruf des Menüs mit einer anderen Kaskadierungsstufe aus, wodurch es praktisch beliebig erweiterbar ist. Leider setzen die etwas begrenzten Möglichkeiten von HTML in dieser Sache der Handlichkeit des Menüs schnell enge Grenzen. In einer möglichen späteren Version sollte es durch ein besser angepaßtes Menü ersetzt werden.

#### **4.7.5. Allgemeines und Dokumentenweite Einstellungen**

Ähnlich im Aufbau aber unterschiedlich in der Funktionsweise sind TAFs, die das Einstellen dokumentenweiter Daten wie Metas, Stylesheets oder auch den Dateinamen der Seite erlauben. Auch hier sind, wie in Abbildung 14 beschrieben, zwei Funktionszweige vorhanden, deren Auswahl auf der Frage beruht, ob der Aufruf parameterbehaftet erfolgte oder nicht.

Anders ist, daß hier keine Änderung an der Konstruktionstabelle vorgenommen wird. Die Änderungen werden zwar auch von `Seite.tcf` über diverse Methodenaufrufe durchgeführt, haben aber keine neuen Blöcke zur Folge. Das bedeutet, daß auch die Funktionen `getItemData` und `setItemData` untätig bleiben. Alle Änderungen erfolgen an Nutzer- und Verwaltungsdaten in der Datenbanktabelle `Seiten`.

Derzeit existieren zwei solcher Dateien. `Allgemeines.taf` setzt Metaangaben, Dateiname und Hintergrundbild, während `Stylez.taf` das Einstellen der jeweiligen Stilvorgabe (Stylesheet) und damit das Aussehen des Ergebnisses erlaubt<sup>36</sup>. Letzteres ist dabei mit kleinen, beispielhaften Screenshots versehen, an denen der Nutzer eine Vorstellung vom Ergebnis bekommen kann. Er kann es allerdings nicht sofort im gleichen Frameset in Form der Bearbeitungsseite bewundern, da selbige auf einem angepaßten Stylesheet (Radiobuttons) beruht. Diese Einschränkung mag sich als nützlich erweisen, wenn sich zu den vordefinierten Stilvorlagen ungewöhnliche oder schwer lesbare gesellen sollten, die zwar den modern gesinnten und multimediagestählten Betrachter erfreuen aber ein vernünftiges Bearbeiten sehr

---

<sup>35</sup> siehe Abbildung 11 – Die Oberfläche des Systems Athen

erschweren könnten. Bei der Definition der Stile ist demnach drauf zu achten, daß keine schweren Diskrepanzen im grundlegenden Layout zwischen diesen beiden Stilen auftreten.

---

<sup>36</sup> siehe Kapitel 4.5. „Stildefinitionen“

# 5. Bewertung und Analyse

## 5.1. Probleme und Mißstände

Wie es bei Entwurf und Implementierung einer Software wie dieser leider nicht anders zu erwarten war, traten zahlreiche Probleme und unerwünschte Effekte auf, die streckenweise signifikanten Einfluß auf die Struktur und die Möglichkeiten des System hatten und auf die ich nun eingehen möchte. Im folgenden dieses Kapitels werde ich mir erlauben, auch Anregungen und Konzepte zur Behebung dieser bedauerlichen Mißstände und zur Weiterentwicklung anzusprechen.

### 5.1.1. Geschwindigkeit

Die Gesamtperformance des Systems, speziell die der Seitenrepräsentationsklasse `Seite.tcf` erwies sich leider als für die Praxis absolut unbefriedigend. Dieser Umstand beruht auf verschiedenen Faktoren, die zum Zeitpunkt der Entwicklung maßgeblich waren:

