Fachhochschule Konstanz
Fachbereich Informatik

Diplomarbeit

Zur Erlangung des Akademischen Grades

Diplom-Informatiker (FH)

an der

Fachhochschule Konstanz
Hochschule für Technik, Wirtschaft und Gestaltung

Fachbereich
Informatik

Studiengang
Technische Informatik

Thema: **Implementierung einer horizontalen Baumdarstellung als JavaBean unter der Verwendung des SVG(XML) Grafikformates.**

Diplomand: Thomas Wolfschläger, Rheingutstraße 36/O-117, 78462 Konstanz

Firma: Dornier GmbH, EADS Systems & Defence Electronics,
88039 Friedrichshafen, Germany

Betreuer: Professor Dr.-Ing. Jürgen Neuschwander
Dipl.-Inf. Christopher Schmidt

ABSTRACT

Im Rahmen dieser Diplomarbeit wird eine horizontale Baumkomponente als JavaBean erstellt. Die Baumkomponente soll als Truppenbaum in ein bestehendes Führungs-Informationssystem implementiert werden. Dazu ist die Verwendung des SVG (XML) Grafikformates zur Darstellung der Symbole einzelner Knoten erforderlich.

Weiterhin wird die Interaktion von Java-Komponenten mit Windows COM und DCOM untersucht. Es werden mehrere unterschiedliche Java-COM-Bridges getestet.

In der vorliegenden Arbeit wird zuerst auf die Grundlagen eingegangen, indem das XML und SVG Format vorgestellt wird und die Grundlagen von JavaBeans sowie die benutzten Bibliotheken erläutert werden. Anschließend wird die Entwicklung der Komponente mit Pflichtenheft, Architektur und Implementierung dargestellt. Abschließend werden die Java-COM-Bridges beschrieben.

EHRENWÖRTLICHE ERKLÄRUNG

Hiermit erkläre ich, Thomas Wolfschläger, geboren am 14.06.1969 in Ulm-Söflingen, ehrenwörtlich,

(1) dass ich meine Diplomarbeit mit dem Titel:

„Implementierung einer horizontalen Baumdarstellung als JavaBean unter der Verwendung des SVG (XML) Grafikformates“

bei der European Aeronautic Defence and Space Company (EADS) unter Anleitung von Professor Dr.-Ing. Jürgen Neuschwander und Dipl.-Inf Christopher Schmidt selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

KONSTANZ, 14. APRIL 2003

INHALTSVERZEICHNIS

Kapitel 1

EINLEITUNG

Der technische Fortschritt kann nicht aufgehalten werden! Damit kommt es zu einer wahren Flut von ständig und immer schneller wachsendem Wissen. Aber kann die Menschheit mit dieser Entwicklung Schritt halten? Diese Frage stellt man sich heute immer häufiger. Und schon früh hat man erkannt, dass gerade dem erfolgreichen Management von Informationen eine Schlüsselrolle bei der Lösung dieses Problems zukommen wird.

Es wird immer wichtiger, in allen Bereichen unserer Umwelt Systematiken zu finden, mit denen sich Daten und Wissen strukturieren lassen.

Was wäre ein Computeranwender, wenn es keine Möglichkeit gäbe, Dateien in einem System von Ordnern zu speichern? Das Wiederauffinden von bestimmten Dokumenten auf der Festplatte wäre ein nervenaufreibender Zeitvertreib.

Aber auch mit einer Systematik ist diese Aufgabe kein Kinderspiel. Erst die grafische Darstellung der Ordnerstruktur, beispielsweise in einer vertikalen Baumstruktur im Windows-Explorer, ermöglicht ein schnelles Finden der Dateien. Dies ist damit zu erklären, dass der Mensch komplexe Zusammenhänge meist schneller erfasst, wenn sie grafisch dargestellt sind.

Besonders effizient sind Systematiken in der Regel dann, wenn sie hierarchisch aufgebaut sind. Dies sieht man beispielsweise auch an der Gliederung einer Diplomarbeit in Kapitel und Abschnitte, an Bibliothekskatalogen, Unternehmenshierarchien oder auch der Gliederung von militärischen Organisationseinheiten. Auch das System von Ordnern auf unserem Computer ist hierarchisch angelegt.

Nicht immer entspricht die Darstellung einer Hierarchie als vertikale Baumstruktur (wie im Datei-Explorer) der intuitiven Auffassung des Menschen. Vielmehr würde man eine Unternehmensstruktur oder auch einen Stammbaum eher in einem horizontalen Baum darstellen. Daraus lassen sich für den Menschen die Strukturen oder die Rangordnung eines angezeigten Datums einfacher erkennen.

Im Rahmen eines von der Firma Dornier EADS entwickelten computergesteuerten Führungssystems für die Armee der Vereinigten Arabischen Emirate, soll die Darstellung von Truppenbäumen in einer horizontalen Form erfolgen. Die Beschreibung und Erstellung eines Tools zur Erstellung und Darstellung von horizontalen Bäumen ist Hauptbestandteil dieser Diplomarbeit.

1.1 Dornier EADS

Die European Aeronautic Defence and Space Company (EADS) ist der größte Luft- und Raumfahrtkonzern Europas und der zweitgrößte weltweit. EADS ist in den Bereichen Zivil- und Militärluftfahrt, Raumfahrt, Verteidigungssysteme und Serviceleistungen tätig. Das Unternehmen entstand durch die Fusion der deutschen DaimlerChrysler Aerospace AG, der französischen Aerospatiale Matra und der spanischen CASA am 10. Juli 2000.

1.2 Aufgabenstellung

Das zentrale Thema dieser Diplomarbeit ist die Implementierung einer horizontalen Baumdarstellung als JavaBean unter Verwendung des SVG(XML) Grafikformates. Die Bean soll in ihrer Interaktion gleich oder zumindest ähnlich dem `JTree` von Java Swing sein. Sie soll auch das gleiche Datenmodell wie der `JTree` nutzen.

Beans definieren ein Software-Komponentenmodell für Java, so dass sie sich von Entwicklern zu Anwendungen „zusammenbauen“ lassen. Es soll weiterhin die Verfügbarkeit von Java-Komponenten für andere Anwendungen und Programmiersprachen unter Windows mit den Technologien Java Bridge für Active-X und CAS COM Bridge untersucht werden.

1.3 Gliederung

Diese Diplomarbeit gliedert sich in einen theoretischen Teil, welcher die Grundlagen erörtert und die benutzen Werkzeuge und Bibliotheken auflistet. Der nächste Teil behandelt die Softwarekomponente. Es wird das Pflichtenheft, die Architektur und die Schritte der Verifikation vorgestellt. Abschließend wird die Wiederverwendbarkeit in der Windowswelt beleuchtet.

Kapitel 2

EXTENSIBLE MARKUP LANGUAGE (XML)

2.1 Definition

XML steht für *eXtensible Markup Language* und bedeutet wörtlich übersetzt soviel wie „erweiterbare Textauszeichnungssprache“. Es ist ein internetfreundliches Format zur Darstellung von Daten und Dokumente, welches von W3C entwickelt bzw. standardisiert wurde. Der Ursprung von XML liegt in SGML, welches ebenfalls eine Textauszeichnungssprache ist und sehr häufig im Verlagswesen benutzt wird.

SGML ist ein internationaler Standard zur Beschreibung von Informationenstrukturen. Es beschreibt die inhaltlichen Aufbau eines Dokumentes und nicht das Layout. Die wohl bekannteste SGML Anwendung ist HTML, XML ist im Gegensatz zu HTML keine Anwendung, sondern eine Teilmenge von SGML.

XML hat keine vordefinierten Elemente, es definiert lediglich Regeln, welche es ermöglichen, andere Sprachen wie HTML oder SVG zu erstellen.

2.2 Wozu dient XML?

Wie oben erwähnt, wird die Metasprache[1] XML zur Definition von Textauszeichnungssprachen benutzt. So wie HTML mit SGML definiert ist, so kann man mit XML eigene Textauszeichnungssprachen definieren. Künftige HTML-Versionen sollen ebenfalls auf der Grundlage von XML definiert (XHTML) werden. In der Weise, wie HTML festgelegt und normiert ist, dient es hauptsächlich dem weltweiten Austausch sowie der Übertragung und Verwendung von Webseiten zwischen vielen verschiedenen Webserver und -browser. Ähnlich kann man mit XML eigene Strukturen für unterschiedliche Zwecke definieren und normieren. Solche Strukturen können dann auch von vielen Anwendern mit verschiedenen Programmen auf vielen verschiedenen Rechnern verwendet werden.

---

[1] Eine Sprache, die für die Definition anderer Sprachen verwendet wird. Oft auch als Sprachbeschreibungssprache bezeichnet.

Mit der Hilfe von Style-Sheets können XML-Dokumente gleich wie HTML-Dateien von Webbrowsern dargestellt werden.

- Mit XML kann man die logische Bedeutung von Daten, Informationen und Texten definieren - ähnlich wie die Tabellen- und Spaltenbezeichnungen in Datenbanken und Tabellenkalkulationen.
- XML ermöglicht im Gegensatz zu HTML die Definition eigener oder zusätzlicher Elemente (Tags) - ähnlich wie bei der Definition von Makros in der Textverarbeitung.
- XML-Anwendungen eignen sich als plattform- und softwareunabhängiges Format für den Austausch von Daten zwischen verschiedenen Programmen und Rechnern - ähnlich wie RTF für Texte, CVS für Tabellen und EDI für kommerzielle Anwendungen - aber in einem einheitlichen, allgemein verwendbaren und herstellerunabhängigem Format.

Weiterhin ist die Syntax von XML so streng festgelegt, dass solche Anwendungen wesentlich einfacher, bequemer und effizienter von Programmen weiter verarbeitet werden können als zum Beispiel HTML-Dateien.

2.3 Syntax

Natürlich ist die Syntax von XML schon in vielen Büchern und auch im Internet ausreichend dargestellt und erklärt worden. Die heutzutage wohl bekannteste Auszeichnungssprache ist HTML. Hier will ich noch ein paar wenige syntaktische Differenzen zu HTML erläutern.

2.3.1 Start- und Ende-Tags

Die meisten Elemente (Tags) in SGML- und XML-Anwendungen - wie auch in HTML - treten paarweise als Start- und Ende-Tags auf. Sie geben an, welche Bedeutung der dazwischen liegende (eventuell durch weitere Tags unterteilte) Text hat:

```
<xxx> ... Text ... </xxx>
```

In HTML ist das Ende-Tag in vielen Fällen optional, d.h. es darf weggelassen werden. Der Webbrowser kann dann aufgrund der in der HTML-Norm festgelegten Bedeutung der Tags „erraten“, an welcher Stelle sich der Browser das nicht angegebenen Ende-Tag hinzu „denken“ muss.

Beispiel: Das HTML-Tag `<p>` beginnt einen neuen Absatz. Das Ende-Tag `</p>` muss nicht angegeben werden, denn immer, wenn ein neuer Absatz mit `<p>`, eine Überschrift mit `<h1>` bis `<h6>` oder eine Liste mit `<ul>`, `<ol>` oder `<dl>` beginnt, bedeutet dies automatisch das Ende des vorherigen Absatzes.

Bei XML-Anwendungen müssen (im Gegensatz zu HTML) die Ende-Tags immer angegeben werden:

Richtig ist:

```
<p> ... Text ... </p><p> ... Text ... </p>
```

Falsch wäre:

```
<p> ... Text ... <p> ... Text ...
```

Diese „strengere“ Regel hat unter anderem die folgenden Gründe: Im Gegensatz zur festgelegten HTML-Norm soll der Anwender bei XML-Anwendung die Möglichkeit haben, nachträglich zusätzliche Tags in der DTD (*Document Type Definition*: beschreibt die Struktur einer Klasse von XML- oder SGML-Dokumenten) oder im XML-Schema[2] zu definieren, und daher können die Verarbeitungsprogramme nicht, so wie der Webbrowser es bei HTML macht, immer nach denselben Regeln die fehlenden Ende-Tags selbst korrigieren. Weiterhin sind die verarbeitenden Programme viel einfacher zu schreiben und können viel effizienter ablaufen, wenn sie sich darauf verlassen können, dass die XML-Datei syntaktisch richtig ist und keine automatische Fehlerkorrektur programmiert werden muss. Doch was geschieht mit den Befehlen, welche keine Ende-Tags besitzen?

2.3.2 Einzelne Elemente ohne Ende-Tag

In HTML gibt es Tags, zu denen es kein Ende-Tag gibt, weil sie nicht die Eigenschaften eines Textbereiches definieren sondern einzelne und selbständige Elemente darstellen. Beispiele hierfür sind `<br>` für einen Zeilenwechsel, `<hr>` für eine Trennlinie oder auch `<img>` für ein Bild.

In XML müssen alle Tags der Form `<xxx>` immer ein Ende-Tag der Form `</xxx>` haben. Die Elemente, zu denen es kein Ende-Tag gibt, müssen zur Unterscheidung in der Form `<xxx/>` geschrieben werden.

Nicht erlaubt ist also

```
<p> ... <br> ... Text ... <br> ... Text ... </p>
```

sondern man muss stattdessen richtigerweise

```
<p> ... <br/> ... Text ... <br/> ... Text ... </p>
```

oder eventuell auch

```
<p> ... <br></br> ... Text ... <br></br> ... Text... </p>
```

schreiben.

[2] XML-Schemata dienen der eindeutigen Beschreibung und Verifizierung von Daten in einer XML-Umgebung.

Der Grund für diese „strengere“ Regel liegt ebenfalls in einer Vereinfachung der Programmierung: Das Verarbeitungsprogramm muss nicht darauf „warten“, welche Ende-Tags eventuell noch kommen, sondern weiß immer sofort, ob es sich um ein einzelnes Element oder um das Start-Tag eines längeren Elementes handelt.

2.3.3 Groß- und Kleinschreibung

Im Gegensatz zu HTML ist bei XML die Groß- und Kleinschreibung nicht egal: `<xxx>` und `<Xxx>` und `<XXX>` sind verschiedene Elemente. Wenn also ein Element als `<xxx>` definiert ist, dann darf man nicht stattdessen `<XXX>` schreiben, und auch eine Kombination wie `<XXX> ... Text ... </xxx>` wäre nicht erlaubt.

2.3.4 Parameter (Attribute)

Mit Attribute bezeichnet man die Parameter, die innerhalb eines Elements auftreten können. Mit ihrer Hilfe werden die Eigenschaften von Elementen näher bestimmt.

Die Regeln, wann die Werte von Attributen zwischen Anführungszeichen eingeschlossen werden müssen, sind bei XML strenger als bei HTML üblich. Alle Attributwerte werden immer in Anführungszeichen geschrieben, unabhängig vom Typ der Werte. Dabei darf entweder das ASCII-Zeichen " (Anführungszeichen) oder das ASCII-Zeichen ' (Apostroph) verwendet werden (nur paarweise, nicht vermischt), aber nicht die ähnlich aussehenden typographischen Anführungszeichen oder Akzentzeichen. Es werden Strings genauso in Hochkommas angegeben wie Zahlen.

Richtig sind also nur die folgenden beiden Varianten:

```
<xxx  attribut="wert"  ...  >
<xxx  attribut='wert'  ...  >
```

2.3.5 Entities

Entity wird in SGML-konformen Auszeichnungssprachen ein Zeichen genannt, dass über den Standard ASCII-Zeichensatz hinausgeht. Zu diesen gehören z.B. die deutschen Umlaute ä, ö, ü und das ß oder andere mit Akzenten versehene Buchstaben.

Wie in SGML (und daher auch in HTML) kann man auch in XML Entities definieren, bei denen einem Namen ein bestimmter Text zugeordnet wird. Diese Entities kann man dann überall im Text und auch in Parametern von Befehlen in der Form `&name;` verwenden.

Typische Beispiele sind:

- `<` für das „kleiner“ Zeichen <
- `>` für das „größer“ Zeichen >

- `&` für das „und“ Zeichen &

Entities können aber nicht nur einzelne Zeichen sondern auch längere Textteile enthalten.

2.3.6 Verschachtelung von Tags

In XML müssen die Start- und Ende-Tags immer richtig geschachtelt werden.

Richtig sind:

```
<person> ... <vorname> ... </vorname> ... </person>
<vorname> ... </vorname> <zuname> ... </zuname>
```

Nicht erlaubt wären:

```
<person> ... <vorname> ... </person> ... </vorname>
<vorname> ... <zuname> ... </vorname> ... </zuname>
```

(Anmerkung: Diese Regeln gelten bei HTML und bei XML).

Bei XML darf die Software solche Fehler nicht durch „rateversuche“ korrigieren, sondern muss die Verarbeitung von fehlerhaften XML-Dokumenten immer mit einer Fehlermeldung abbrechen.

Ein Grund für diese „strenge“ Regel ist, dass die XML-Befehle nicht nur das Layout des Textes (wie bei HTML), sondern die Bedeutung der Textteile definieren. Falsche Interpretationen der Befehle führen daher nicht bloß zu unschönen, sondern zu logisch und inhaltlich falschen Ergebnissen.

Die XML-Tags dienen nicht nur der Interpretation sichtbarer Texte, sondern auch der Weiterverarbeitung der Informationen durch automatisch laufende Programme. Deshalb müssen die syntaktischen Regeln von XML strenger eingehalten werden als die von HTML. Soweit zu den wichtigsten Unterschieden zwischen HTML und XML. Welche fertige XML Anwendungen existieren und was benötigt wird, um eigene XML-Applikationen zu erstellen zeigen die folgende Kapitel.

2.4 XML-Anwendungen

Wie man bereits feststellen konnte, ist XML keine eigene Applikation, sondern beschreibt Regeln und Normen, um eigene Anwendungen zu erstellen. Nach der Veröffentlichung der XML-Spezifikation wurde eine wahre Flut von XML-Anwendungen vorgeschlagen, und es ist noch kein Ende abzusehen. Nachfolgend werde ich ein paar wichtige Applikationen kurz vorstellen:

- **Mathematical Markup Language (MathML)**: Sprache zur Beschreibung und Darstellung mathematischer Formeln, sog. semantische Tags [1].
- **Extensible 3-D (X3D)**: Sprache zur Beschreibung und Abbildung von 3-D VR-Grafikobjekten [2]. X3D ist der Nachfolger von VRML97.

- **Synchronized Multimedia Integration Language (SMIL)**: Sprache zur Integration und Synchronisation von Multimedia-Objekten [3].

- **Scalable Vector Graphics (SVG)**: Sprache zur Beschreibung und Abbildung von 2-D Grafikobjekten wie Vektorgrafikelemente, Texte und Bilder [4]. Auf diesen Standard werde ich später noch im speziellen eingehen (siehe Kapitel Kapitel 3).

Der Interpretation von XML-Anwendungen dient das DOM, welches im folgenden Abschnitt beschrieben wird.

2.5 Document Object Model (DOM)

Das *Document Object Model* wird vom W3C definiert. Es stellt den Aufbau eines XML-Dokumentes in einer Baumstruktur dar, welche sich dann über das DOM-API auslesen oder editieren lässt.