- Mindere Anpassung des Applicationsservers an diese Art der Aufgabenstellung. Tango2000 ist ein für diese Gattung typisches Produkt. Es ist auf nicht allzu lange und komplizierte, mit umfangreichen SQL-Statements versehenen TAFs ausgelegt und eher weniger auf rekursive Programmstrukturen, die bekanntermaßen sehr ressourcenbedürftig sein können und es in diesem Falle zumeist auch sind. Ein Umstand, der sich in der Anzahl der zur Verfügung stehenden Programmstrukturen widerspiegelt. Dazu kommt Filelogging und die sicher auftretende parallele Aktivität der Servers. Alles in allem dauert die Generierung selbst einer einfachen Seite deutlich zu lang, was in noch größerem Maße für die `edit`-Seite zutrifft.
- Die für heutige Maßstäbe relativ langsamen Macintosh-Server erlaubten es dem recht neuen Produkt Tango2000 nicht, mit optimaler Geschwindigkeit zu agieren. Möglicherweise ist demnach auch der Hardware-Flaschenhals ein nicht zu unterschätzender Einflußfaktor.
- Der zu Test- und Entwicklungszwecken genutzte Datenbankserver Pervasive.SQL lief auf einer - subjektiv betrachtet - älteren Linuxmaschine (Pentium 133, 64 MB RAM) unter SuSE-Linux 6.4 und einem nicht optimierten Standardkernel. Diese Konfiguration mag für derartige Zwecke ausreichend erscheinen, muß es aber durchaus nicht sein. Die Software ist durch die kurz hintereinander auftretenden kleinen Datenbankabfragen auf

sehr schnelle Reaktion des Servers angewiesen, da sich selbst kleinste Verzögerungen durch die Vielzahl der Anfragen schnell zu langen Wartezeiten aufaddieren können.

### **5.1.2. Nutzerführung innerhalb des Browsers**

Die Oberfläche erwies sich innerhalb der engen, von einfachen HTML-Dokumenten gesteckten Grenzen als instabil und anfällig gegenüber Irregularitäten, die durch die Verwendung der Back- und Fwd-Buttons des Webbrowsers durch den unbedarften Nutzer (die Zielgruppe beinhaltet primär Anwender mit wenig Erfahrung auf diesem Sektor) auftreten. Diese Problematik ist symptomatisch für komplexere Webanwendungen und nicht spezifisch auf dieses Produkt beschränkt. Wann immer der Nutzer etwas außerhalb der Anwendung tut, was diese zwar beeinflusst, aber von ihr nicht nachzuvollziehen ist, kann es zum vollständigen Versagen der Nutzerführung kommen. Dies kann schon durch einen einfachen `reload`-Befehl geschehen und ist leider kaum aufzuhalten.

Tests mit Testanwendern haben gezeigt, daß besonders die geringe Geschwindigkeit bei der Seitengenerierung schnell dazu verleitet, voreilig die unpassende Meinung zu vertreten, das System zeige eine Fehlfunktion und es sei das beste, die Seite erneut anzufordern oder eine Seite zurück zu gehen. Ein Verhalten, welches gerade der eher hektische Internetanwender von heute leider nicht selten an den Tag legt.

Nun ist es theoretisch möglich, durch eine Vielzahl von Statusvariablen und Fallabfragen ein derartiges Fehlverhalten einzudämmen, jedoch würde dies die Gesamtleistung wiederum empfindlich beeinträchtigen und den Effekt eher verstärken als verringern. Ohnehin sind auch durch derartige Aktionen nicht alle Fehlerquellen zu erfassen.

Athen reagiert auf diese Irrtümer zumeist mit einem Neuinitialisieren der Anwendung. Datenverlust tritt in der Regel nicht auf, da die Anweisungen an den Tango-Server geschickt werden und dieser seine Arbeit unabhängig von den Aktionen des Nutzers verrichtet. Jener muß lediglich die Login-Prozedur wiederholen und kann die Arbeit fortsetzen.

### **5.1.3. Bedienkomfort und Oberfläche**

An dieser Stelle gibt es wiederum einige Kritikpunkte. Die Bedienoberfläche ist vielfach nicht im eigentlichen Sinne intuitiv bedienbar und auf den ersten Blick leicht verständlich. Das hat seine Ursachen vor allem in den beschränkten Möglichkeiten der rein auf HTML und ein wenig Javascript basierten `edit`-Seite. Die Lösung, Elemente mittels vorgelagerter Radiobuttons anwählbar zu machen erfüllt ihren Zweck, ist aber wenig komfortabel, da nicht immer sofort klar ist, welches Element denn nun gemeint ist und was zum Beispiel ein Druck

auf den Menüpunkt `edit` jetzt genau bewirken wird. Dies trifft in besonderem Maße auf den oben angesprochenen „Normalnutzer“ zu, welcher ja von diesem Wissen um HTML-Strukturen, Tags und Blockelemente weitestgehend abzuschirmen war.

Hinzu kommt, daß die unterschiedlichen Interpretationsformen für die Stylesheets der `edit`-Seite durch die verschiedenen Browser es mitunter unmöglich machten, daß der betreffende Button dem Element tatsächlich direkt vorgelagert ist. Die betreffenden Kommandos sind in dieser Sequenz in HTML-Dokumenten praktisch niemals anzutreffen, weil die Art der Klasse `Seite.tcf` diese zu erzeugen, mitunter zu scheinbaren (weil ja syntaktisch auch vorhandenen) Trennungen von Button und Element führt. Dies ist besonders bei der Verwendung der Browsersoftware „Opera“<sup>37</sup> zu spüren. Hier war es leider nur selten möglich, für ein korrektes Erscheinungsbild Sorge zu tragen. Aber auch die beiden wichtigsten Systeme „Netscape“ und „Internet Explorer“ haben viele der Anweisungen in dieser notwendigen Form ignoriert<sup>38</sup>. Besonders deutlich war dies bei der testweisen Implementierung von Tabellen zu spüren, einem wichtigen Layoutinstrument im Webdesign. Darüber hinaus ist allgemein mehr Transparenz bei der Bearbeitung von Elementen wünschenswert. So kann der Nutzer zum Beispiel sehr wohl Überschriften als solche leicht hinzufügen und löschen. Will er sie aber ändern, trifft er auf die Schwierigkeit, den Block „Überschrift“ mit dessen Eigenschaften und den Text darin getrennt editieren zu müssen. Ein Umstand, der für einige Verwirrung auf Seiten des Anwenders sorgen wird.

## 5.2. Erfüllte Vorgaben

Bei all den eben genannten Einschränkungen und der dem Autor eigenen Bescheidenheit muß dennoch festgestellt werden, daß, wie gezeigt wurde, es durchaus Möglichkeiten gibt, auf Basis eines Applicationsservers wie Tango2000 HTML-Dokumente aus Datenbanken in dieser komplexen Art von Grund auf zu generieren. Die Ergebnisse möchte ich wie folgt charakterisieren:

- Das Endergebnis des Seitengenerators war stets ein sehr sauberes, allgemein verständliches und kompatibles Dokument. Ein Vorteil, der in der zusammenhängenden Datenstruktur begründet liegt. Tags ohne Abschluß, zusammenhangloser Text oder dergleichen mehr können nicht vorkommen und tun es auch nicht.

---

<sup>37</sup> <http://www.opera.com/>

<sup>38</sup> siehe Kapitel 2.1.2. „Strukturdaten“

- Die Struktur und Arbeitsweise der Seitenrepräsentationsklasse `Seite.tcf` erlaubt vielfältige Erweiterungsmöglichkeiten mit wenig Aufwand.
- Auch die Oberfläche kann leicht um unterstützte Komponenten erweitert werden, einfach indem vorhandene TAFs angepaßt werden. Die Wartung gestaltet sich unkompliziert, da die simplen Programmstrukturen der TAFs in der Regel nur sehr leicht verändert werden müssen. Lediglich die Parameterabfrageseite ist neu zu gestalten.
- Auch optische Veränderungen in den Ergebnissen sind leicht zu realisieren, indem gute und den Anwender ansprechende Stylesheetdateien geschaffen werden.
- Die Anforderungen an die Datenbank bezüglich des Datenaufkommens sind minimal.
- Die Bedienung der Oberfläche ist zwar etwas gewöhnungsbedürftig, aber im Grunde einfach und unkompliziert. Der Nutzer hat lediglich die Möglichkeit eines der Elemente anzuwählen, oder eine Aktion im Menü mit dem aktuellen auszulösen. Hinzu kommen die beiden dokumentenweit aktuellen Seiten für allgemeine Einstellungen und Stilauswahl. Aber mehr ist eigentlich auch nicht zwingend zu erlernen. Einfache „Gehversuche“ sind leicht möglich.

### 5.3. Zukünftige Verbesserungen und Erweiterungen

Für den Fall, daß sich der Auftraggeber zu einer Weiterentwicklung von Athen bis zur Serienreife entschließt, sind zahlreiche Veränderungen und logisch konsequente Weiterentwicklungen erforderlich.