Zunächst einmal beschreibt es den Scriptsprachenzugriff auf beliebige Elemente eines Auszeichnungssprachen-Dokuments. Das DOM ist keine eigene Scriptsprache. Es definiert lediglich Objekte, Eigenschaften und Methoden, die eine Scriptsprache umsetzen sollte, wenn sie sich DOM-fähig bezeichnen will. Anwendbar sollen diese Objekte, Eigenschaften und Methoden auf alle Dokumente sein, die in einer XML-gerechten Auszeichnungssprache geschrieben sind.

Das W3-Konsortium betont ausdrücklich, dass das DOM nicht einfach nur eine Norm für „Dynamic HTML“ sein soll. Das DOM ist auch nicht auf die Client-Seite, also etwa den Webbrowser beschränkt. Ebenso gut lässt es sich in serverseitigen Scripts, z.B. in CGI-Scripts einsetzen, um Dokumente dynamisch zu erzeugen.

Das DOM definiert mehrere Interfaces (Schnittstellen), mit denen die Daten in einer Baumstruktur beschrieben werden können. Das wichtigste davon ist die Schnittstelle Node, von der alle anderen Interfaces abgeleitet sind. Das folgende Klassendiagramm macht dies deutlich:

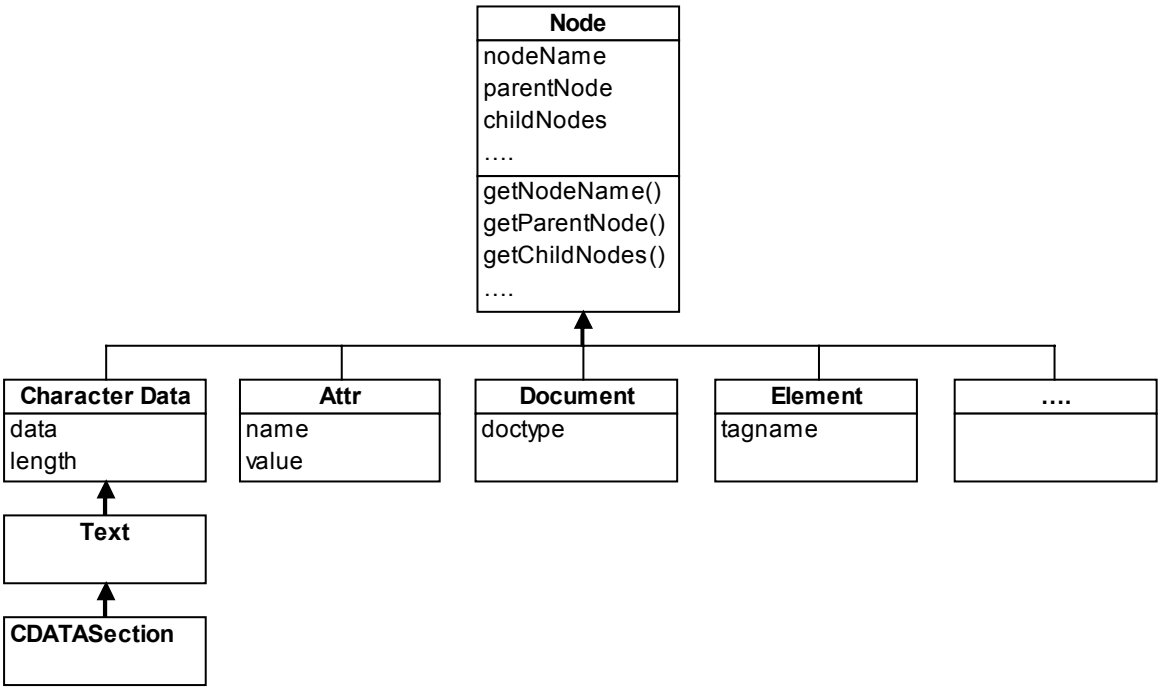


Abbildung 2-1: Baumstruktur der DOM-Schnittstelle

Das Node-Objekt besitzt dabei sämtliche Methoden, welche zum Traversieren des Baumes nötig sind. In den abgeleiteten Klassen müssen nur noch die spezifischen Eigenschaften gespeichert werden (z.B. die Eigenschaft „tagname“ im Interface „Element“). Selbstverständlich gibt es noch andere von Node abgeleitete Schnittstellen als die vier in Abbildung 2-1 aufgeführten. Dazu gehören noch Notation, Entity, EntityReference und ProcessingInstruction.

Zum besseren Verständnis betrachten wir folgendes kleine Beispiel eines einfachen XML-Dokuments:

Beispiel:

```
<?xml version="1.0"?>
<!--DOM Demo-->
<xdoc>
  <Begruessung>Hallo <laut>XML</laut>Parser!</Begruessung>
  <Applaus art="anhaltend"/>
</xdoc>
```

Daraus ergibt sich folgender DOM-Baum:

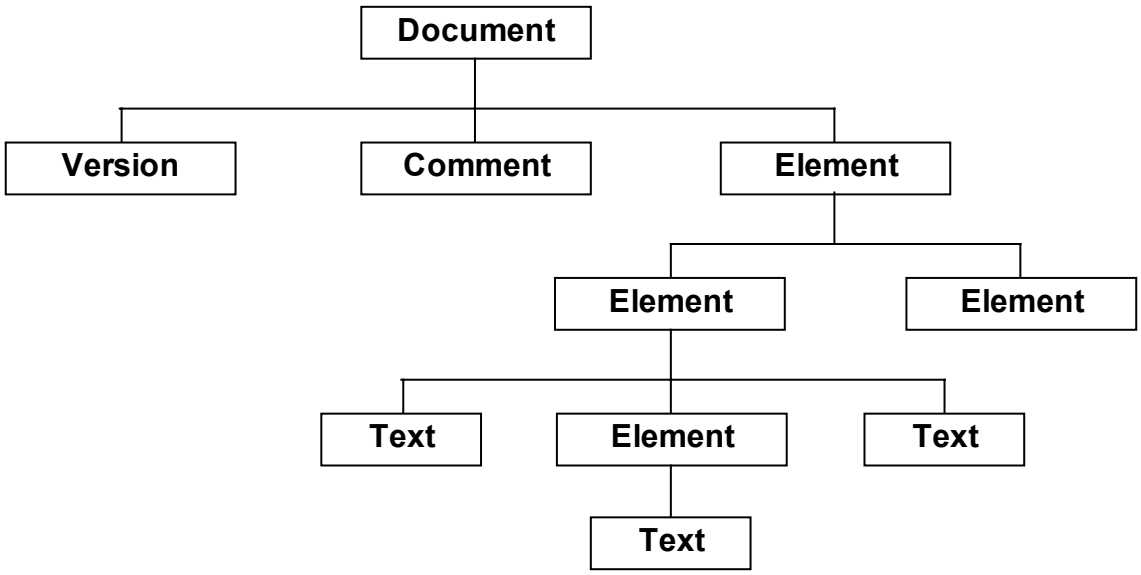


Abbildung 2-2: Einfacher DOM-Baum

Die Wurzel eines DOM-Baumes ist immer ein *Document*. Dieses hat im oben gezeigten Beispiel drei Kind-Elemente: *Version*, *Comment* und *Element*. Die ersten beiden dürfen selber keine Kind-Elemente haben, aber *Element* hat wiederum zwei Kind-Elemente und so setzt sich der Baum fort.

Weiter möchte ich an dieser Stelle nicht auf das DOM eingehen, da das Parsen von bestehenden XML-Dokumenten innerhalb von Java oder C++ mit SAX weit mehr im Vordergrund dieser Arbeit steht.

2.6 Simple API for XML (SAX)

Analog zu DOM definiert auch SAX Schnittstellen für das Parsen von XML-Dokumenten. Allerdings ist SAX kein W3C-Standard, sondern wurde von der XML-DEV Mailinglist entwickelt und Anfang 1998 in der Version 1.0 herausgebracht. Zurzeit ist die Version 2.0 aktuell.

Das SAX stellt einen Mechanismus zum seriellen Verarbeiten von XML-Dokumenten zur Verfügung. Das heißt, dass ein Dokument Elementweise (Element nach Element) abgearbeitet wird. Die Kommunikation mit anderen Anwendungen wird dabei über sogenannte Handler (siehe Abschnitt 2.6.1) abgewickelt, weshalb man auch von einem ereignisgesteuerten Protokoll spricht.

Folgende Grafik illustriert den Aufbau des SAX:

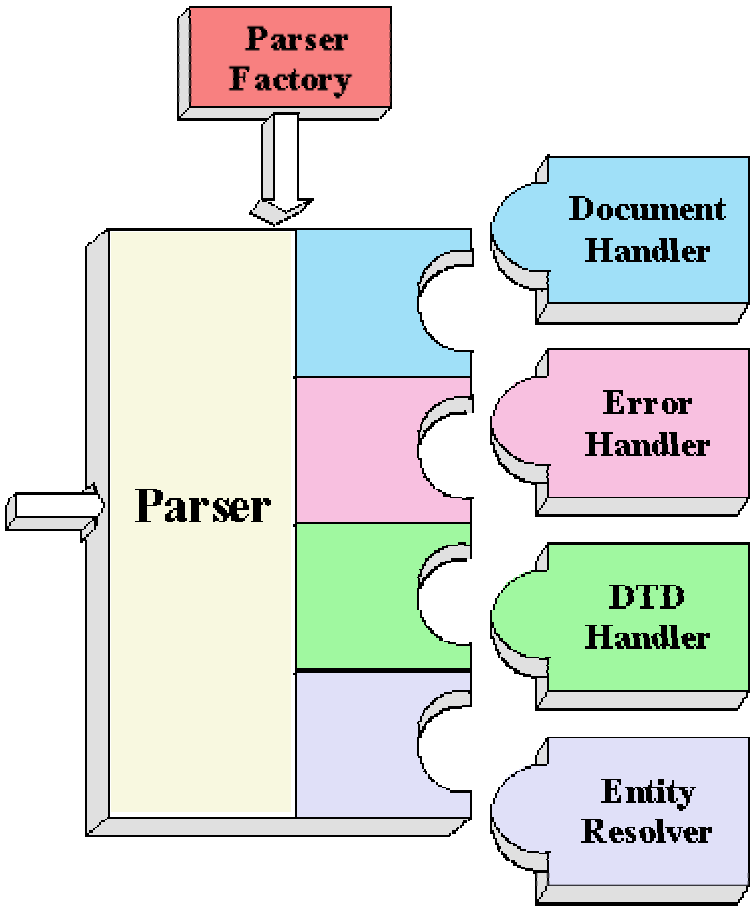


Abbildung 2-3: SAX-API Version 1.0

Um ein XML-Dokument zu parsen, erzeugt man sich zunächst eine Instanz eines SAX-Parsers. Dies geschieht über die *ParserFactory*, die gemäß dem Entwurfsmuster der Fabrikmethode den Erzeugungsprozess des Parsers kapselt. Der Parser darf aus einem beliebigen XML-Parser-Paket kommen und muss sich an die SAX-Spezifikation halten.

Danach muss man die jeweiligen Handler (entweder generische von einem XML-Parser oder jene, die man selbst gemäß den durch SAX gegebenen Schnittstellen implementiert hat) erzeugen und beim Parser registrieren.

Zum Schluss ruft man die Methode `parse()` des Parsers mit dem zu interpretierenden XML-Dokument als Parameter auf. Der Parser wird nun das Dokument sequentiell lesen. Je nach gelesenem Element werden dann bestimmte *callback*-Methoden in den Handlern aufgerufen. Über diese Methoden und vor allem über den *DocumentHandler* lässt sich die Verarbeitung des XML-Dokuments steuern.

2.6.1 SAX Handler

Das SAX-API definiert folgende Handler zur Verarbeitung eines XML-Dokuments:

- **DocumentHandler**: Dies ist der wichtigste Handler zur Verarbeitung überhaupt. In ihm werden die Methoden `startDocument()` und `endDocument()` implementiert, die jeweils zu Beginn und zum Ende eines Dokuments auftreten. Des Weiteren gibt es die Methoden `startElement()` und `endElement()`, die aufgerufen werden, wenn der Parser ein einleitendes Tag bzw. ein schließendes Tag erkennt. Erkennt der Parser z.B. ein Start-Tag, wird die Methode `startElement()` mit dem Namen des Tags und sämtlichen Attributwertpaaren als Parameter aufgerufen. In dieser Methode kann der Programmierer nun die Verarbeitung des Tags steuern, z.B. Name und Attribute des Tags ausgeben. Analog dazu wird die Methode `endElement()` aufgerufen, wenn der Parser ein Ende-Tag erkennt. Erkennt der Parser hingegen freien Text, der zwischen Elementen steht, wird die Methode `characters()` aufgerufen, findet er eine Prozessanweisung, ruft er die Methode `processingInstruction()` auf. Außerdem gibt es noch die Methode `setDocumentLocator()`, mit der ein Locator-Objekt an den *DocumentHandler* übergeben wird. Dieser Locator gibt an, wo der Parser sich gerade im XML-Dokument befindet. Zur Behandlung von whitespace[3] gibt es die Methode `ignorableWhitespace()`.
- **ErrorHandler**: Dieser Handler behandelt alle Fehler, die beim Parsen auftreten. Dafür gibt es die Methoden `warning()`, `error()` und `fatalError()`, die je nach Schwere des Fehlers beim Parsen aufgerufen werden. Als Parameter erhalten diese Methoden eine `SAXParseException`, in der nähere Angaben zur Art des Fehlers und der Position, an der er aufgetreten ist, gespeichert sind. Das SAX liefert zwar standardmäßig einen ErrorHandler in der `HandlerBase` Klasse, dieser erzeugt aber nur Fehlermeldungen bei sogenannten *fatal Errors*, andere Fehler ignoriert er. Um Fehler in XML-Dokumenten zu erkennen, sollte man hier also auf jeden Fall seinen eigenen ErrorHandler mit speziellen Fehlerbehandlungen schreiben und beim Parser registrieren.
- **DTDHandler**: Dieser Handler kommt zur Anwendung, wenn der Parser in einer DTD auf ein unparsed entity, also Binärdaten mit einer zugehörigen Notation, trifft. Er ruft dann entweder die Methode `unparsedEntityDecl()` oder `notationDecl()` auf, in welcher der Programmierer dann seine eigene Behandlung der unparsed entities durchführen kann.
- **EntityResolver**: Der *EntityResolver* wird benötigt, wenn der Parser auf Referenzen zu externen Dateien, z.B. DTDs trifft, und diese lesen muss. Der Parser ruft dann die einzige Methode dieses Handlers, `resolveEntity()` auf, die eine öffentliche Identifikation (URN=Universal Resource Name) in eine Systemidentifikation (URL=Universal Resource Locator) wandelt.

[3] Whitespaces: Nicht sichtbare Zeichen. Dazu zählen Tabulatoren, Leerzeichen und Zeilenumbrüche.

Kapitel 3

SCALABLE VECTOR GRAPHICS (SVG)

In Abschnitt 2.4 wird darauf hingewiesen, dass SVG eine XML-Anwendung ist. Scalable Vector Graphics -SVG- ist ein neuer XML-basierter Grafikstandard von W3C, welcher das Erstellen von kleineren und somit schnelleren und vor allem interaktiven Webdokumenten ermöglicht [4]. Obwohl es ursprünglich für den Interneteinsatz geschaffen wurde, spielt SVG nun auch in Java- oder C++-Anwendungen eine immer größere Rolle.

Im Gegensatz zu den Bitmap-Formaten GIF, JPEG oder PNG sind SVGs auflösungs- und geräteunabhängig. Sie können auf die gewünschte Größe beliebig skaliert werden. Somit spielt es keine Rolle, mit welcher Bildschirmauflösung der Betrachter das Bild anschaut. SVGs lassen sich absolut verlustfrei vergrößern. Weiterhin brauchen sie weniger Speicherplatz als Bitmaps und können somit schneller übertragen werden (vgl. Abschnitt 3.3).

3.1 Grafiksysteme

Die wichtigsten Grafiksysteme auf Computern sind Raster- und Vektorgrafiken. Nachfolgend werden kurz diese beiden Typen erklärt.

3.1.1 Rastergrafik (Pixelgrafik)

Pixel ist ein zusammengesetztes Wort aus dem englischen „Picture“ und „Element“. In einer Pixelgrafik wird jeder einzelne Bildpunkt beschrieben und kann in seiner Farbe verändert werden. Alle Bildpunkte zusammen ergeben durch den Kontrastunterschied zueinander den Bildinhalt (vgl.

Abbildung 3-1).

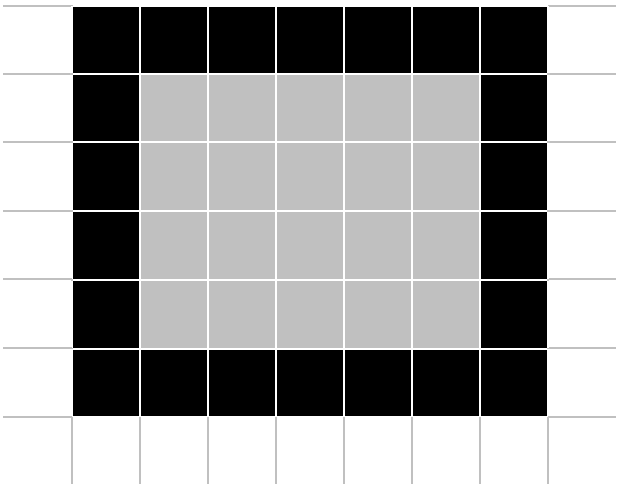


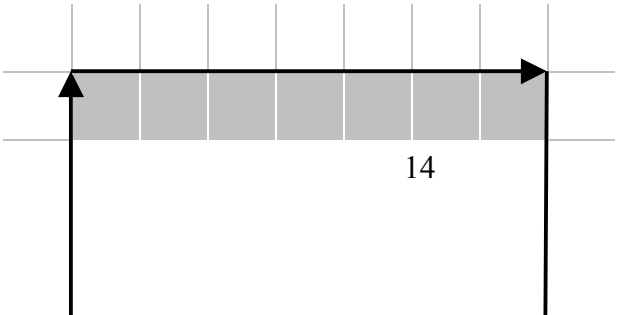
Abbildung 3-1: Rechteck dargestellt als Rastergrafik

Eine Pixelgrafik, auch als Bitmap bezeichnet, wird oft in einem komprimierten Format gespeichert (GIF, PNG, TIFF, usw.). Es gibt viele Möglichkeiten um Bitmaps zu komprimieren. Komprimierungsmethoden sind standardisiert und Grafiken und Tools sind weit verbreitet. Dies ist der Grund, warum das Internet bis zur Einführung von SVG nur Rastergrafikformate unterstützt hat.

Obwohl die meisten Bildschirmgrafiken heutzutage Rastergrafiken sind, benötigt man ein Bildbearbeitungsprogramm um solche Grafiken zu bearbeiten. Programme zur Erstellung von Rastergrafiken gibt es zuhauf und im Allgemeinen sind sie leichter in der Handhabung als vektorbasierte Grafikprogramme, denn im Gegensatz zu Vektorgrafiken ist ein Bitmap ein Bild ohne Objekte.