#### **Oberfläche:**

- Komplette ausgestaltete Oberfläche mit ansprechendem Layout und umfassender Hilfestellung. Erklärungen aller Kommandos und Funktionen können direkt eingebettet und über Links erreichbar gemacht werden.
- Die Funktionalität der `edit`-Seite kann über Javascript erweitert werden. So sind zum Beispiel kontextsensitive Menüs und ein Ersatz der Radiobuttons möglich. In diesen ist unter Umständen eine Vielzahl der Kommandos leichter erreichbar und intuitiv verständlicher unterzubringen.
- einige Sicherheitsabfragen könnten die häufigsten der durch die in Abschnitt 5.1.2. angesprochenen Irregularitäten abfangen, die durch die Verwendung der `fwd`- und `back`-Buttons der Browser entstehen. So sind zum Beispiel Speicherung eines Statuswortes in Verbindung mit einer Nutzer-id durch ein Cookie denkbar. Dies kann verhindern, daß

erneut die Login-Prozedur durchgeführt wird, wenn der Anwender die vom Programm vorgegebenen Abläufe versehentlich unterbricht.

- Vordefinierte Seiten, aus welchen der Anwender gleichsam Templates für sein Dokument wählen kann. Dies ist recht einfach realisierbar indem einfach die betreffende Konstruktionstabelle kopiert und ein neuer Eintrag in der Tabelle Seiten erfolgt. Vorstellbar sind an dieser Stelle zum Beispiel vorbereitete Visitenkarten mit einem Bild, ein wenig Text und einer Linksammlung oder dergleichen.

### **System:**

- Eine Anpassung der Hardware an die recht hohen Bedürfnisse Athens verspricht eine Leistungssteigerung im Bereich der Seitengenerierung. Besonders ein gut angepaßtes Linuxsystem erscheint mir hier angemessen.
- Das Datenbanksystem Pervasive.SQL könnte wesentlich schneller arbeiten, wenn es sich auf dem gleichen Rechner wie der Tango2000 Server befände. Besonders im Falle eines Linuxrechners könnte die Kommunikation zwischen den beiden Komponenten direkt über einen lokalen Linux Network Socket erfolgen, was den Umweg über das TCP/IP Protokoll und Netzwerkhardware unnötig macht.
- Verbesserung der Stylesheets, insbesondere jener der `edit`-Seite. Möglicherweise ist es mit höherem Aufwand in mehr Fällen als bisher machbar, die `inline`-Darstellung der Radiobuttons zu erzwingen und auch generell ein besseres Gefühl für das Gestalten der Seite zu vermitteln. Zum Beispiel über automatisch eingefügte Symbole und Strukturelemente.
- Die Generierung der `edit`-Seite durch den Server könnte mit Cachemechanismen ausgestattet werden, was ein erneutes Erzeugen der ganzen Seite nach der Veränderung eines Blockes unnötig machen würde. Unter Umständen ist diese zeitkritische Aufgabe im Falle eines Umlagerens auf ein javafähiges System auch von einem eingebundenen JavaBean oder auch einer DLL (Windows) ausführbar, was allerdings tiefgreifende Veränderungen im System erfordern würde.
- Unterstützung umfangreicherer Komponenten als nur simpler Tags und Blöcke. Es können zum Beispiel gleich vollständige, vordefinierte Strukturelemente, Javascripts oder auch Java-Applets bereitgestellt werden. Anwendungsbeispiele sind z.B. Seitencounter oder die beim Nutzer oft recht beliebten graphischen Scripts zur Veränderung des Mausverhaltens oder dergleichen mehr.

# Anhang A

## Thesen

1. Applicationserver bieten eine leistungsfähige und flexible Schnittstelle zwischen Datenbanken und Webserver.
2. Es ist möglich, auch komplexe Applikationen mit Hilfe einer solchen Software einer breiten Anwenderschaft anzubieten.
3. Der Applicationserver Tango2000 bietet zu diesem Zweck eine komfortable Plattform.
4. Die Struktur von HTML-Dokumenten läßt es zu, daß diese vollständig aus Datenbanken heraus generiert werden können.
5. In diesem Sinne kann ein webserverbasierter HTML-Editor entworfen und entwickelt werden.

# Anhang B

## Glossar und Abkürzungsverzeichnis

### A

Apache: populärer →Webserver, der für verschiedene Systeme erhältlich ist, aber vor allem im UNIX- und Linux-Bereich sehr verbreitet ist. Der Name leitet sich von „A PAtCHy server“ und bezieht sich auf ein paar Codefragmente und Patches, „mit denen alles begann“.

Applicationserver: Allgemeiner Terminus für einen →Server, dessen Aufgabe es ist, Verbindungen zwischen Nutzer und Datenbanksystem herzustellen

### B

Browser: umgangssprachlich für Clientsoftware, die zum Durchforsten der Inhalte des →World Wide Web (WWW) dient. Der Browser stellt →HTML-Seiten dar und leitet über →Links weiter. Derzeit existieren zwei wichtige Systeme. Der Netscape Navigator und der Internet Explorer der Microsoft Corporation. Möglicherweise wird demnächst als Ergebnis des Mozilla-Projektes etwas mehr Konkurrenz auftreten.

### C

CGI: Common Gateway Interface. Das allgemein verbreitetste Mittel eines →Webservers, mit einem CGI-Programm zu interagieren, welches seinerseits in vielen Sprachen (oft Scriptsprachen) geschrieben sein kann.

Child: allgemein umgangssprachliche Kurzbezeichnung für ein Element, welches von einem anderen abhängig bzw. in ihm enthalten ist. (wörtlich „Kind“)

Client: wörtlich „Kunde“; Allgemeine Bezeichnung für die empfangende und fordernde Seite einer Client-Server-Architektur.

Cookie: „Keks“; kleine Datenvariable, die ein →Webserver im →Browser des Anwenders unterbringen und bei Bedarf lesen kann.

CORBA: Common Object Request Broker Architecture. Eine Architektur, die es Programmteilen ermöglicht, miteinander zu kommunizieren, unabhängig davon, wo sie sich befinden oder in welcher Sprache sie geschrieben sind. Gängige Implementierungen sind „SOM“ und „DSOM“ von IBM.

CSS: Cascading Style Sheets; Definitionssprache zur optischen Spezifizierung des Erscheinungsbildes von →HTML-Seiten.

## D

Data Warehouse: Datenbankoberfläche, die Daten nicht nur anbietet, sondern auch zur leichteren Entscheidungsfindung aufbereitet. „Informations-selbstbedienung“

Datenkapselung: Begriff aus der objektorientierten Programmierung. Ein Objekt (→Klasse) verbirgt seine spezifischen Daten vor der Außenwelt und bewahrt sie so vor unberechtigtem oder unkoordinierten Zugriff.

Datenquelle: Begriff aus der Welt der →Middleware und →ODBC. Eine Datenquelle bildet eine allgemein leicht zugreifbare Schnittstelle über ein Netzwerk zu einem Datenbankserver.

DLL: Dynamic Link Library; unter dem Betriebssystem Windows übliche Bezeichnung für Programmbibliotheken, die dynamisch in den Programmablauf eingebunden werden können.

DNS: Domain Name Service; Internetdienst, der Domainnamen in →IP-Adressen konvertiert. (www.middleware.org → 206.253.222.74)

## H

Header: Allgemeiner Ausdruck für einleitende oder beschreibende Teile eines Protokolls, Programms oder Dokuments.

Hyperlink: Verweis auf andere HTML-Dokumente innerhalb einer Seite mittels eines →URL. (siehe auch →Link)

## F

Flaschenhals: ugs. für eine kritische Stelle oder Operation in Programm- oder Aktionsabläufen.

Frameset: Layoutinstrument in Webseiten. Ein Frameset teilt den Bildschirm des Benutzers in Abteilungen, in denen wiederum eigene Seiten angezeigt werden können.

## G

Firewall: ugs. Schutz eines Lokalen Netzwerkes oder Rechners vor unerwünschten Zugriffen aus dem Internet. Hard- oder Softwarelösungen sind vorhanden.

## H

HTML: Hypertext Markup Language. Allgemeine Seitenbeschreibungssprache des World Wide Web. Nähere Informationen in Kapitel 2.

Hyperlink: Ein Verweis auf einen →URL innerhalb von Webdokumenten.

## I

Instantiierung: Erzeugung eines →Objektes einer →Klasse. Aus der abstrakten Beschreibung wird eine konkrete Instanz.

IP: Kurzform für eine IP-Adresse im Internetprotokoll TCP/IP.

ISP: Internet Service Provider; ein Dienstleister für netzwerkrelevante Dinge wie Einwahlknoten, Zugangssoftware und →Proxydienste.

## J

Java: Eine plattformunabhängige. Programmiersprache, deren Programme auf einer simulierten, virtuellen Maschine laufen. Von der Firma Sun

entwickelt und heute sehr verbreitet schickt sich Java an, Standardlösung für viele Fälle im Bereich netzwerkrelevanter Software zu werden.