3.1.2 Vektorgrafik

Bei einer Vektorgrafik besteht das Bild aus einer Reihe von Objekten (Figuren), welche nebeneinander oder übereinander gelagert sind. Diese Objekte werden mit Vektoren oder Kurven beschrieben (vgl. Abbildung 3-2). Zusätzlich wird für jedes Objekt und seinen Umriss die Position, die Form, die Größe, die Füllung und Farbe usw. angegeben, wobei auch mehrere Objekte in einer Gruppe zusammengefasst werden können.



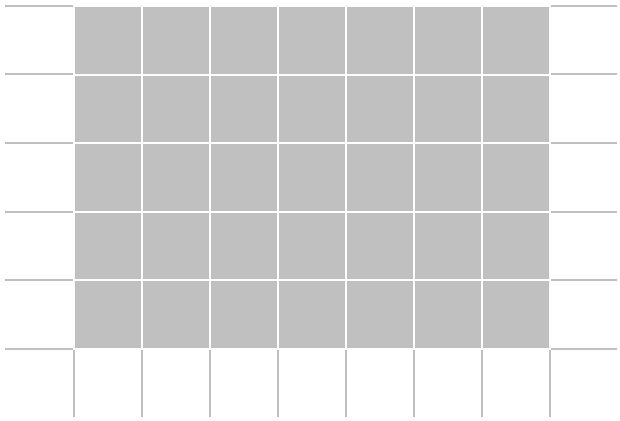


Abbildung 3-2: Rechteck dargestellt als Vektorgrafik

Ein Vektorgrafikprogramm benötigt Kommandos, um Figuren oder Text an eine bestimmte Position zu setzen. Jedes Objekt innerhalb einer Vektorgrafik gehört zu einem bestimmten Typ (Kreis, Ellipse, Rechteck, Text, usw.). Elemente vom Typ Text werden dann auch als Text erkannt. Somit kann nach Textstellen gesucht werden, egal, wie der Text formatiert ist, oder in welcher Richtung er ausgerichtet (oder gedreht) ist. Vektorgrafiken werden hauptsächlich in folgenden Bereichen genutzt:

- CAD-Programme, wo exakte Abmessungen und Vergrößerungsmöglichkeiten zur Detailbetrachtung eine Rolle spielen.
- Grafikprogramme, welche für den Gebrauch von hochauflösenden Druckern angewandt werden (z.B.: Adobe Illustartor).
- Überall, wo Adobe PostScript zum Einsatz kommt.
- Vektorbasierte Programme für den Entwurf von Animationen und Präsentationen auch für das Internet (z.B.: Macromeadia Flash).

3.1.3 Direkter und indirekter Zeichenmodus

Bei der Erzeugung von Grafiken am Computer kann zwischen dem direkten und indirekten Modus unterschieden werden. Grafiksysteme, welche Bilder im direkten Modus erzeugen (z.B. MS Paint), können zwar vordefinierte Formen wie Rechtecke, Kreise oder Polygone in die Abbildung einfügen, jedoch kann der Benutzer nicht im Nachhinein eine schon gezeichnete Form auswählen und verändern. Das Programm fügt die Formen direkt in die Grafik ein, ohne sie separat zu speichern (vgl. Abschnitt 3.1.1). Manipulationen können auf Pixelebene vorgenommen werden. Eine Pixelmenge, häufig ein rechteckiger Bildausschnitt, kann ausgewählt und beispielsweise in Farbe und Größe verändert werden.

Man spricht vom indirekten Modus, wenn der Benutzer jederzeit die zuvor platzierten Figuren auswählen und editieren kann. Je nach Anwendung heißen die vordefinierten Figuren auch Symbole, Schablonen oder Objekte. Die Objekte werden vom Grafiksystem mit den zugehörigen Attributen, wie Größe und Position, gespeichert. Dies ermöglicht die spätere Bearbeitung der Objekte. Wie in Abschnitt 3.1.2 erwähnt, werden die Grafikobjekte in der Regel aus Linienzügen (vgl.

Abbildung 3-2) zusammengesetzt, die im Grafiksystem als Vektoren abgespeichert werden. Wird ein Objekt vom Nutzer verändert, erzeugt das Zeichenprogramm automatisch eine aktualisierte Grafik.

Die Nutzung von Vektorgrafiken in Animationen hat ebenfalls sehr große Vorteile gegenüber der Rastergrafik. Da die Attribute eines Grafikobjektes bei der Vektorgrafik mit abgespeichert werden, können auch alle Attribute animiert werden. Der Anwender ist nicht mehr auf die Animation von Pixelposition und -farbe beschränkt, sondern er kann ganze Objekte in Größe, Position, Farbe und Form animieren. Bei der Erstellung von Simulationen können neben den oben aufgeführten geometrischen Attributen noch Eigenschaften wie Gewicht, Temperatur, etc. den Objekten zugewiesen werden. So können auch komplexe Vorgänge durch die zeitliche Veränderung dieser Eigenschaften simuliert und als Animation dargestellt werden.

3.2 Struktur eines SVG-Dokuments

Der Aufbau einer SVG ist vom Ansatz immer gleich. Da SVG von XML abgeleitet ist, beginnt eine SVG immer mit der Standard XML Anweisung. In der nächsten Zeile kommt immer der DOCTYPE und dann das Root Element mit den gewünschten Attributen, wie aus dem folgenden Beispiel zu entnehmen ist. Der Title-Tag gibt an, was in der Titelleiste eines Bildbetrachtungsprogramms angezeigt wird. Im Desc-Tag folgt eine komplette Beschreibung der Grafik. Die beiden letztgenannten Elemente sind optional und daher nicht notwendiger Bestandteil einer SVG.

Beispiel:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC ”-//W3C//DTD SVG 1.0//EN”
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="140" height="170">
  <title>Katze</title>
  <desc>Einfaches Bild einer Katze</desc>

  <!-- Alle Objekte der SVG -->

</svg>
```

3.3 Vorteile von SVGs

Ein großer Vorteil von Vektorgrafiken ist die Skalierbarkeit ohne Qualitätsverlust, was anhand des folgenden Beispiels verdeutlicht wird.

Die Abbildung 3-3 zeigt eine Katze als Rastergrafik erstellt. Die gleiche Figur wird in Abbildung 3-4 als Vektorgrafik dargestellt (beide Figuren sind Bildschirmkopien eines Monitors mit einer Auflösung von 72 dpi[1]).

Abbildung 3-3: Rastergrafik einer Katze

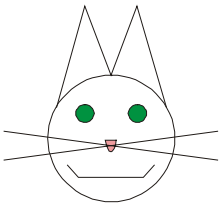


Abbildung 3-4: Vektorgrafik einer Katze

Wird die Rastergrafik vergrößert, ist die Aufgabe des Programms, einen Weg zu finden, die Pixel zu vergrößern. Die einfachste Möglichkeit, ein brauchbares Ergebnis bei einer Vergrößerung um den Faktor 2 zu erzielen, wäre jeden Pixel doppelt so groß zu machen. Das Ergebnis ist, wie Abbildung 3-5 zeigt, nicht wirklich befriedigend. Eine Alternative für die Vergrößerung eines Bitmap wäre die Kantendetektion oder Interpolation. Dies würde ein brauchbareres Ergebnis liefern, jedoch sind diese Verfahren mit zusätzlichem Rechenaufwand verbunden. Außerdem gibt es keine Garantie, dass ein Erkennungsalgorithmus alle Kanten einer Figur korrekt bestimmt.

Die Vergrößerung einer Vektorgrafik um den Faktor 2 bedarf wenig Aufwand. Das Darstellungsprogramm multipliziert alle Vektoren einer Figur mit dem Vergrößerungsfaktor und stellt die neu berechnete Grafik in der vollen Auflösung dar. In Abbildung 3-6 ist das Ergebnis einer zweifachen Vergrößerung zu sehen.

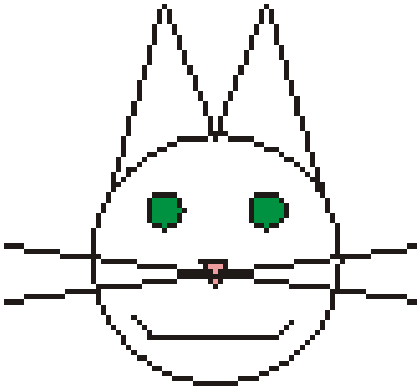


Abbildung 3-5: 200 % Zoom der Rastergrafik

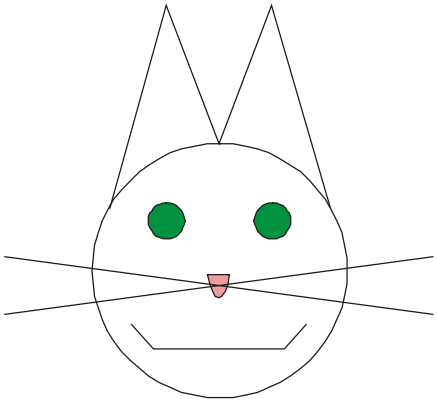


Abbildung 3-6: 200 % Zoom der Vektorgrafik

Ein weiterer Vorteil von SVG ist die geringe Dateigröße. Egal wie groß etwas dargestellt wird, die Informationsmenge, die man zur Beschreibung benötigt, bleibt immer dieselbe. Dies erlaubt hoch-

[1] dpi: Dots per Inch. Maß für die Auflösung von Bildschirmen und Druckern. Anzahl der Bildpunkte pro Zoll (~ 2,54 cm)

komplexe Darstellungen mit geringen Dateigrößen, die sich natürlich beliebig zoomen lassen. Während ein Rasterbild jeden einzelnen Bildpunkt speichert, ist die Vektorgrafik ein reines Textformat. Abbildung 3-5 entspricht 206 * 189 Pixel * 24 Bit / Pixel = 934.416 Bit ≈ 114 KB, während Abbildung 3-6 in reinem Textformat einer Dateigröße von 1,3 KB entspricht. Natürlich lässt sich das Bitmap noch durch verschiedene Algorithmen komprimieren, jedoch wird niemals die geringe Dateigröße einer SVG erzielt. Weiterhin sind die meisten Methoden der Komprimierung nicht ohne Informations- und somit auch nicht ohne Qualitätsverluste.

Der SVG-Standard hat zwei weitere bedeutende Vorteile: SVGs sind textbasiert und somit editier- und durchsuchbar, ein zentrales Kriterium fürs semantische Web. Außerdem basieren SVGs auf XML, was wiederum zur Kombination mit anderen XML-basierten Technologien sowie die Interaktivität mit DOM/Javascript ermöglicht.

Kapitel 4

JAVABEANS

JavaBeans sind ein von der Firma Sun entwickeltes auf Java basierendes Komponentenmodell zur Softwareentwicklung, d.h. eine Architektur für die Definition und Wiederverwendung von Softwarekomponenten. Einzelne Beans dienen als Basisbausteine aus denen man durch Zusammensetzen größere Softwareteile erstellen kann.

Damit so ein Zusammenfügen funktioniert, müssen Beans bestimmte Schnittstellenanforderungen erfüllen und bestimmte Namenskonventionen einhalten. Beans sind insbesondere für die Verwendung in graphischen Entwicklungsumgebungen konzipiert. Das bringt weitere Aspekte mit sich: Visuelle Darstellung von einzelnen Beans, Erledigen mehrerer Aufgaben zum selben Zeitpunkt (Threads), Benachrichtigung und Reagieren auf Ereignisse (Events).

Um zu verstehen, was Java-Komponenten bzw. JavaBeans sind und welche Technologie dahinter steckt, ist ein kurzer Ausflug in die Geschichte der Komponentenentwicklung notwendig.

4.1 Geschichte der Komponententechnik

Nach dem Aufkommen der objektorientierten Programmierung, fragte sich so mancher Experte, ob darin die endgültige Lösung für die Softwareentwicklung der 90er Jahre liegen könnte. Kaum hat sich dieses Paradigma auf breiter Front etabliert, erscheint ein neuer Stern am Himmel: die "Componentware". Kleine Module (Komponenten) sollen zu komplexen Programmen „zusammengeklickt“ werden und lästige „Handarbeit“ vermindern helfen. Damit solche Komponenten genutzt werden können, muss ein einheitlicher Unterbau, eine taugliche Komponentenarchitektur, vorhanden sein. Die Entwicklung von Komponententechnologie(n) begann mit dem Aufkommen grafischer Benutzeroberflächen und das JavaBeans-Modell definiert sich sogar daraus:

„A JavaBean is a reusable software component that can be manipulated visually in a builder tool."

Diese Aussage, hier als Zitat aus der JavaBeans-Spezifikation [11], ist allgemeingültig für alle Komponentenmodelle. Bekannte Technologien sind OpenDoc (propagiert und entwickelt von Apple und

IBM), CORBA[1] (von einem Konsortium einiger hundert Firmen) und DCOM/ActiveX (Microsoft). Die verschiedenen Technologien sind nicht vergleichbar da sie teilweise unterschiedliche Probleme lösen. Marktrelevanz der oben genannten Technologien haben nur noch die Vorschläge von Microsoft und CORBA. Die Entwicklung von OpenDoc wurde 1997 eingestellt, nicht zuletzt ein Tribut an die neue Technologie der JavaBeans, welche nun auch von IBM und Apple propagiert wird. CORBA wird es trotz seiner allgemeinen Unterstützung und Verbreitung weiterhin schwer gegen ActiveX haben, was an der absoluten Marktdominanz der Microsoft Betriebssysteme liegt, welche alle ActiveX implementieren und anbieten.

Die meisten Technologien starteten mit proprietären Lösungen, bei denen einfach alle verwendeten Steuerelemente des Wirtssystems[2] in eigene Elemente eingesetzt wurden. Bekannte Beispiele sind hier Fensterrahmen, Fensterlaufleisten oder auch komplexere Objekte wie ein komplettes Dialogfenster (Ja/Nein/Abbrechen). Erster Vertreter dieser "neuen" Compiler war Visual Basic (VB). Sowieso stellte sich VB als Schrittmacher der Komponentenprogrammierung bei Microsofts Windows-Oberfläche heraus. Die ersten Komponenten waren VBX-Komponenten (Visual Basic eXtension). Anfangs noch stark an die Bedürfnisse von Basic angepasst, waren diese für die meisten höheren Programmiersprachen nur sehr schwer ansprechbar. VBX wurde ein großer Erfolg, nachdem Microsoft eine einheitliche Schnittstelle mit einer festgelegten Struktur für Komponenten vorschrieb. Dies ermöglichte "3rd-party-development" - die Herstellung und Programmierung von Komponenten durch Drittanbieter - mit der Garantie, dass die vom Programmierer gekauften Komponenten mit dem Compiler und dem Betriebssystem vollkommen kooperieren. (Den gleichen Ansatz verfolgt auch Sun mit dem JavaBeans-Modell.) Aufgrund des Erfolgs wurde ein Nachfolgemodell entwickelt, dass auch die oben erwähnten Mängel (z. B. waren alle Komponenten noch 16-Bit) beseitigen sollte: OCX-Komponenten (OLE[3] Control EXtension), welche z. B. 32-Bit-Technologie einführten. Beide Varianten konnten bereits über die definierten OLE/COM-Schnittstellen auf einem System miteinander Module tauschen und/oder kommunizieren. Trotz DCOM[4] (Distributed COM) konnte von „echter“ Verteilung - sprich: von Verteilung und Kommunikation von Modulen über Rechner- und Betriebssystemgrenzen hinweg - keine Rede sein.

Dies war der technische Stand (bei Microsoft) etwa Mitte 1996. Im gleichen Jahr sollte eine „neue“ Programmiersprache – Java - die Entwicklerszene revolutionieren. Die Sprache fiel auf durch kleine, effektive Programme eingebettet in Internet WWW-Seiten. Offenbar war es möglich geworden mit

---

[1] Spezifikation für die Kommunikation von Objekten unterschiedlicher Programme.

[2] Wirtssystem bezeichnet das real gebootete Betriebssystem.

[3] Eine Technologie zum Austausch und zur gemeinsamen Nutzung von Daten zwischen verschiedenen Anwendungen.

[4] Komponentenspezifikation für verteilte Systeme.

vergleichsweise winzigen Programmen in einem weltweiten, mehr oder weniger chaotisch verwalteten Netzwerk, mit Millionen von Benutzern, eine stabile und sichere Methode zur Verfügung zu stellen, Programme vom Server- auf einen Clientrechner zu übertragen und dort auszuführen. Java erlebte einen gewaltigen Boom. Alle großen Softwarefirmen lizenzierten Java, tausende von Freaks entwarfen immer neue Applets für ihre Internetseiten: Vom Börsenticker bis zum Kreuzworträtsel. Java öffnete offenbar ein neues Gebiet im Bereich des „distributed computing“. Möglich wurde dieser Erfolg durch ein neues durchdachtes System, das alle Anforderungen an eine moderne Programmiersprache erfüllte. Tatsächlich vereinigt Java „nur“ altbekannte Konzepte anderer Programmiersprachen. Java bringt die Industrie dem Ziel eines verteilten, plattformunabhängigen Betriebssystems oder Programms ein gutes Stück näher. Durch die Einhaltung von bekannten, internationalen Standards, der kompletten Offenlegung aller Quelltexte, Schnittstellen usw. und nicht zuletzt der kostenlosen Verfügbarkeit ist Java ein voller Erfolg.

Zu diesem Zeitpunkt hatte Microsoft der neuen Technik nichts entgegenzusetzen. Das OCX/DCOM-Modell scheiterte schlicht an der Größe der Komponenten, die bis in den Megabytebereich vordringen können. Eine (weltweite) Übertragung solcher Files über das vergleichsweise langsame Internet ist inakzeptabel. Microsoft veröffentlichte gezwungenermaßen ein neues Objektmodell mit dem Namen ActiveX. Durch die volle Abwärtskompatibilität dieses Modells standen den Entwicklern sofort tausende „alter“ Komponenten zur Verfügung. Natürlich blieb es bei der langen Entwicklung nicht aus, dass eine Menge von Rückwärtskompatibilitäten, „work-arounds“ usw. in dieser Technologie haften geblieben ist, die die Programmierung von ActiveX-Controls[5] spannend machen können. Desweiteren glaubte man bei Microsoft wohl, dass sich die Erfahrungen mit lokalen Systemen ohne weiters auf die Internet-Programmierung übertragen lassen. Schutzrechte sind ActiveX unbekannt und warum sollte es mich stören, dass ein beliebiger Programmierer oder Komponentennutzer alles mit der lokalen Festplatte machen kann? Auch das Senden von „I`m-still-alive“-Messages von ActiveX-Objekten (auch eine „alte Erbschaft“), welches nach wie vor notwendig ist um die Objektqueue aktuell zu halten, unterstreicht nicht gerade die Netztauglichkeit von ActiveX, denn im Internet gehen nach wie vor TCP/IP-Pakete verloren.

4.2 Einordnung des JavaBean-Modells

Etwa zeitgleich zu ActiveX veröffentlichte Sun erste „Papers“ zu Javas Komponentenmodell - den JavaBeans. Die Technologie basiert vollständig auf der Programmiersprache Java und ist somit plattformunabhängig. Weiterhin „erbt“ jede JavaBean den hohen Sicherheitsstandard Javas in Bezug auf verteilte Anwendungen und Inter-/Intranetkommunikation. Speziell die Plattformunabhängigkeit er-

[5] Grafische Oberflächenelemente wie z.B. Buttons, Textfelder, Listen usw. Werden auf Deutsch ActiveX-Steuerelemente genannt.

öffnet aber zunächst eine neue Problematik. Ein Windows-Dialog hat ein anderes Aussehen als z. B. ein Motif-Dialog. Dialoge verschicken Nachrichten und Ergebnisse an z.B. ihre aufrufende Instanz - dies kann wiederum auf völlig unterschiedliche Art und Weise geschehen. Sun ist es inzwischen gelungen dieses Problem weitestgehend zu lösen, weshalb der Programmierer auch von dieser Problematik befreit ist. Der Hauptunterschied zu den in Abschnitt 4.1 vorgestellten Technologien ist wohl, dass es sich bei Beans-Technologie nicht um konkreten „neuen“ Code handelt. Alle zur Programmierung von JavaBeans notwendigen Elemente, sind bereits im Java Development Kit[6] (JDK) enthalten. Das Konzept integriert sich voll und ganz in die Programmiersprache Java.

Das Modell besteht vielmehr aus bindenden Vereinbarungen, die jeder Programmierer einhalten muss, um eine Bean zu programmieren und auch veröffentlichen zu können - ganz ähnlich wie es bei Visual Basic praktiziert wurde. Diese Vereinbarungen sind insoweit verpflichtend, als das Sun ein Softwarepaket zum schnellen entwickeln, testen und veröffentlichen von Beans anbietet, welches genau diese Vereinbarungen benötigt und überprüft. Von Seiten der Entwickler stellt diese Vereinbarung sowieso keinen Nachteil dar, denn eine Komponente, die nur auf einem Rechnersystem und/oder nur einer Konfiguration läuft, widerspricht dem Sinn einer Komponente - es ist einfach keine. Die Vereinbarung betrifft z.B. konkrete Methodennamen der JavaBean.

Prinzipiell lässt sich aus jedem Java-Applet und auch jeder Java-Applikation eine JavaBean machen. Eine JavaBean ist viel flexibler „einstellbar“ als ein Applet oder eine Applikation. Dem Programmierer muss dazu weder Quellcode noch eine exakte Schnittstellenbeschreibung vorliegen. Durch die definierte Schnittstelle (bzw. die Vereinbarung) der Komponente kann eine Java Virtual Machine während der Laufzeit feststellen, welche Fähigkeiten eine JavaBean hat und wie diese genutzt werden können. Bei der Programmierung (oder auch später) können so während der Programmausführung neue Komponenten mit Drag&Drop eingefügt und sofort genutzt werden. Diese neue Leistung ist natürlich nicht nur auf die Spezifikation einiger Schnittstellen zurückzuführen, sondern eben auch konkret auf Eigenschaften der Programmiersprache Java und der Laufzeitumgebung; sie unterstreicht die Leistungsfähigkeit Javas und demonstriert eindrucksvoll die Fähigkeiten einer modernen Programmiersprache - JavaBeans sind bereits "eine Anwendung" von Java!

Zum Schluss dieses Abschnitts soll nicht verschwiegen werden, dass die Entwicklung der JavaBeans-Komponententechnologie noch lange nicht abgeschlossen ist. Um schon jetzt diese Technologie in bestehende Betriebssysteme integrieren zu können wurde beispielsweise von Sun die Java Bridge für ActiveX programmiert. Sie ermöglicht die Benutzung von JavaBeans als ActiveX-Controls.

[6] Von der Firma Sun kostenlos zur Verfügung gestellte Entwicklungsumgebung für Java

4.3 Das JavaBean-Modell

Eine JavaBean zeigt nach außen ihre Eigenschaften (public properties), ihre Methoden (public methods), und ihre Ereignisse (events). Informationen über das Innenleben (intospective) der Bean wird durch die Klasse BeanInfo realisiert.

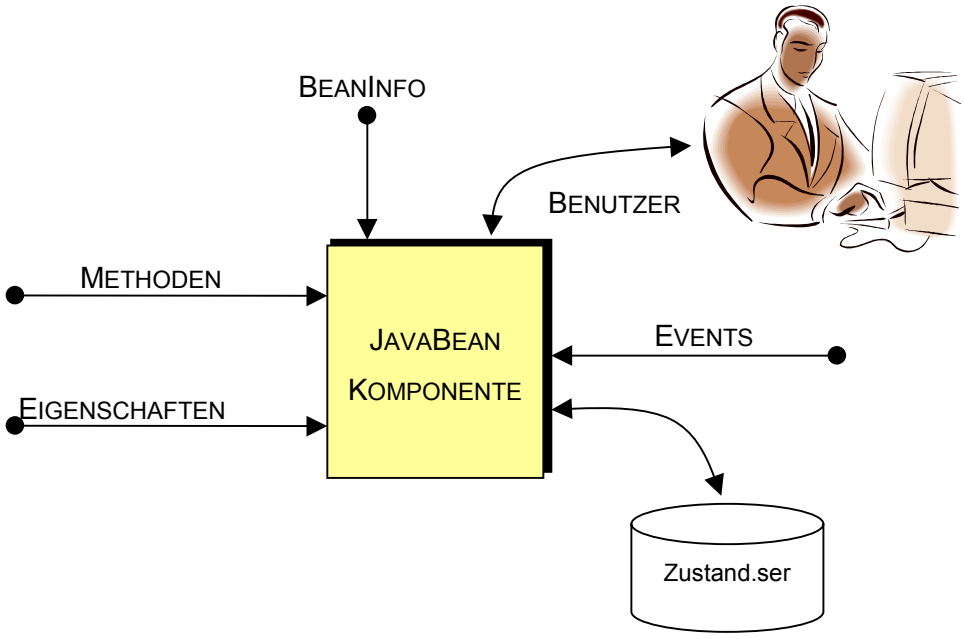


Abbildung 4-1: Blackbox-Ansicht einer JavaBean

Ein sehr großer Vorteil von JavaBeans ist die Persistenz. Dies bedeutet, der Zustand einer Bean kann (z.B. in einer seriellen Datei) gespeichert werden, um nach der Wiederaufnahme des Programms (der Komponente) in den alten, gespeicherten Zustand zu kommen. Nachfolgend werden die Begriffe Events, Properties, Methods und Introspetive erklärt.

4.3.1 Ereignisse (Events)

Events dienen der Benachrichtigung anderer Komponenten beim Eintreten von bestimmten Ereignissen. Aus der Sicht der Bean werden Events als Ausgangsschnittstellen gesehen.

Die Bean stellt eine Schnittstelle (Interface) zur Verfügung, an der sich andere Komponenten registrieren können, um dann beim Auftreten von Ereignissen benachrichtigt zu werden. Es gibt also eine Ereignisquellen-Bean und eine Ereignisverarbeitungs-Bean (Listener). Ein Ereignis selbst wird ebenfalls als Klasse definiert, demnach gibt es auch noch ein Ereignisobjekt.

Ein Ereignis wird im Standardfall von beliebig vielen Listenern verarbeitet (multicast). Wird ein Ereignis jedoch von max. einem Listener verarbeitet, nennt man das „unicast“.

4.3.1.1 Ereignisquellen-Bean

Sie ist verantwortlich für das Generieren spezifischer Ereignisse. Jede Art von Ereignis(sen) wird durch ein Listener-Interface definiert. Dieses Interface gibt Methoden an, die beim Auftreten eines Ereignisses aufgerufen werden.

Beispiel: Listener-Interface

```
public interface ModelChangedListener {
  public void modelChanged( ModelChangedEvenet evt );
  // ModelChangedEvent wird von EventObject abgeleitet
}
```

Damit sich Listener anmelden können, muss die Ereignisquellen-Bean Methoden zum Registrieren zur Verfügung stellen (siehe folgendes Beispiel).

Beispiel: Methoden für das An- und Abmelden von Listener

- `public void addModelChangedListener( ModelChangedListener m )`
- `public void removeModelChangedListener( ModelChangedListener m )`

Die Ereignisquellen-Bean stellt intern (meist private) Methoden zum Auslösen von Ereignissen bereit.

Beispiel: Methode zum Auslösen eines Ereignisses

```
private void fireModelChanged( ModelChangedEvent evt ) {
  //ruft für jeden registrierten Listener die modelChanged()-Methode auf
.....
}
```

4.3.1.2 Ereignisverarbeitungs-Bean

Sie muss das Listener-Interface implementieren und wird dadurch gezwungen, auch die Methoden (im Beispiel `public void modelChanged(…)`) zu implementieren. Damit sie beim Eintritt von Ereignissen benachrichtigt wird, muss sie sich bei der Ereignisquellen-Bean selbst registrieren (durch die Methode `addModelChangedListener(this)`).

4.3.1.3 Das Ereignisobjekt (EventObject)

Die Basisklasse `EventObject` definiert die folgenden Methoden, welche Informationen über Ereignisse enthalten.

- `public Object getSource()`,
  liefert Objekt, welches das Ereignis erzeugt hat (Ereignisquelle).

- `public String toString()`,
  liefert den Ereignisnamen als String.

Spezifische Ereignisklassen werden bei Bedarf von `EventObject` abgeleitet und ggf. mit zusätzlichen Attributen und Methoden erweitert. Diese Klassen werden dann beim Eintritt eines Events von der Ereignisquelle generiert.

4.3.2 Eigenschaften (Properties)

Eine Property ist eine mit einem Namen bezeichnete Eigenschaft einer Bean (im technischen Sinne ein Attribut), welche ihr Aussehen und ihr Verhalten bestimmen kann (z.B. Hintergrundfarbe oder Beschriftung einer Schaltfläche). Das Setzten oder Lesen der Eigenschaften kann im Allgemeinen zur Generierungszeit und zur Laufzeit der Anwendung geschehen. Es gibt folgende unterschiedliche Eigenschaften:

- Einfache (simple property),
- Feld von Eigenschaften (indexed property),
- gebundene (bound property) und
- eingeschränkte (constrained property).

Zu jedem dieser Typen ist ein Codemuster definiert. Anhand dieses Codemusters kann man erkennen, um welchen Typ es sich handelt.

4.3.2.1 Einfache Eigenschaft (simple property)

Die simple property besteht aus genau einem Wert. Folgendes Codemuster ist für diesen Typ definiert:

Attribut:	`<PropertyName>` vom Typ `<PropertyType>`
Lesender Zugriff:	`<PropertyType> get<PropertyName>()`, bei Bool'schem Wert `boolean is<PropertyName>()`
Schreibender Zugriff:	`void set<PropertyName>( <PropertyType> value )`

4.3.2.2 Feld von Eigenschaften (indexed property)

Die Indexed property besteht aus einem Feld von Werten. Folgendes Codemuster ist für diesen Typ definiert:

Attribut:	`<PropertyName>[]` vom Typ `<PropertyType>`
Lesen eines Wertes:	`<PropertyType> get<PropertyName>( int index )`
Setzen eines Wertes:	`void set<PropertyName>( int index, <PropertyType> value )`
Lesen des Feldes:	`<PropertyType>[] get<PropertyName>()`
Setzen des Feldes:	`void set<PropertyName>( <PropertyType> values[] )`

4.3.2.3 Gebundene Eigenschaft (bound property)

Diese Eigenschaften beinhalten eine Benachrichtigung bei Änderungen. Registrierte Objekte und Beans werden benachrichtigt, wenn sich der Wert dieser Eigenschaft ändert. Es wird also ein `PropertyChangedEvent` (von `EventObject` abgeleitet) generiert, wenn sich der Wert ändert.

Gebundene Eigenschaften stellen zusätzliche Registrierungsmethoden (`addPropertyChangedListener( … )`) und interne Event Methoden (`firePropertyChanged( … )`) bereit. Zusätzlich gibt es auch hier für die Listener ein Interface, welches sie implementieren müssen.

Das Codemuster ist identisch mit dem der einfachen Eigenschaft (siehe Kapitel 4.3.2.1) bzw. dem des Feldes von Eigenschaften (siehe Kapitel 4.3.2.2).

4.3.2.4 Eingeschränkte Eigenschaft (constrained property)

Constrained properties sind gebundene Eigenschaften, mit dem Unterschied, dass die Listener ein Veto nach der Änderung einlegen können. Wird also ein Listener über die Änderung eines Wertes benachrichtigt, kann er diese Änderung rückgängig machen. Das Veto wird durch „werfen“ einer `PropertyChangedException` eingelegt.

Es gelten dieselben Bedingungen und das gleiche Codemuster wie bei der gebundenen Eigenschaft (siehe Kapitel 4.3.2.3).

4.3.3 Methoden

JavaBean Methoden unterscheiden sich nicht von Methoden anderer Java-Klassen. Die Methoden sind Eingangsschnittstellen und können somit von anderen Beans aufgerufen bzw. benutzt werden.

4.3.4 Introspektion

Introspektion bedeutet die Ermittlung der Schnittstellen. Es sollen also Informationen eingezogen werden, welche Methoden, Eigenschaften und Ereignisse eine Bean anbietet. Die Introspektion dient hauptsächlich visuellen Generierungswerkzeugen (Visual Builder Tool – VBT, z.B. Borland JBuilder). Mit Hilfe solcher Werkzeugen und der Information kann der Entwickler zur Generierungszeit der Anwendung die Bean auf Kundenwünsche zuschneiden und/oder die Bean mit anderen Anwendungen verknüpfen.

Für die Durchführung der Introspektion gibt es die folgenden zwei Alternativen:

- **Automatische Introspektion**: Bei Befolgung der Codemuster und mit Hilfe von Java Reflection Features (Metainformation der Klasse) kann eine Entwicklungsumgebung wie Borland JBuilder die Introspektion automatisch durchführen. Es muss dazu also keine

eigene Klasse erstellt werden. Das vereinfacht die Erstellung einer Bean und verringert somit den Aufwand für den Entwickler.

- **Erstellung einer Infoklasse (d.h. Spezifikation) zur Bean**: Der Anwender erstellt eine eigene Informationsklasse. Dieses Klasse implementiert das Interface `BeanInfo` Der Klassenname muss `<BeanName>BeanInfo` lauten.

Entwicklungsumgebungen versuchen bei der Analyse einer Bean zuerst die BeanInfo-Klasse zu finden, ist diese nicht vorhanden, wird automatisch die Introspektion durchgeführt. Die bei beiden Methoden benutzte Inspektor Klasse liefert als Ergebnis ein BeanInfo Objekt, welches die Beschreibung der Schnittstellen enthält.

4.3.5 Persistenz

Persistenz erlaubt die exakte Wiederherstellung einer Komponente, die in einem bestimmten Zustand gesichert wurde. Die Speicherung des Zustandes der Bean ist erforderlich, um ein Programm nach der Kundenanpassung oder nach der Beendigung und Wiederaufnahme mit demselben Aussehen und Verhalten ausführen zu können. Typischerweise besteht der zu speichernde Zustand aus allen Eigenschaften und eventuell zusätzlicher private Attribute, die nicht von den Eigenschaften abgeleitet werden können.

Beispiel: Zu speichernde Eigenschaften, um benutzerdefiniertes Aussehen bewahren zu können.

Eigenentwickelte Schaltfläche mit den Eigenschaften

- Vordergrundfarbe
- Hintergrundfarbe
- Beschriftung

Damit der gespeicherte Zustand rekonstruiert werden kann, muss die persistente Bean mit der statischen Methode `instantiate( … )` der Klasse `Beans` geladen werden.

Beispiel: Benutzung der Methode `instantiate( … )` der Klasse `Beans`