Javascript: Mit →Java verwandte Scriptsprache, die aber im Gegensatz zu dieser nicht kompiliert wird sondern als Interpretersprache dient. Haupteinsatz ist die programmtechnische Aufwertung von Webdokumenten.

## K

Klasse: Begriff aus der objektorientierten Programmertechnologie. Eine Klasse ist die abstrakte Beschreibung eines →Objektes und dessen Eigenschaften.

## L

Layout: Generelle Eigenschaften des optischen Erscheinungsbild eines oder mehrerer Dokumente.

Link: Verweis; meist verwendet innerhalb von Webseiten.

Logging: Das Protokollieren der Aktionen eines bestimmten Systems, meist eines →Servers, welches nicht auf konkrete Anweisungen des Administrators reagiert sondern auf Anforderungen eines →Clients.

## M

Metatag: Auch innerhalb von Tango gebrauchte Terminologie zur Umschreibung eines Strukturelements in einem Dokument, welches vor dem Versand der Seite durch Aktionen des Servers in konkrete Daten umgewandelt wird (Berechnung, Datenbankabfragen, Konstanten, etc.).

Member: „Mitglied“; ein Member ist ein Angehöriger einer Klasse, zum Beispiel eine Eigenschaft (auch „Property“), die nach der →Instantiierung mit einem Wert versehen wird. Man unterscheidet Membervariablen und Memberfunktionen (→Methoden).

Methode: siehe →Member

MicroKernel: Systemprogrammieretechnologie, welche sich durch modulare, kleine Systemkerne auszeichnet, die leicht durch weitere Module erweitert werden können. Im Gegensatz dazu stehen monolithische Kerne.

Middleware: Vermittlerschicht zur Kommunikation zwischen Servern und Datenbanken. Mehr Informationen sind in Kapitel 3.1. „Middleware und Applicationserver“

## O

ODBC: Open Database Connectivity; von der Microsoft Corporation entwickelte Zugriffsmethode, die es jeder Applikation erlauben sollte, mit jeder Datenbank zu kommunizieren, solange diese über einen ODBC-Treiber verfügt.

## P

Parameter: Die Argumente und speziellen Umstände eines Funktions- oder Methodenaufrufes.

Parent: Gegenstück zum →Child. Das einem Element in einer Baumstruktur übergeordnete.

Plattform: Die Zusammenfassung von Computer, Betriebssystem und sonstigen Merkmalen zu einem virtuellen Gebilde, auf welchem Software eingesetzt wird.

Plugin: Programmmodul welches als Teil eines größeren agiert und von diesem aufgerufen wird. Es fügt dem Hauptprogramm eine nützliche Eigenschaft hinzu.

Proxy: „Vermittler“ zwischen →Clients in einem lokalen Netzwerk (LAN) und den →Servern des Internet.

## Q

Query: Allgemeines Kurzwort für eine Datenbankabfrage (z.B. SQL-Statement).

## R

rekursiv: „in eine Baumstruktur hinein verzweigend“. Ein Adverb, welches bezeichnet, daß eine Aktion nicht nur für das betreffende Element sondern auch für alle Child und deren Child usw. erfolgen soll. (Auch eine Algorithmierstrategie)

Runtime Engine: Unterstützungsinterpreter oder in diesem Fall virtuelle Java-Maschine (VM), die es ermöglicht, die erzeugten Programme auf einer konkreten →Plattform lauffähig zu machen.

## S

Schlüssel: Im Datenbankbereich ein Identifikator eines Tabellenelements (Entity). Für vorliegenden Falls sind Primärschlüssel relevant (Primary Key), welche von Pervasive.SQL durch den Datentyp `identity` repräsentiert werden.

Scope: Gültigkeitsbereich einer Tango-Variablen (z.B. `user`, `cookie`, `method`)

SDK: Software Developer Kit; Sammlung von Tools, die es im Gegensatz zu einer →Runtime Engine nicht nur ermöglichen, Programme auszuführen, sondern auch zu entwickeln.

Server: „Diener“; Programm oder Rechner, dessen Aufgabe es ist, →Clients im Netzwerk Dienstleistungen zur Verfügung zu stellen.

Servlet: Javaklasse, die auf Seiten und zur Unterstützung eines Webservers dynamisch Webdokumente kreiert. Servlets sind im Grunde als Applicationserver einsetzbar.