```
..
// Typzuweisung myDbBrowser ist Variable des Objektes DbBrowser
DbBrowser myDbBrowser
..
// gespeicherter Zustand der Bean mit dem Namen "DavesBrowser" wird
// rekonstruiert
myDbBrowser = (DbBrowser)Beans.instantiate( theClassLoader,"DavesBrowser" )
..
```

Bei der persistenten Speicherung von Beans gibt es zwei Alternativen.

(1) **Serialisierung**: Dies ist die Standardalternative. Der Objektzustand wird in eine Bytefolge umgewandelt und in einer Datei mit der Endung „.ser“ gespeichert. Die Serialisierung ist sehr einfach zu verwirklichen. Die Bean-Klasse implementiert das Interface `Serializable` und zeigt damit schon die Persistenz an. Es werden automatisch alle einfachen Attribute und alle eingebetteten serialisierbaren Objekte gespeichert. Attribute vom Typ `transient` und vom Typ `static` werden nicht gespeichert. Wenn dies erforderlich ist, muss man die Serialisierung selbst implementieren. Dies nennt man Externalisierung.

(2) **Externalisierung**: Diese Alternative ähnelt der Serialisierung, jedoch muss man die Methoden zur Speicherung selbst implementieren. Dies ist zwar mit einem größeren Programmieraufwand verbunden, jedoch kann man bei der Speicherung ein eigenes Datenformat benutzen, welches beispielsweise zu anderen Tools kompatibel ist.

4.3.6 Beans speichern: Java-Archive

Java-Archive (erkennbar an der Dateiendung `.jar`) sind mit dem JDK Version 1.1 eingeführt worden. Auf Binärebene entsprechen sie den allseits bekannten ZIP-Archiven[7]. Das JAR-Packformat wurde eingeführt um dem Anwender eine leichte Möglichkeit zu geben, JavaBeans zu verpacken. Viele Beans enthalten neben den notwendigen Klassen auch Grafiken usw. Es ist äußerst lästig eine evtl. größere Menge von Dateien einzeln zu laden und zu starten. Das JAR-Format ermöglicht das Laden aller Klassen und notwendigen Grafiken in einer Datei.

Das Java Developer’s Kit enthält ein Werkzeug um Java-Archive erstellen zu können.

4.3.7 Visuelle und nicht-visuelle Komponenten

Sun definiert Komponenten derart, dass sie sich visuell durch so genannte Applikation-Builder manipulieren lassen. Die Bean kann auf dem Bildschirm dargestellt und angepasst werden. Vielen dürfte dies von GUI-Buildern bekannt vorkommen. Und hier gibt es auch eine Verwandtschaft, denn in Java sind auch die AWT-Komponenten[8] nichts anderes als Beans. Nehmen wir etwa eine Schaltfläche: Sie besitzt ein Aussehen (View) und einen internen Zustand (Modell). Die Schaltfläche lässt sich auf einem Arbeitsblatt positionieren und über einen speziellen Dialog des Builders kann der Text oder die Schriftfarbe angepasst werden.

Eine Komponente muss aber nicht zwingend grafisch sein. Genauso gut kann es eine Datenstruktur oder eine FTP-Klasse sein. Eine häufig eingesetzte Komponente ist etwa ein Timer, der in festen Ab-

[7] Bestimmter Algorithmus, um Daten zu komprimieren. Bekanntestes Werkzeug dafür ist WinZip

[8] Standardpaket mit vielen Oberflächenkomponenten für Java.

ständen ein Ereignis meldet. Von IBM wird eine Reihe von Beans angeboten, die Elemente einer Programmiersprache aufweisen, etwa Schleifen und Kontrollstrukturen.

4.3.8 Vor- und Nachteile

Das JavaBean Konzept unterstreicht die Plattformunabhängigkeit von Java. Diese ermöglicht es, dass ein Programm ohne Änderungen auf jeder Rechnerplattform eingesetzt werden kann. JavaBeans weisen Gemeinsamkeiten mit den ActiveX-Steuerelementen von Microsoft auf. Allerdings gibt es wesentliche Unterschiede: JavaBeans können nur in der Programmiersprache Java entwickelt werden, ActiveX-Steuerelemente dagegen mit nahezu beliebigen Sprachen. Auch hinsichtlich der verwendbaren Rechnerplattformen unterscheiden sich beide Architekturen: JavaBeans lassen sich auf beliebigen Rechnerplattformen einsetzen, ActiveX-Steuerelemente dagegen nur unter dem Betriebssystem Windows.

Kapitel 5

VERWENDETE BIBLIOTHEKEN

Zur Implementierung des horizontalen Truppenbaumes in Java habe ich unter anderem die folgenden Klassenbibliotheken benutzt.

5.1 Batik SVG Toolkit

Das Projekt Batik ist ein Hilfswerkzeug für Java, welches kostenlos unter der Internetadresse http://xml.apache.org/batik/dist/archives/ inklusive Dokumentation erhältlich ist. Es dient nicht nur Anwendungen oder Applets[1], welche Grafiken im SVG-Format für unterschiedlichste Zwecke nutzen wollen, sondern bietet auch einige Hilfswerkzeuge und fertige Anwendungen zur Bearbeitung von SVGs an. Für den praktischen Teil dieser Diplomarbeit wurden die Möglichkeiten zur Erstellung und zur Darstellung von SVGs in Java genutzt.

Grundsätzlich gilt für Batik: Alle Module von Batik können einer der drei Kategorien zugeordnet werden.

- **Kernmodule**: Das Herzstück der Batikarchitektur. Diese Module können einzeln oder in Kombination benutzt werden, um individuelle Fälle im Umgang mit SVGs abzudecken.
- **Anwendungsmodule**: Fertige Klassen, welche die Kernmodule benutzen. Auch wenn es bequem und schön erscheinen mag, dass es bereits fertige Klassen zum Umgang mit SVGs gibt, so dienen sie doch eher zur Illustration der Leistungsfähigkeit von Batik.
- **Systemnahe Module**: Diese Module werden von Entwicklern normalerweise nicht direkt benutzt. Sie unterstützen eher die Kernmodule.

Folgende Abbildung zeigt den Zusammenhang der einzelnen Bausteine:

---

[1] Ein Applet ist ein Programm, das innerhalb eines anderen Programms gestartet wird. Meist wird es innerhalb des World Wide Webs benutzt. Es kann auf jedem Java™-fähigen Browser ausgeführt werden.

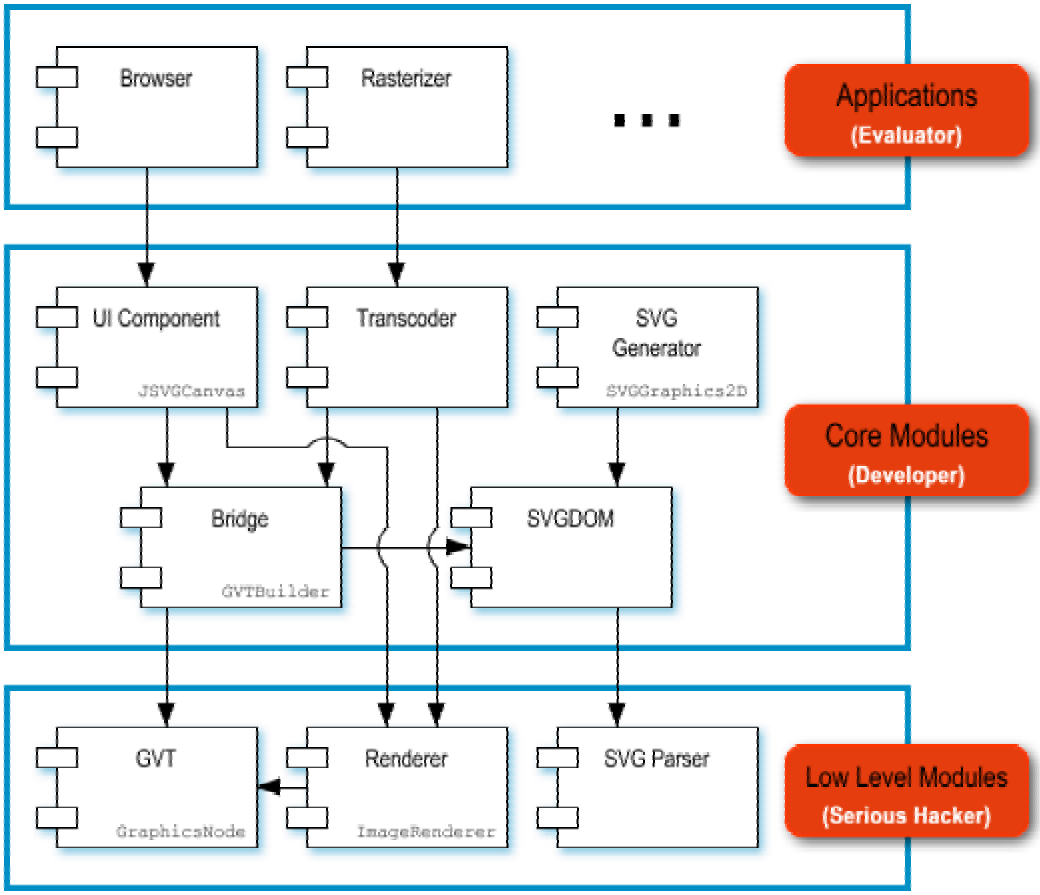


Abbildung 5-1: Zusammensetzung der Module bei Batik

Dabei bietet Batik folgende Anwendungen an:

- **SVG Browser:** Ermöglicht das Ansehen, Rotieren und Zoomen von SVGs. Weiterhin können animierte SVGs dargestellt werden.
- **SVG Rasterizer:** Hilfswerkzeug zum Konvertieren von SVGs in ein Pixelgrafikformat wie JPEG, PNG oder TIFF
- **SVG Pretty Printer:** Formatiert fertige SVGs zur besseren Lesbarkeit (z.B. einrücken der Subbefehle).
- **SVG Font Converter:** Ermöglicht das Konvertieren von TrueType Fonts in SVG Fonts. So wird gewährleistet, dass das Aussehen von Schriften auf jedem System gleich ist.

Da außer zu Testzwecken keine der oben genannten Anwendungen innerhalb dieser Diplomarbeit zum Einsatz kam, werden die Anwendungen nicht weiter untersucht.

5.1.1 Erstellen einer SVG

Eine SVG ist in erster Linie ein Dokument in Textformat, welches bestimmten Regeln unterworfen ist (Regeln sind aus XML abgeleitet). Kennt man diese Regeln, kann man natürlich sehr einfach eine

SVG innerhalb einer Applikation erstellen. Dazu muss man nur eine ASCII-Datei erstellen und die einzelnen Tags der gewünschten Grafik sequenziell in diese Datei schreiben (oder man hält die komplette Grafik in einem String im Hauptspeicher). Allerdings erhält man keinerlei Verifikation dieses Dokumentes. Letztendlich wird man erst beim Versuch, die Grafik darzustellen, sehen ob die Syntax fehlerfrei ist und alle Regeln eingehalten wurden. Um das zu umgehen, bietet das Projekt Batik Klassen und Methoden an, SVG-Dokumente durch Einfügen bzw. Ändern von Tags oder durch Umwandeln von Java-2D-Graphics-Objekten in ein SVG-Dokument zu erstellen. In den folgenden Beispielen wird das Erstellen derselben SVG mit den zwei oben genannten Möglichkeiten erklärt.

Beispiel 1: Erstellen einer SVG durch das Einfügen von Tags.

```
// Hole Instanz von DOMImplmentation
DOMImplementation impl = SVGDOMImplementation.getDOMImplementation();

// Erstelle Instanz des Document-Objektes basierend auf SVG
Document doc = impl.createDocument(null, "svg", null);

// Besorge das root Element (das SVG Element)
Element svgRoot = doc.getDocumentElement();

// setzen der width und height Attribute des svg-root Elementes
svgRoot.setAttributeNS(null, "width", "31");
svgRoot.setAttributeNS(null, "height", "31");

//rectangle Erzeuge ein Rechteck
Element rectangle = doc.createElementNS(svgNS, "rect");
rectangle.setAttributeNS(null, "x", "10");
rectangle.setAttributeNS(null, "y", "10");
rectangle.setAttributeNS(null, "width", "100");
rectangle.setAttributeNS(null, "height", "100");
rectangle.setAttributeNS(null, "style", "stroke:black; fill:red;");


// Fügt das Rechteck zum root Element hinzu
svgRoot.appendChild(rectangle);
```

In Abbildung 5-2 wird das Ergebnis dieser SVG dargestellt.

Beispiel 2: Erstellen einer SVG durch Wandeln der Java-2D-Grafik in eine SVG.

```
// In der Methode Paint wird die Java-2D-Grafik gehalten
public class SVGTest {
  public void paint(Graphics2D g2d) {
    g2d.setPaint(Color.red);
    g2d.fill(new Rectangle(10, 10, 100, 100));
  }
```

```
  public static void main(String [] args) throws IOException {
    // Hole Instanz von DOMImplmentation
    DOMImplementation impl = SVGDOMImplementation.getDOMImplementation();

    // Erstelle Instanz des Document-Objektes basierend auf SVG
    Document doc = impl.createDocument(null, "svg", null);

    // Erstelle eine Instanz des SVG Generators
    SVGGraphics2D svgGenerator = new SVGGraphics2D(doc);

    // Veranlasse SVGTest die Grafik in die SVGGraphics2D Implementierung zu
       rendern
    SVGTest test = new SVGTest();
    test.paint(svgGenerator);

    // Letztlich kann man z.B. die SVG an der Standardausgabeeinheit in UTF-8-
    // Codierung (wandeln von Zeichen in Bytes) ausgeben oder alternativ
    // sequentiell in eine Datei schreiben.

// Angabe, dass CSS style Attribute benutzt werden sollen
    boolean useCSS = true;
    Writer out = new OutputStreamWriter(System.out, "UTF-8");
    svgGenerator.stream(out, useCSS);
  }
}
```

In folgender Abbildung sieht man die Darstellung der SVG innerhalb es Explorers.

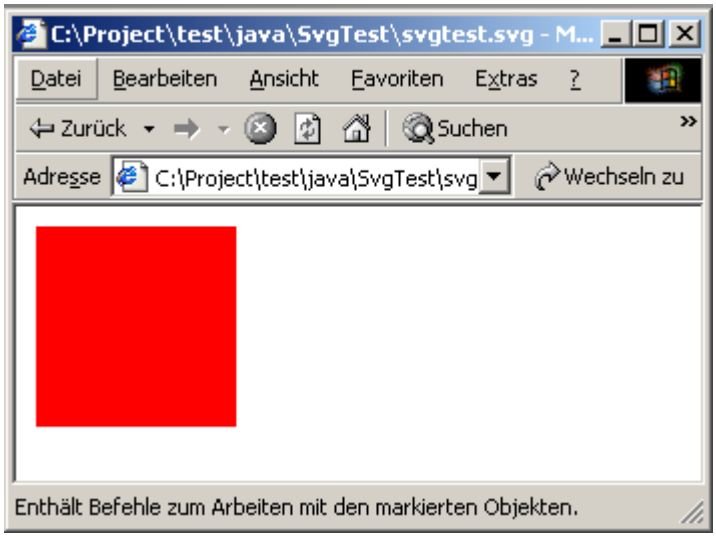


Abbildung 5-2: Ergebnis der SVG aus den beiden oberen Beispielen

Wie man sieht, ist die Erstellung einer SVG sehr einfach. Der dabei erstellte Code für die SVG sieht wie folgt aus:

```
<svg height="110" width="110">
    <rect width="100" x="10" height="100" y="10" style="stroke:none; fill:red;"/>
</svg>
```

Innerhalb von Java reicht dieses Dokument aus, um es in eine Grafik zu rendern und anzuzeigen. Soll jedoch diese Grafik als Datei gespeichert werden, so bietet Batik dazu die Klassen `DOMUtilities` und `SVGGraphics2D` an. Mit ihrer Hilfe wird der Standardkopf einer SVG (siehe Kapitel 3.2) hinzufügt und das Dokument mit einem `Writer`–Objekt[2] ausgeben.

Normalerweise ist erwünscht, dass eine SVG nicht als Text ausgegeben, sondern das Bild dargestellt wird. Dazu benötigt man Programme, welche SVGs interpretieren und in eine Grafik rendern.

5.1.2 Anzeigen einer SVG

Wie bereits erwähnt, kann man Vektorgrafiken mit speziellen Vektorgrafikprogrammen anzeigen bzw. erzeugen (siehe Abschnitt 3.1.2). Da SVG ursprünglich „nur“ für das Internet gedacht war, geht man davon aus, dass SVGs auch problemlos von Standardbrowsern dargestellt werden können. Dem ist zurzeit leider noch nicht so. Weder Microsoft® noch Netscape™ bieten in ihren aktuellen Browserversionen (Explorer 6.0 / Navigator 7.0) eine Interpretation von SVGs an. Will man sich SVGs mit einem Webbrowser anschauen, so bleibt nur die Möglichkeit, sich ein Plugin[3] via Internet herunterzuladen und zu installieren. Jedoch wird es vermutlich nicht mehr lange dauern, bis Standardbrowser solche Plugins für SVGs bereits bei der Auslieferung enthalten.

Batik bietet zum Anzeigen von SVGs zwei Module an. Mit dem Modul *Transcoder* lassen sich SVGs in ein Pixelgrafikformat rendern. Das Modul *JSVGCanvas* ermöglicht das Betrachten bzw. Anzeigen von SVGs innerhalb eines Frames.

5.1.2.1 Transcoder Modul

Wie bereits erwähnt, bietet Batik die Möglichkeit, SVG-Dokumente in ein Pixelgrafikformat zu rendern. Dabei stehen das JPEG, das PNG und das TIFF Format zur Verfügung. Das folgende Beispiel transformiert eine SVG in ein JPEG Bild. Der Ablauf zur Wandlung in PNG oder TIFF ist derselbe.

[2] Writer-Klassen geben Objekten an definierte Ausgabeperipheriegeräte, wie z.B. Datei, Bildschirm oder Drucker, aus.

[3] Erweitert die Funktionalität eines Programms. Im Allgemeinen werden Plugins von Drittanbietern erstellt.

Beispiel: Wandeln einer SVG in eine Pixelgrafik im JPEG Format

```
import java.io.*;
import org.apache.batik.transcoder.image.JPEGTranscoder;
import org.apache.batik.transcoder.TranscoderInput;
import org.apache.batik.transcoder.TranscoderOutput;
public class SaveAsJPEG {
  public static void main(String [] args) throws Exception {
    // Instantiiere einen JPEG transcoder
    JPEGTranscoder t = new JPEGTranscoder();
    // setzen der Eigenschaft QUALITY
    t.addTranscodingHint(JPEGTranscoder.KEY_QUALITY, new Float(.8));
    // Instantiiere TranscoderInput
    String svgURI = new File(args[0]).toURL().toString();
    TranscoderInput input = new TranscoderInput(svgURI);
    // instantiiere transcoder output
    OutputStream ostream = new FileOutputStream("out.jpg");
    TranscoderOutput output = new TranscoderOutput(ostream);
    // speichern der JPEG Grafik
    t.transcode(input, output);
    // Buffer leeren und schließen der i/o - streams
    ostream.flush();
    ostream.close();
  }
}
```

Es muss keine SVG-Datei als *input* für den *Transcoder* verwendet werden, es kann ebenso ein SVG-Dokument verwendet werden. Außerdem muss der *output* auch keine Datei sein. Es ist auch ein `ByteStream` möglich, welcher dann z.B. in einem Java Swing `JLabel` angezeigt werden kann.

Das Wandeln in eine Pixelgrafik bietet außer der im Beispiel verwendeten Eigenschaft `QUALITY` noch weitere Eigenschaften. So kann mit den Schlüsseln `KEY_HEIGHT` und `KEY_WIDTH` die Ausgabegröße der Pixelgrafik bestimmt werden und mit `KEY_AOI` wird durch Angabe eines Rechtecks als Parameter nur ein Ausschnitt der gesamten SVG gerendert. Weitere Eigenschaften stehen in der mitgelieferten Dokumentation zu Batik [5].

5.1.2.2 JSVGCanvas

Canvas ist Englisch und bedeutet Gewebe oder Leinwand. In Java wird der Canvas meist als eine Art Frame zur Darstellung von Grafiken benutzt. Von batik wird ein spezieller `JSVGCanvas` benutzt. Er dient lediglich dazu, SVG-Dokumente am Bildschirm innerhalb eines Fensters oder Frames in Java anzuzeigen. Dabei werden die SVGs ebenfalls wie in Abschnitt 5.1.2.1 gerendert, wobei dies automatisch geschieht, wie man am folgenden Beispiel sehen kann.

Zuerst wird eine Instanz vom Objekt des Typs `JSVGCanvas` angelegt.

```
JSVGCanvas svgCanvas = new JSVGCanvas();
```

Anschließend wird mit der Methode `setURI(String uri)` dem Canvas die darzustellende SVG übergeben. Bei dem Parameter `uri` handelt es sich entweder um eine Datei (1) oder um ein im Hauptspeicher befindliches Dokument (2).

Entweder

```
(1)  String svgUri = new File("svgFile.svg").toURL().toString();
```

oder

```
(2)  String svgUri = mySVGDocument.getURL();
```

```
try {
  svgCanvas.setURI(svgUri);
} catch (IOException ex) {
  ex.printStackTrace();
}
```

Die Instanz `svgCanvas` kann nun jedem beliebigen `JFrame` oder `JPanel` zur endgültigen Darstellung am Monitor hinzugefügt werden.

Besonderheiten des `JSVGCanvas` sind die EventHandler. Nachfolgend werden Listener für die wichtigsten Ereignisse aufgeführt:

- **SVGDocumentLoaderListener:** Feuert ein Event bei der Ladephase: Wenn aus einer SVG-Datei ein SVG DOM Baum konstruiert wird.
- **GVTTreeBuilderListener:** Feuert ein Event bei der Aufbauphase: Wenn ein GVT (Graphic Vector Toolkit) Baum aus einem SVG DOM Baum konstruiert wird. Der GVT Baum wird anschließend gerendert.
- **GVTTreeRendererListener:** Feuert ein Event beim Beginn der Renderphase: Wenn ein GVT Baum in eine Pixelgrafik gewandelt wird.
  Anmerkung: Bei dynamischen Dokumenten wird dieser Event nur einmalig bei der Initialisierung des Renderers ausgelöst.

Weitere EventHandler sind in der Dokumentation zu Batik beschrieben [5].

5.1.3 Erfahrungen

Innerhalb meiner Diplomarbeit habe wurde auf das `JSVGCanvas` verzichtet, weil es bei `repaint` Methoden zu Schwierigkeiten kam. Oftmals wurde auf dem `JSVGCanvas` das `repaint` nicht ausgeführt. Speziell auf langsameren Computern oder bei häufigen `repaints` kam es immer wieder zu diesem Fehler.

Ein weiteres Problem von Batik ist, dass das erste Rendern einer SVG sehr lange dauert. Dabei ist nicht das Rendern selbst ausschlaggebend, sondern die lange Ladezeit sämtlicher Bibliotheken, welche von Batik benutzt (oder auch nicht benutzt, aber trotzdem geladen) werden müssen. Werden also mehrere Grafiken hintereinander gerendert, so braucht man ab der zweiten SVG fürs Rendern wesentlich weniger Zeit.

Beispiel: Zeitmessung bei der Darstellung von zwei gleichgroßen, jedoch unterschiedlichen SVGs:

Erstmaliges Laden der SVG (inklusive Laden aller Bibliotheken): 3085 ms
Erstmaliges Erstellen GVT Baum 280 ms
Erstmaliges Rendern der Grafik und öffnen des Panels: 892 ms

Erneutes Laden einer Grafik gleicher Größer: 260 ms
Erstellen des GVT Baums: 180 ms
Rendern der Grafik und öffnen des Panels: 351 ms

Die Messungen wurden auf einem Pentium II 400 MHz unter Borland® JBuilder™ Version 5.0 zehnmal durchgeführt und anschließend die Durchschnittswerte gebildet.

Im Beispiel lässt sich gut der Zeitaufwand zum Laden der Bibliotheken ablesen. Letztendlich werden für das einfache Rendern von SVGs in eine Pixelgrafik nie alle Bibliotheken benötigt, jedoch war es im Rahmen dieser Diplomarbeit aus Zeitgründen nicht möglich, eine lauffähige Minimalversion von Batik zu erstellen. Hierzu noch ein Zitat von Vincent Hardy, führender Entwickler von Batik [9]:

„About the Batik size. The SVG specification is over 500 pages long and uses other specs (DOM, CSS) which are also very large. It takes a lot of code to implement SVG 1.0 fully (which is what Batik is doing) and this explains the size. There are things like filter effects and very sophisticated text support (SVG even has its own font format) which take code to implement... This also explains the initial class loading time because there are lots of classes to load. Sometimes, users think that the time taken before visualising the first SVG document is taken up by rendering. It is not: most of that time is taken by class loading. If you reload the document after the first rendering, you will see what the real rendering time is.

Note also that Batik is a toolkit and that the overall size includes several modules which you do not need for rendering (the binary distribution contains jars which are correctly split up). For example, you do not need the SVGGraphics2D for viewing and you do not need the font converter either. Furthermore, if you are interested in only a subset of the Batik features, it is possible to achieve because the Batik design allows it.

About the performance, it is always a difficult question when it is not qualified with numbers compared to expectations. We have and are using our performance tuning skills to the best of our abilities and, as far as we know, we are doing well in the world of SVG renderers, even when compared to native implementations. For example, our gradients and filters are known to be quite efficient. We are working on improving the speed of our text rendering which is not an easy task because, as I mentioned, SVG has a lot of very sophisticated text features (see the text and fonts examples in the Batik distribution).

Now, there is a clear need for lean and mean SVG, as you put it. This is why the SVG working group in the W3C (which I am part of) is working on profiles of SVG (we call them SVG Tiny and SVG Basic) with a restricted feature set which should allow leaner implementations. The Batik project may provide an implementation for these profiles at some point, but this has not been discussed or decided yet"

Auch wenn anhand dieses Zitates die Vermutung groß ist, dass die Erstellung einer Minimalversion von Batik relativ simpel wäre: Es ist nicht so.

Bis auf die genannten Probleme kam es mit Batik zu keinen weiteren Schwierigkeiten.

5.2 JTree von Java Swing

Die Aufgabenstellung dieser Diplomarbeit beinhaltet unter anderem, dass der horizontale Baum dieselbe Funktionalität wie der `JTree` von Java Swing besitzt, deswegen wird hier der `JTree` genauer beleuchtet.

5.2.1 Bäume (Trees)

Ein Baum ist ein Kontrollinstrument um Daten hierarchisch darzustellen. Bei Computern wird die Baumdarstellung hauptsächlich zur Anzeige der Inhaltsverzeichnisse von Datenträgern (Festplatte, Diskette, CD-Rom, Netzlaufwerk, usw.) genutzt.

Das elementare Objekt eines Baumes wird Knoten genannt. Ein Knoten wiederum repräsentiert ein Datum innerhalb einer Hierarchie. Daraus folgt, dass ein Baum aus mindestens einem, jedoch meist mehreren Knoten aufgebaut ist. Der Wurzelknoten oder auch Root ist der Anfang einer hierarchischen Struktur und kommt nur einmal vor.

Knoten unterhalb des Wurzelknotens werden Kindknoten oder einfach Knoten genannt. Knoten, welche wiederum keine eigenen Kindknoten haben, sind so genannte Blattknoten oder kurz Blätter. Jeder Knoten, der kein Blatt ist, kann beliebig viele Knoten und Blätter (sogar den Null Knoten[4]) enthalten. Diesen Knoten mit seinen Kindknoten und Blättern bezeichnet man als Zweig.

[4] Null Knoten ist ein Knoten, der kein Datum repräsentiert. Prinzipiell ein leeres Objekt.

Man spricht von einem vertikalen Baum, wenn jede Zeile eines Baumes nur jeweils einen Knoten oder ein Blatt enthält.

`JTree` ist derzeit die einzig fertige Implementierung einer Baumdarstellung in Java. Jedoch bietet der `JTree` nur die vertikale Baumdarstellung an (vgl. Abbildung 5-3).

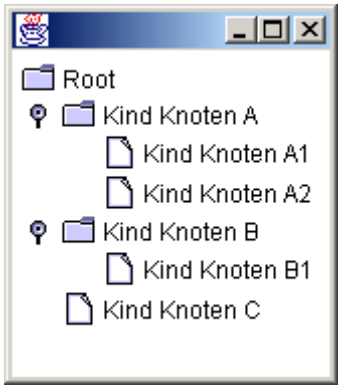


Abbildung 5-3: Beispiel eines JTree's

Root ist der Wurzelknoten, A, B und C sind Kindknoten von Root. A1, A2, B1 sind Blattknoten, wobei A1 und A2 auch Kindknoten von A sind, und B1 ein Kindknoten von B ist. Da C keine weiteren Kinder hat, ist C auch ein Blattknoten.

Der Pfad eines Knotens wird durch den Knoten selbst mitsamt seiner Elternknoten angegeben. Somit wäre der Pfad von A1: {[Root],[A],[A1]}

5.2.2 Struktureller Aufbau des JTree´s

In vorigen Abschnitt wurde der allgemeine Aufbau eines Baumes beschrieben. Jedoch ist noch unklar, wie der ganze Baum bzw. ein Knoten beim `JTree` realisiert wird. Die Klasse `JTree` ist „nur" die Schnittstelle für den Programmierer. `JTree` benutzt eine Menge von Subklassen, die transparent sind, also nur beim Kompilieren hinzu gelinkt werden.

Jeder Knoten (egal, ob Root-, Kind- oder Blattknoten) wird bei `JTree` von der abstrakten Klasse `TreeNode` abgeleitet. Dem Entwickler bleibt es selber überlassen, ob er eine eigene Klasse zum Repräsentieren eines Knotens erstellt, oder ob er die von Java Swing mitgelieferte Klasse `DefaultMutableTreeNode` nutzt.

Alle Knoten werden so ineinander geschachtelt, dass sie einer Hierarchie entsprechen. Beispielsweise werden die Knoten A1 und A2 aus Abbildung 5-3 dem Knoten A untergeordnet, B1 dem Knoten B, und anschließend werden A, B und C dem Wurzelknoten (Root) hinzugefügt.

Der so verschachtelte Wurzelknoten wird einer Klasse übergeben, welche von `TreeModel` abgeleitet ist. Auch hier ist es egal, ob der Entwickler eine eigene Klasse erstellt oder die mitgelieferte Standardklasse `DefaultTreeModel` nimmt.

Abschließend wird das Modell der Instanz des `JTree`‘s das Modell übergeben - der Baum ist fertig.

Der Position eines Knotens innerhalb des Baumes wird über den Pfad des jeweiligen Knotens ermittelt. Die Pfadinformationen werden in der Klasse `TreePath` gehalten. Sie enthält jeweils einen beliebigen Knoten mitsamt seiner Elternknoten.

Für die Darstellung der einzelnen Knoten am Bildschirm ist eine eigene Klasse verantwortlich. Auch hier kann der Entwickler selbst eine Klasse erstellen, welche von `TreeCellRenderer` abgeleitet ist, oder er kann die Standardklasse `DefaultTreeCellRenderer` benutzen.

5.2.3 Funktionalität und Interaktion

Das Design des `JTree` basiert auf einer MVC (Model Viel Controller) Architektur. Somit kann er für sehr komplizierte oder auch einfache Strukturen (bzw. Anwendungen) verwendet werden. `JTree` bietet eine Vielzahl von Listenern zum Eventhandling an. Der Nutzer kann somit eine oder mehrere beliebige Aktion(en) durch ein Ereignis auslösen. Durch folgende Ereignisse werden Events „gefeuert“:

- Bevor ein Zweig zugeklappt wird,
- nachdem ein Zweig zugeklappt wurde,
- bevor ein Zweig aufgeklappt wird,
- nachdem ein Zweig aufgeklappt wurde und
- wenn ein anderer Knoten selektiert wird.

Weiterhin wird dem Nutzer mitgeteilt, wenn sich ein Knoten, ein Zweig oder das gesamte Modell ändert.

Standardmäßig wird ein Zweig auf- oder zugeklappt, wenn auf dem jeweiligen Konten ein Doppelklick der Maus erfolgt. Der Entwickler muss sich somit nicht um die Implementierung diese Funktion kümmern.

5.2.4 Flexibilität des JTree’s

Der `JTree` kann, wie am Anfang des Abschnittes 5.2 erwähnt, für einfache und komplizierte Anwendungen genutzt werden. Damit er also universell einsetzbar ist, muss er sehr variabel gestaltet werden können. Dies erreicht der `JTree` zum Einen durch die Möglichkeit, eine eigene Knoten- und Modell-Klasse erstellen zu können, zum Anderen kann ein Knoten eine beliebige Klasse bzw. ein beliebiges Objekt einer Klasse repräsentieren. Damit auch eigene Objekte dargestellt werden können, ist das Erstellen eines eigenen `TreeCellRenderer`’s erforderlich. Dies soll nochmals im folgenden Beispiel verdeutlicht werden:

Beispiel: Um das Beispiel so übersichtlich wie möglich zu gestalten, wurde die Klasse `myClass` implementiert, die nur den Namen des Knotens sowie ein Icon enthält. Die Datenstruktur der Klasse `MyClass` sieht wie folgt aus:

```
String knotenName;
Icon knotenSymbol;
```

Die Struktur des Baumes wird im `DefaultTreeModel` gehalten und die einzelnen Knoten sind `DefaultMutableTreeNodes`, welche eine Instanz der Klasse `myClass` als `userObject` enthalten. Ein Zweig wird wie folgt definiert.

```
...
MyClass knoten1 = new MyClass("Knoten 1", icon1);
MyClass knoten1A = new MyClass("Knoten A", iconA);
MyClass knoten1B = new MyClass("Knoten B", iconB);
DefaultMutableTreeNode nodeA = new DefaultMutableTreeNode(knoten1);
DefaultMutableTreeNode node1A = new DefaultMutableTreeNode(knoten1A);
DefaultMutableTreeNode node1B = new DefaultMutableTreeNode(knoten1B);
nodeA.add(node1A);
nodeA.add(node1B);
...
```

Die Variablen `icon1`, `iconA` und `iconB` sind vom Typ `Icon`. Zum Darstellen der `userObjects` ist eine eigene `TreeCellRenderer` Klasse erforderlich. Diese Klasse könnte wie folgt aussehen:

```
class MyCellRenderer extends TreeCellRenderer {
  Component getTreeCellRendererComponent(JTree tree, Object value,
    boolean selected, boolean expanded, boolean leaf, int row, boolean hasFocus)
  {
    DefaultMutableTreeNode node = (DefaultMutableTreeNode)value;
    Object nodeObject = node.getUserObject();
    // Prüfe, ob das userObject vom Typ MyClass ist.
    if(nodeObject instanceof MyClass) {
      // wenn userObject vom Typ MyClass, dann wird die Darstellung eines
      // Knotens "zusammen gebaut".
      JPanel myPanel = new JPanel(new FlowLayout());
      JLabel iconLabel = new JLabel(((MyClass)nodeObject).knotenSymbol);
      JLabel text = new JLabel(((MyClass)nodeObject).knotenName);
      myPanel.add(iconLabel);
      myPanel.add(text);
      if(hasFocus)
        mYPanel.setBackground(Color.blue);
      return myPanel;
    } else {
      //Standardrückgabe einer Komponente
```

```
        }
    }
}
```

Jetzt muss der Instanz des `JTree`'s noch eine Instanz von `MyCellRenderer` übergeben werden:

```
myJTree.setCellRenderer(new MyCellRenderer());
```

Das Ergebnis dieses Beispiel sieht in etwa wie folgt aus:

Abbildung 5-4: Mögliche Darstellung einer eigen erstellten JTree Komponente

Nun hängt es alleine an der Kreativität des Programmierers, wie er die `JTree` Komponente verwendet. Ein Knoten könnte zum Beispiel jederzeit auch eine ganze Liste anderer Knoten beinhalten. Somit würde sich theoretisch auf Umwegen auch ein horizontaler Baum mit dem `JTree` erstellen lassen. Warum sich aber dieser Ansatz praktisch jedoch nicht realisieren lässt, möchte ich im folgend Abschnitt erläutern.

5.2.5 Erfahrungen

Der `JTree` nutzt zur graphischen Darstellung des vertikalen Baumes Java Swing Komponenten wie `JLabel`, `JPanel` etc. Zu jeder Komponente gibt es so genannten Look&Feel oder auch UI (User Interface) Klassen. Sie sind für das Zeichnen der Komponenten am Bildschirm verantwortlich. Es wäre denkbar, seine eigene(n) Klasse(n) zur Darstellung des Baumes zu schreiben und die Swing-Klassen zu ersetzen- So könnte man auch gezielt Einfluss auf das Design und insbesondere die Position der einzelnen Komponenten nehmen.

Auf den ersten Blick erscheint es auf diesem Wege möglich, aus einem vertikalen Baum durch andere Positionierung der Knoten einen horizontalen Baum zu zeichnen. Dann wäre es jedoch verwunderlich, dass diese Option beim `JTree` nicht angeboten wird. Allein der Fakt, dass trotz des offensichtlichen Bedarfs eine horizontale Darstellung hier nicht möglich ist, gibt schon einen Hinweis darauf, dass diese Aufgabe nicht so einfach zu lösen ist. Dies mag darauf zurückzuführen sein, dass die Berech-

nung der Position von Komponenten und Verbindungslinien unter Aufrechterhaltung aller anderen `JTree`-Funktionalitäten in einem horizontalen Baum weitaus komplizierter ist als bei vertikaler Darstellung. Es gibt allerdings durchaus fertige Komponenten im Internet, welche eine horizontale Anzeige einer Hierarchie realisieren [10] - allerdings nur mit sehr eingeschränkter Funktionalität. Dies liegt vor allen daran, dass der einfache Zugriff auf die Knoten anhand der Reihennummer bei horizontalen Baumstrukturen nicht möglich ist.

Eine reine Änderung der Darstellungsklassen ist beim `JTree` auch deshalb nicht möglich, weil verschiedene Methoden des `JTree`'s ungenügende oder gar falsche Ergebnisse liefern würden. Beispielsweise kann die Methode `getPathForRow( .. )` kein gültiges Ergebnis mehr liefern, da bei einer horizontalen Baumdarstellung logischerweise mehrere Knoten in einer Zeile sein können.

Somit ist der `JTree` aufgrund seiner Klassenstruktur für die Lösung der Aufgabe nicht geeignet und die Implementierung einer eigenen Klasse zur Baumdarstellung nötig.

Kapitel 6

PFLICHTENHEFT

Dieses Pflichtenheft wurde im Rahmen einer Diplomarbeit für die Firma Dornier EADS erstellt. Es soll eine Komponente zur Darstellung von horizontalen Truppenbäumen unter Verwendung des SVG (XML) Grafikformats in Java erstellt werden.

Grundlagen und benutze Werkzeuge wurden in den vorherigen Kapiteln erläutert.

Bevor das Pflichtenheft erstellt wurde, gab es eine Bedarfsanalyse, wie die Komponente auszusehen hat und welche Funktionen erfüllt werden müssen. Anschließend gab es anhand dieser Analyse eine Recherche über hilfreiche Teillösungen oder fertige Produkte.

6.1 Bedarfsanalyse

Die Baumdarstellung soll horizontal mit der Hierarchie von oben nach unten erfolgen (siehe Abbildung 6-1).

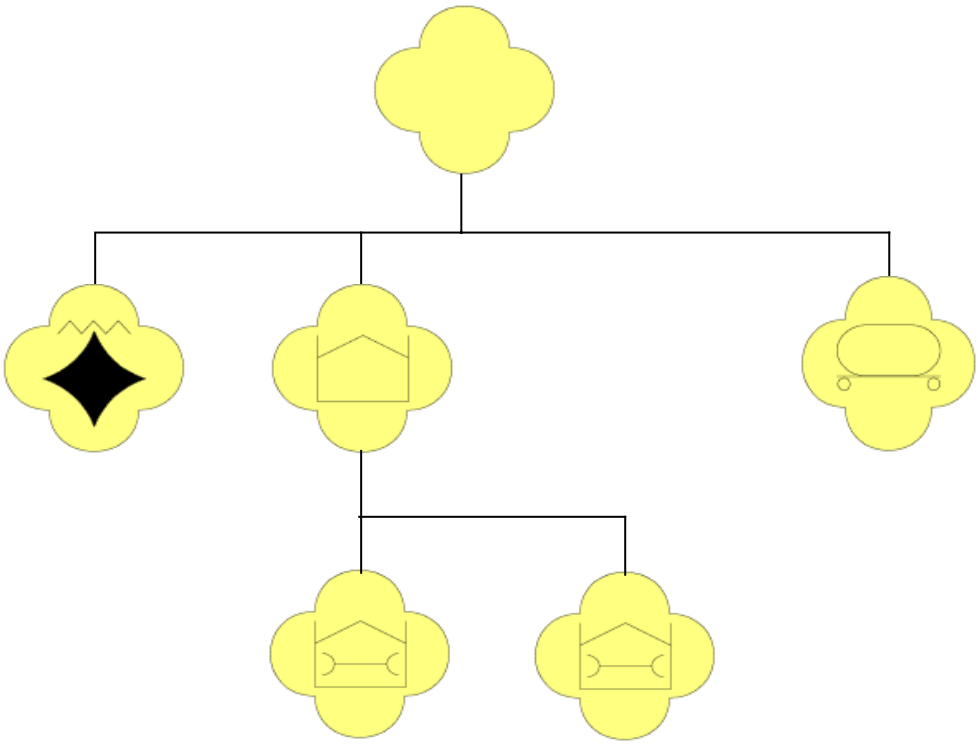


Abbildung 6-1: Beispiel der Darstellung eines horizontalen Truppenbaumes

Bisher wurde der Truppenbaum mit einer vertikalen Baumdarstellung durch den `JTree` realisiert. Damit bei der Umstellung auf eine waagerechte Baumkomponente nicht auch die Baumstruktur bzw. der Aufbau des Baumes neu erstellt werden muss, ist eine wichtige Anforderung, dass der horizontale Baum (`HTree`) das Modell und die Knoten des `JTree` benutzt und auch die gleiche Funktionalität wie der `JTree` besitzt.

Der horizontale Baum soll in der angezeigten Symbolgröße variabel sein. Ein Symbol wird als Instanz eines `SVGDocument`'s zur Verfügung gestellt.

Die Instantiierung der horizontalen Baumkomponente soll ein von `JComponent` abgeleitetes Objekt liefern, das sowohl Drag&Drop[1]-fähig ist, als auch das `Scrollable` Interface implementiert.

Da das Gesamtsystem (JOCCIS) auch in den Vereinigten Arabischen Emiraten eingesetzt wird, ist ein horizontale Ausrichtung der Darstellung sowohl von Links nach Rechts, als auch von Rechts nach Links erforderlich.

Basierend auf dieser Analyse ist das Pflichtenheft entstanden.

6.2 Marktübersicht

Die Recherche über horizontale Baumdarstellungen als JavaBean ergab, dass Zurzeit noch keine komplette Lösung auf dem Markt. Allerdings trifft man auf zahlreiche Anfragen über solch eine Implementierung (unter anderen in Newsgroups), was den Bedarf für solch eine Komponente belegt. Im Folgenden werden die einzig existierende Teillösung sowie Hinweise zur Implementierung solcher Bäume vorgestellt.

6.2.1 Jazz – Zoomable User Interface

Dieses Produkt dient hauptsächlich der Erstellung bzw. Darstellung von Grafiken, welche eine Vergrößerungsfunktion benötigen. Weiterhin sind aber auch Klassen zu Visualisierung von Bäumen vorhanden. Es können sowohl horizontale als auch vertikale Baumstrukturen dargestellt werden. Allerdings ist bei einem mit Jazz erstellten Baum kaum Interaktivität möglich. So kann ein Knoten zwar selektiert werden, jedoch können die Knoten nicht modifiziert und es können auch keine Zweige zu- oder aufgeklappt werden. Auch ist die Verwendung von `TreeModel` des `JTree`'s (siehe Abschnitt 5.2) nicht möglich.

[1] Das Ausführen von Operationen in einer grafischen Benutzeroberfläche. Es werden dabei Objekte mit der Maus am Bildschirm verschoben.

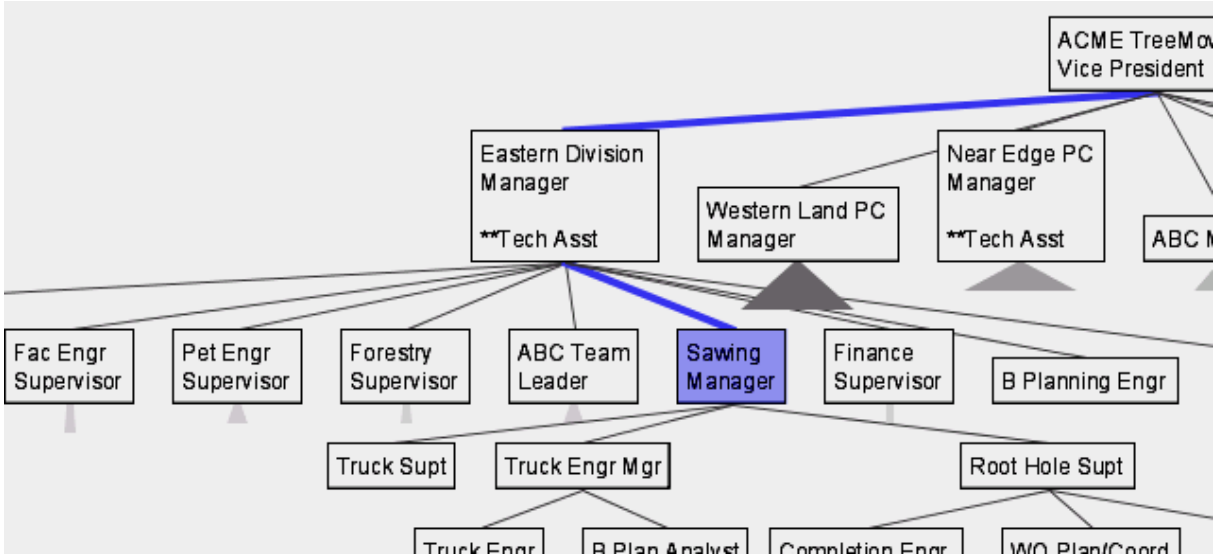


Abbildung 6-2: Ausschnitt aus einer mit Jazz erstellten Anwendung der University of Maryland

Jazz wurde von der *Human-Computer Interaction Lab* (HCIL) an der Universität von Maryland (USA) erstellt und steht als open Source[2] zur Verfügung[10]. Auch wenn somit die Möglichkeit besteht, Quellcode teilweise wiederzuverwenden, sind die erforderlichen Änderungen so umfangreich, dass von einer Nutzung der Klassen abgesehen wurde. Diese Entscheidung wurde insbesondere vor dem Hintergrund getroffen, dass in der Aufgabenstellung der Diplomarbeit eine an den `JTree` angelehnte Realisierung gefordert wird.

6.2.2 Hinweise bzw. Vorschläge zur Implementierung

Das Diskussionsforum der Java Homepage von Sun enthält einige interessante Lösungsansätze, welche für die Aufgabenstellung leider auch nicht verwendet werden können.

(1) **`TreeNode` des `JTree` enthält ein Feld von weiteren Knoten**:

Der `JTree` erlaubt in jeder Zeile nur einen Knoten, der jedoch jedes beliebige Java-Objekt enthalten darf. Da auch die Darstellung der Knoten durch Hinzufügen eines eigenen `TreeCellRenderer` (siehe Abschnitt 5.2.4) durch den Programmierer selber gestaltet werden kann, wird nun vorgeschlagen, einfach ein Feld von Knoten in einen `TreeNode` zu „stecken“. Dies würde aber nur funktionieren, solange die Knoten unterhalb des Rootknoten keine Kinder mehr enthalten. Doch selbst dann ist es sehr schwer, die richtige Position der einzelnen Knoten für die Darstellung zu ermitteln. Außerdem würden viele Funktionen des `JTree`'s (z.B.: `getLeadSelectionRow()`) ein falsches Ergebnis liefern.

(2) **Ändern der Klasse `TreeUI`**:

Wie aus Abschnitt 5.2.2 ersichtlich, gibt es zu jeder Komponente von Java Swing eine Klasse, welche die Darstellung übernimmt. Im Falle des `JTree` heißt diese Klasse `TreeUI`. Der Vorschlag lautet, eine eigene Klasse zur Darstellung zu implementieren, welche von `TreeUI` abgeleitet ist und die einzelnen Methoden nach Bedarf überschreibt.

Prinzipiell ist dies ein sinnvoller Ansatz, doch auch hier ist der Vorschlag unzureichend. Wie schon im vorherigen Ansatz würden verschiedene Methoden des `JTree` zu falschen oder unerwünschten Ergebnissen führen (siehe auch Abschnitt 5.2.5).

6.3 Produktvoraussetzungen

Da es sich hier um eine Komponente handelt, die zwar universell einsetzbar sein soll, jedoch in erster Linie in einem bestehenden System integriert wird, sind die Anforderungen stark an die Bedürfnisse des Gesamtsystems (JOCCIS) angepasst.

6.3.1 Betriebssystem und sonstige Software

Java-Anwendungen sind weitestgehend unabhängig von Betriebssystemen. Voraussetzung ist, dass auf dem Betriebssystem die Java Virtual Machine (JVM) läuft und die benutzten Bibliotheken zur Laufzeit zu Verfügung stehen. Die JVM ist ein abstraktes Maschinenmodell, das jedem Java-Interpreter zugrunde liegt. Die Tatsache, dass der Java-Compiler Objektcode für ein abstraktes Modell erzeugt, ermöglicht es, Java-Code auf allen Plattformen auszuführen, auf denen eine Implementierung einer JVM (also ein Java-Interpreter) existiert.
Erstellt wird die Komponente mit JDK Java von Sun. Zusätzlich wird das SVG Tool „Batik“ in der Version 1.5 beta4 von Apache [5] benutzt.

6.3.2 Hardware

Da Java plattformunabhängig ist, gilt dies auch ein Stück weit für die Hardware. Voraussetzung ist daher ein Computer, auf dessen Betriebssystem die Java Virtual Machine läuft. Da Java-Anwendungen meist viele Ressourcen benötigen und Batik ebenfalls eine sehr aufwendige Applikation ist, wird ein Computer, der vergleichbar mit einem Pentium II 400 MHz mit 128 MB Hauptspeicher ist empfohlen.

---

[2] Kostenlose Software, welche zusätzlich den Quellcode zur Verfügung stellt.

6.3.3 Anwender

Der Anwender, in diesem Falle der Entwickler (Programmierer) von Java-Applikationen, braucht Kenntnisse in Java Swing. Von großem Vorteil ist im Speziellen, wenn der Nutzer Erfahrungen mit dem `JTree` von Java Swing hat. Ansonsten werden keine weiteren Kenntnisse vorausgesetzt.

6.4 Nutzungsfälle

Da es sich hier um keine abgeschlossene Anwendung, sondern um eine Komponente zur Erstellung von Anwendungen handelt, werden hier die Funktionen der Komponente als Nutzungsfälle betrachtet.

6.4.1 Erstellung des Baum

Der Baum baut sich rekursiv auf. Die hierarchische Anordnung der Daten erfolgt äquivalent zum `JTree`. Benutzt wird das `TreeModel` aus der Swing Bibliothek, welches den Rootknoten enthält. Der Rootknoten wiederum besitzt seine Kindknoten. Die Kindknoten enthalten dann wieder Kinder usw. Nach Übergabe des `TreeModel`'s an die horizontale Baumkomponente ist der Baum fertig gestellt. Zur Darstellung der Struktur muss die Komponente zu einem Panel oder Frame hinzugefügt und angezeigt werden.

6.4.2 Knoten-Operationen

Bevor man Knoten hinzufügen, editieren oder löschen kann, muss zuerst der Knoten lokalisiert werden. Angelehnt an den `JTree` wird ein Knoten über die Mauskoordinaten ermitteln. Die entsprechenden Methoden liefern als Ergebnis ein `TreeNode`. Dieser kann dann gelöscht oder editiert bzw. es kann diesem Knoten ein neuer Kindknoten hinzugefügt werden. Alternativ kann man sich über die Koordinaten der Mausposition auch den Pfad eines Knotens (`TreePath`) als Ergebnis zurückgeben lassen. Daraus lässt sich dann die Position (Rangordnung) des Knotens innerhalb des Baumes ableiten. Aus dem `TreePath` lassen sich auch der aktive Knoten mitsamt seinen Elternknoten ermitteln.

Nach jeder Änderung der Baumstruktur wird automatisch ein Neuzeichnen des Baumes veranlasst.

6.4.3 Darstellung des Baums

Zum Zeichnen der einzelnen Knoten des `HTree` wird standardmäßig eine Klasse bereitgestellt. Diese Standardklasse zum Rendern der knoten erwartet als Knotensymbol eine SVG. Beinhaltet der Knoten keine SVG, zeichnet der Standardrenderer ein eigenes Symbol (siehe Abbildung 6-3). Diese Klasse kann jedoch bei Bedarf durch eine eigene Klasse ersetzt werden.

Abbildung 6-3: Standardsymbol eines Knotens im horizontalen Baum

Alle zusammengehörigen Knoten werden durch eine durchgezogene Linie miteinander verbunden. Mit den Linien wird angezeigt, welche Kindknoten zu welchem Elternknoten gehören (siehe). Existieren Kindknoten, die noch nicht aufgeklappt sind, werden sie durch eine senkrechte Linie unterhalb des Elternknotens angezeigt.