Socket: Softwareobjekt zur Verbindung einer Applikation mit einer Netzwerkverbindung. Allgemein ein Kommunikationsmittel zwischen Anwendungen im Netzwerk.

SQL: Structured Query Language; Standardsprache zur Interaktion mit einer Datenbank.

## T

Tag: „Marker“; strukturelles Beschreibungselement (z.B. <H1> oder <p>).

TAF: Tango Application File; Die vom Tango Server verstandenen XML-Dokumente, die dieser auszuführen vermag.

TCF: Tango Class File; wiederverwendbares Programmmodul, angelehnt an das Klassenkonzept der objektorientierten Programmierung.

Transaktion: Absicherungskonzept zur Kapselung und referentiellen Integritätssicherung von Datenbankabfragen, indem nichtatomare Anfragen (→Queries) zwischengespeichert ausgeführt werden können.

## U

Unicode: neues Darstellungsset für Buchstaben, welches 16, statt der 8 für ASCII verwendeten Bit pro Zeichen enthält.

URL: Uniform Resource Locator; Eindeutige vollständige Internetadresse inklusive Übertragungsprotokoll (Dienst), Serveradresse, Subpfad und evtl. Dokumentenname. ( z.B.: <http://www.fh-zwickau.de/index.html> )

## W

Webapplikation: Gesamtheit von Zusammengehörigen Funktionen eines →Applicationservers, die dem Anwender eine einzige Applikation vorspiegeln. Athen ist eine solche Webapplikation.

## X

XML: Extended Markup Language; Erweiterte und ergänzende Sprachspezifikation zu →HTML. XML ist wesentlich flexibler als dieses und kann neben Dokumenten auch andere Datenstrukturen beschreiben.

# Anhang C

## Quellenverzeichnis

Die überwiegende Rechercharbeit fand in den Informationsweiten des Internet statt. Aus diesem Grund sind die Mehrzahl der aufgeführten Quellen URLs auf Websites, in denen relevante Daten zu finden sind, oder auf die schon früher im Text verwiesen wurde.

Die Links wurden am 10. Mai des Jahres 2001 auf ihre Aktualität überprüft.

1. <http://www.heise.de>
2. <http://www.pervasive.com>
3. <http://www.apache.org>
4. <http://www.webopedia.com>
5. <http://www.independent.co.uk/news/Digital/Update/2000-12/internet051200.shtml>  
(Dieser URL ist leider nicht mehr verfügbar)
6. <http://www.w3.org/Consortium/>
7. <http://purl.oclc.org/dc/>
8. Das Benutzerhandbuch zu Tango2000, angeboten zum Download unter der URL Nr. 2
9. Nutzerdokumentation zu Pervasive.SQL, ebenfalls dort erhältlich
10. <http://www.w3.org/TR/REC-xml>
11. <http://www.apple.com>
12. <http://www.starnine.com>
13. <http://jakarta.apache.org/tomcat/index.html>
14. <http://java.sun.com>
15. „SQL - Der Schlüssel zu relationalen Datenbanken“; Gregor Kuhlmann u. Friedrich Müllmerstadt; Rowohlt Taschenbuchverlag 1999
16. <http://www.w3.org/TR/html4/struct/global.html>
17. <http://www.netzwelt.com/selfhtml/>
18. <http://www.opera.com/>
19. <http://www.microsoft.com>
20. <http://www.middleware.org>
21. „Go to Java 2 – Handbuch der Javaprogrammierung“; Guido Krüger; Addison-Wesley Verlag 2000

# **Anhang D**

## **Erklärung zur selbständigen Anfertigung**

hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig und lediglich unter Verwendung der angegebenen Quellen angefertigt habe.

Leipzig, der 10. Mai 2001

# Anhang E

## Quellcodes

An dieser Stelle möchte ich darauf hinweisen, daß ich mir die Freiheit nehme, keine Quellcodes an diese Arbeit anzuhängen. Grund dafür ist der Umstand, daß die vom Tango Editor generierten XML-Dateien erstens sehr groß (allein `Seite.tcf` umfaßt 77 KBytes) und zweitens ohnehin kaum lesbar und verständlich sind. Nach der Meinung des Autors erfüllen die Programmablaufpläne im Kapitel 4.6. besser diesen Zweck.

Auf Wunsch können jedoch Auszüge der Quellen unter der Adresse [Stephan.Menzel@bigfoot.de](mailto:Stephan.Menzel@bigfoot.de) angefordert werden.