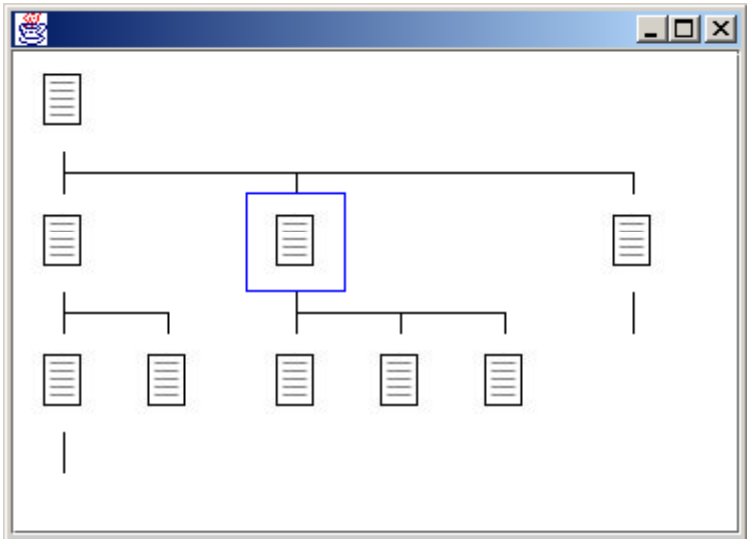


Abbildung 6-4: Knotenverbund beim horizontalen Baum (Ausrichtung von Links nach Rechts)

Die horizontale Ausrichtung des Baumes kann wie in Abbildung 6-4 von Links nach Rechts oder wie in Abbildung 6-5 von Rechts nach Links erfolgen.

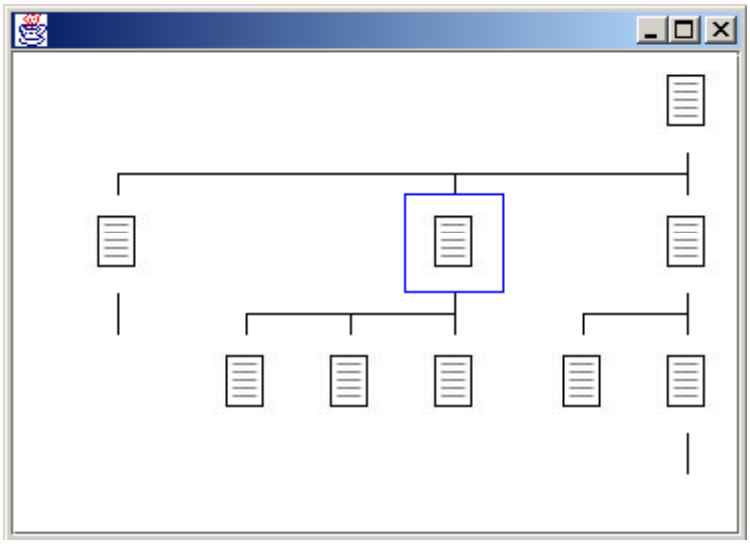


Abbildung 6-5: Baumdarstellung mit Ausrichtung von Rechts nach Links

Der Baum als solches ist eine Java Swing Komponente und dementsprechend von der Klasse `JComponent` abgeleitet. Diese Komponente kann wie jede andere Swing Komponente in einem Frame, einem Panel, einem Canvas oder einem Window dargestellt werden.

Die Markierung des aktiven Knotens (blauer Rahmen in Abbildung 6-5 ist in dem Standardrenderer enthalten.

6.4.4 Interaktion

Die Benutzung des Baumes ist analog zu der Benutzung des `JTree`. Bei einfachem Mausklick auf einen Knoten wird dieser Knoten der aktive Knoten. Bei Doppelklick hängt die Aktion vom Zustand und den Eigenschaften des Knotens ab. Ist der Knoten ein Elternknoten – er enthält also Kindknoten - klappt der Baum auf bzw. zu, je nachdem, ob er vorher zu- oder aufgeklappt war. Ist der Knoten ein Blattknoten und enthält somit keine Kinder, wird der Knoten „nur“ zum aktiven Knoten.

Ferner können sich beliebig viele Listener an der Komponente anmelden, die auf Mausklicks oder auf Änderung der Baumeigenschaften (z.B. Ausrichtung, Hintergrundfarbe, etc) reagieren. Tastaturfunktionen oder so genannte Shortcuts sind nicht implementiert, können aber bei Bedarf hinzugefügt werden.

Die Drag&Drop-Fähigkeit ist berücksichtigt worden, muss jedoch analog zum `JTree` selbst implementiert werden.

6.5 Aussichten

Der Bedarf an interaktionsfähigen horizontalen Bäumen zur Darstellung von Daten in hierarchischer Form kann als durchaus groß eingestuft werden. Zumal es dazu zurzeit noch keine konkrete Lösung

gibt. Umso größer ist das Interesse, diese Komponente nicht nur innerhalb von Java benutzen zu können, sondern auch in anderen objektorientierten Programmiersprachen. Innerhalb der Windowswelt gibt es dafür das ActiveX, welches Instanzen von Objekten auch mit graphischer Oberfläche systemweit zur Verfügung stellt und somit die Komponente von jeder anderen beliebigen Anwendung aus benutzbar macht. Alternativ steht die CAS COM Bridge für Java-Komponenten zur Anwendungskommunikation zur Verfügung.

Die Flexibilität der horizontalen Baumkomponente und die Interkommunikationsmöglichkeit durch ActiveX oder die CAS COM Bridge befürworten ein großes Einsatzgebiet innerhalb der Computerwelt.

Kapitel 7

SYSTEMENTWURF UND IMPLEMENTIERUNG

Der Systementwurf zeigt den strukturellen Aufbau der horizontalen Baumkomponente und die Integration dieser Komponente in Java Swing und im Speziellen die Nutzung der zu `JTree` gehörenden Klassen welche das Modell, den Pfad, die Knoten usw. repräsentieren. Abschließend wird in diesem Kapitel noch auf die Implementierung der Komponente eingegangen.

7.1 Struktureller Aufbau bei Java Swing

Damit sich die Komponente nahtlos in die generalisierte MVC[1] (GMVC) Struktur von Swing integrieren lässt muss sie selbst in dieser Struktur aufgebaut sein. GMVC ist eine Weiterentwicklung des MVC Musters. Bei dem MVC werden die fachlichen Aspekte (Model), die Darstellung (View) und die Reaktion auf Benutzereingaben (Controller) getrennt. Die Trennung dieser Aspekte wird durch verschiedene Klassen realisiert. Dadurch werden die Flexibilität und die Wiederverwendbarkeit der einzelnen Objekte erhöht (siehe Abbildung 7-1).

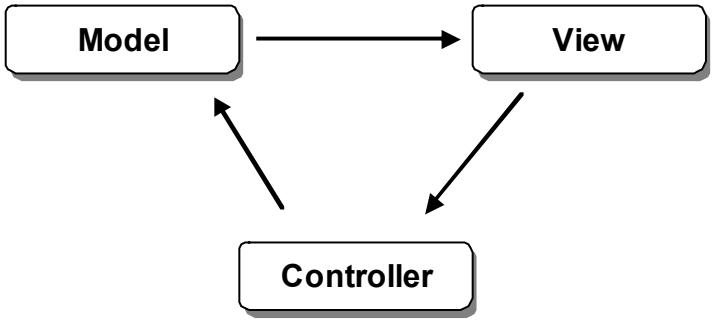


Abbildung 7-1: Klassische MVC Struktur

Im Unterschied zur klassischen MVC wachsen bei der GMVC aufgrund der gegenseitigen hohen Abhängigkeit das View- und Controller-Objekt zu einer Einheit, dem so genannten *Delegate*, zusammen.

---

[1] MVC steht für Model View Control: und ist ein bestimmtes Entwurfsmuster.

Somit muss der Controller nicht mehr die Layout Details des View Algorithmus kennen, um auf Ereignisse reagieren zu können (siehe Abbildung 7-2).

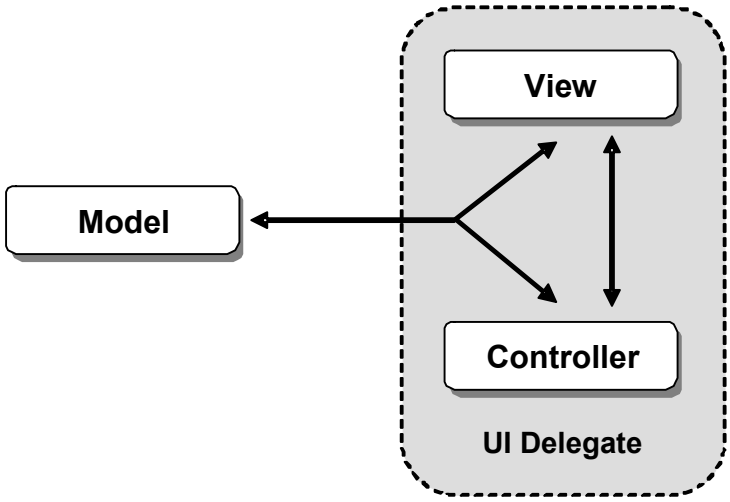


Abbildung 7-2: MVC Architektur von Swing (GMVC)

Bei Swing repräsentiert somit das Model den inneren Zustand einer grafischen Oberflächenkomponente (beispielsweise bei einem `JButton`, ob er enabled/disabled ist, oder wie sein Tastatur-Shortcut lautet).

Innerhalb des Delegates verarbeitet der Controller die Benutzereingabe und führt ggf. Änderungen im Model durch. Das View-Objekt übernimmt die Repräsentation des Models. Zu jeder Zeit hat ein `JComponent` ein einziges Model und ein einziges Delegate assoziiert.

Das Model einer Oberflächenkomponente implementiert das Model-Interface der dazugehörigen `JComponent`-Klasse. Damit beispielsweise eine Klasse als Model eines `JButton` agieren kann, muss sie das Interface von `ButtonModel` implementieren.

`DefaultButtonModel` ist das Standardmodell eines `JButton`'s. `BasicButtonUI` ist das Standard Delegate für `JButtons` unter Microsoft Windows.

7.2 Struktur der horizontalen Baumkomponente

Wie aus Abschnitt 7.1 ersichtlich, ist eine Trennung von Model und View/Control nötig, damit sich eigene grafische Oberflächenkomponenten in das Swing-Konzept einfügen lassen. Die zu erstellende horizontale Baumkomponente (`HTree`) wird von `JComponent` abgeleitet und soll eine vollständige Swing Komponente sein. Aus Kompatibilitätsgründen zu `JTree` benutzt der `HTree` das gleiche Model. Da die horizontale Baumkomponente in erster Linie SVGs als Symbole darstellen soll, wird das Model um einen eigenen Standardknoten, der ein SVG-Dokument als Attribut enthält, erweitert.

Das Delegate wird komplett neu erstellt, die Methodennamen sollen jedoch weitestgehend mit denen von `JTree` identisch sein.

7.2.1 Model

Das Model wird komplett aus der Bibliothek von `javax.swing.tree` übernommen, welche abstrakte und konkrete Klassen für die Baumstruktur (`TreeModel`) und für den Knoten (`TreeNode`) enthält. Erweitert wird das Model um die abstrakte Klasse `HTreeNode` und um die davon abgeleiteten, konkreten Klasse `DefaultHTreeNode`.

Das Model besteht eigentlich aus dem `TreeModel`, welches aber wiederum auf einzelnen Knoten basiert. Somit ist die Struktur der Knoten auch ein Teil des Models.

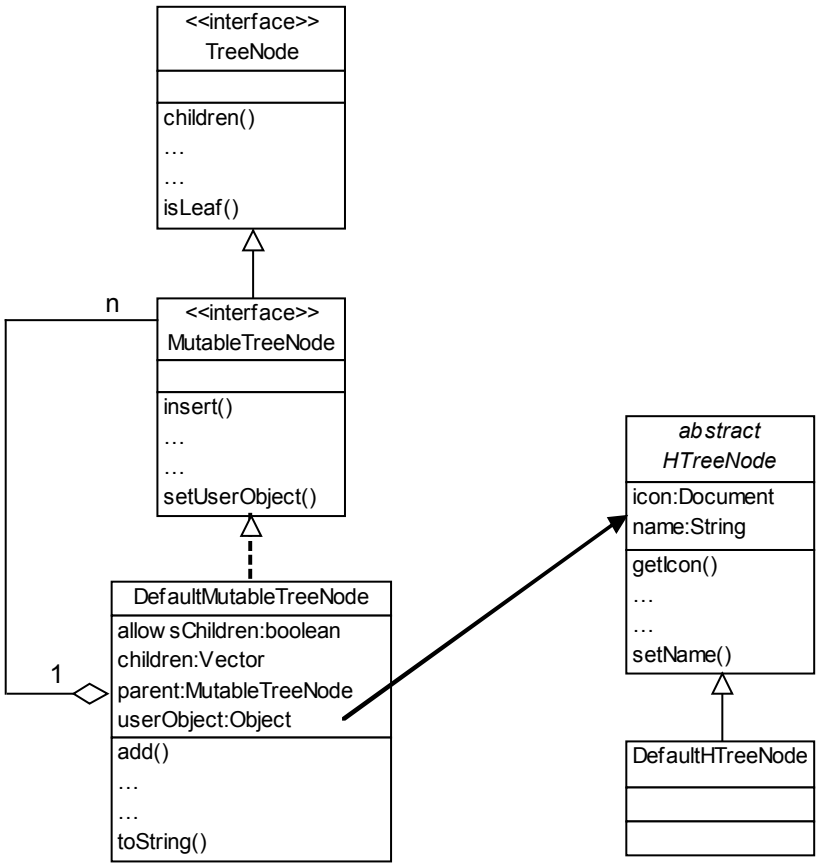


Abbildung 7-3: UML Diagramm der Knotenstruktur

Ein Knoten (`DefaultMutableTreeNode`) enthält beliebig viele Kindknoten und einen Elternknoten (außer der Rootknoten, der keinen Elternknoten hat). Jeder Knoten enthält referenziert eine Instanz der Klasse `DefaultHTreeNode`. Man spricht hierbei vom Benutzerobjekt. Prinzipiell kann ein Knoten jedoch eine Instanz jedes x-beliebige Java-Objektes enthalten. Zum Hinzufügen oder Löschen von Kindknoten stehen die Methoden `add( TreeNode t )` und `remove( TreeNode t )` zur Verfügung.

Das `TreeModel` definiert die Schnittstelle für das passende Datenmodell. Es bietet Methoden zur Ereignissteuerung (An- und Abmelden von Listenern) und Methoden, welche Informationen über den Baum (Rootknoten, Kinderzahl, usw) liefern. Die Klasse `DefaultTreeModel` ist die Standardimplementierung des `TreeModels`. Sie beinhaltet den Rootknoten des Typs `TreeNode` und die Liste aller angemeldeten Listener.

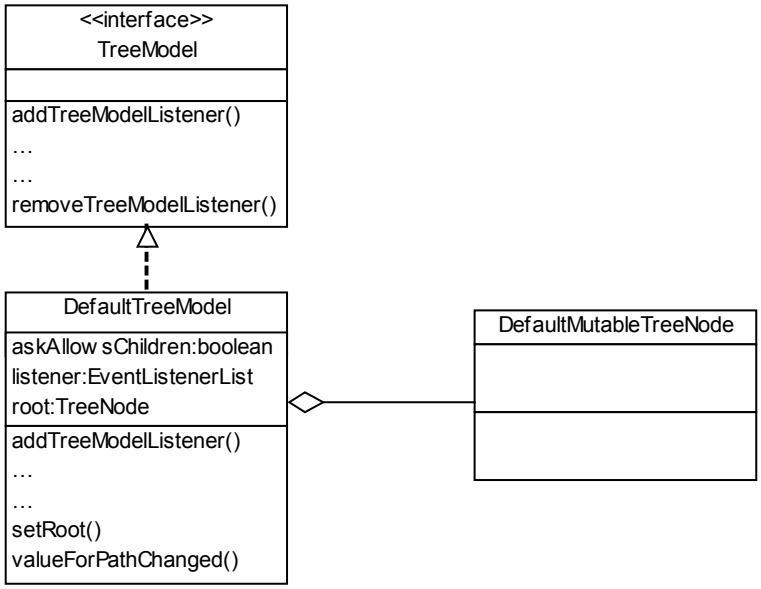


Abbildung 7-4: UML Diagramm des TreeModels

Das `TreeModel` benachrichtigt alle angemeldeten Listener, wenn sich ein Knoten ändert, ein Knoten hinzugefügt oder entfernt wird und wenn sich die Datenstruktur ändert.

7.2.2 Delegate

Das Delagate umfasst mehrere Objekte, welche die grafische Darstellung (View) und die Ereignissteuerung (Controller) repräsentieren. Das folgende UML-Diagramm (siehe Abbildung 7-5) zeigt alle zu Delegate gehörenden Klassen und ihre Assoziationen untereinander.

Die Darstellung der Widgets (graphische Oberflächenkomponenten) wird von der Look&Feel-Klasse `HTreeUI` realisiert. `HTreeUI` benutzt zum Rendern der einzelnen Knoten die austauschbare Klasse `DefaultTreeCellRenderer`. Der aktuelle Zustand der Widgets (z.B. Knoten aus- oder zugeklappt) wird in der Klasse `TreeStateCache` gehalten. Allgemeine Informationen der Komponente wie Hintergrundfarbe und darzustellende Größe werden in der Klasse `HTreeUIManager` statisch hinterlegt.

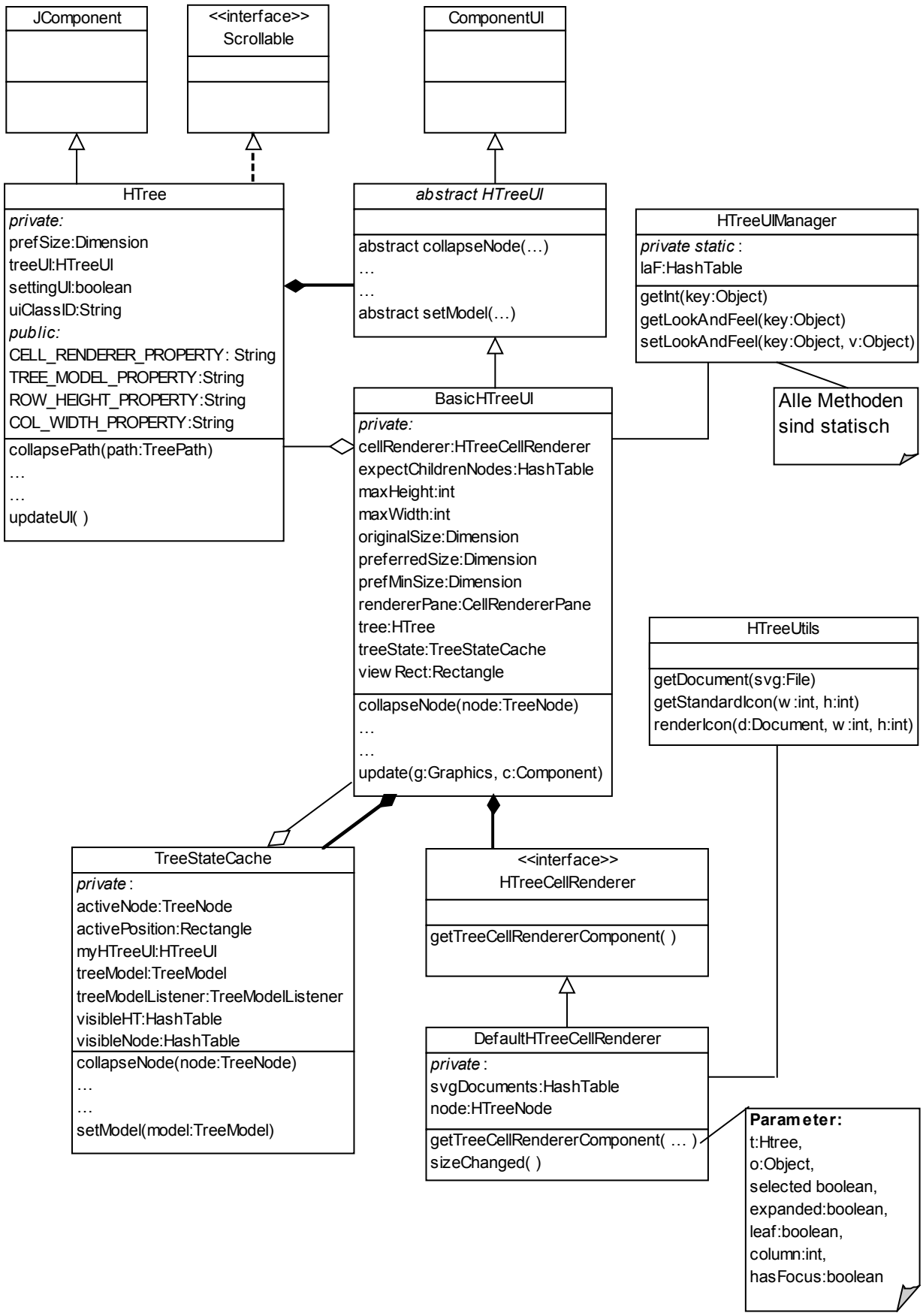


Abbildung 7-5: UML-Diagramm des Delegate's

Der Anwender instantiiert ein Objekt der Klasse `HTree`, welches ein Objekt der Klasse `HTreeUI` erzeugt. `HTree` dient als Schnittstelle für den Programmierer. Alle Ereignisse werden von diesem Objekt an die komponierte[2] Klasse `HTreeUI` weitergeleitet und von dort entsprechend verarbeitet. Damit Mausereignisse zum Markieren des aktuellen Knotens und zum Auf und Zuklappen von Knoten weitergeleitet werden, benutzt die Klasse `HTree` eine interne `MouseAdapter`-Klasse. Diese Klasse reagiert auf Mausklicks und übermittelt nach Verarbeitung des Mausklicks das Ereignis an alle angemeldeten `MouseListener` weiterleitet.

Die Klasse `HTreeUtils` dient dem `DefaultHTreeCellRenderer`. indem sie Standardicon erzeugt, übergebene Icons zu rendern und/oder um eine SVG-Datei in ein Objekt der Klasse `Document` wandelt.

Änderungen des Models bzw. einzelner Knoten werden direkt in der Klasse `TreeStateCache` registriert und dementsprechend verarbeitet.

7.3 HTree innerhalb von Swing

Um konform mit Java Swing zu sein, ist die Look&Feel-Klasse `BasicHTreeUI` und die Rendererklasse `DefaultHTreeCellRenderer` austauschbar. Voraussetzung dafür ist jedoch, dass die neue Look&-Feel-Klasse von der abstrakten Klasse `HTreeUI` abgeleitet wird und dass die Klasse zum Rendern der Symbole das Interface `HTreeCellRenderer` implementiert.

Die volle Integration in Swing wird dadurch erreicht, dass `HTreeUI` von der Oberklasse aller Look&Fell-Klassen, `ComponentUI` und `HTree` von der Oberklasse aller Komponenten, `Component`, abgeleitet ist. So wird auch sichergestellt, dass die von Swing vorgegebene Reihenfolge beim Zeichnen der Komponenten eingehalten wird und alle angemeldeten Listener über Ereignisse informiert werden.

[2] Komposition bedeutet, dass Klasse A eine Referenz auf eine Instanz der Klasse B als Objektvariable besitzt. B ist aus Sicht von Klasse A die komponierte Klasse.

7.4 Implementierung

Dieser Abschnitt behandelt Problemstellungen, welche bei der Implementierung aufgetreten sind, sowie deren Lösungsansätzen. Außerdem werden die Erfahrungen bei der Erstellung von JavaBeans beschrieben und letztendlich die fertige Komponente präsentiert.

7.4.1 Problemstellungen und Lösungen

Problem: Das erste Problem tritt bei der richtigen Positionierung der einzelnen Knoten auf. Hierzu muss erst einmal die hierarchische Rangordnung eines Knoten gefunden werden. Ist dies geschehen, muss man prüfen, ob der Knoten dargestellt werden soll. Ist dies der Fall, berechnet man dann die richtige Bildschirmposition des Knotens und veranlasst das Zeichen des Knotens an diesen Koordinaten. Nun muss die Verbindungslinie dieses Knotens zu seiner Umgebung ermittelt und gezeichnet werden. Abschließend wird geprüft, ob Kinderknoten vorhanden sind, damit auch zu diesen gegebenenfalls eine Verbindungslinie gezeichnet wird.

Lösung: Als Erstes muss man den Status (ist Knoten auf- oder zugeklappt) eines jeden Knotens speichern. (Die Überlegung, die Statusinformation beim Knoten selbst abzulegen, scheitert aus dem Grunde, dass das Modell des `JTree`'s verwendet wird, und solch ein Vorgehen bei diesem Modell nicht vorgesehen ist). Der Status des Knotens wird in der Klasse `TreeStateCache` in einer `Hashtable`[3] abgelegt. Der jeweilige `TreePath` eines Knotens dient hier als Schlüssel. Der zu diesem Schlüssel abgelegte Wert ist vom Typ `Boolean` und zeigt an, ob der Knoten aufgeklappt (`true`) oder zugeklappt (`false`) ist.

Damit die hierarchische Position eines jeden Knotens ermittelt werden kann, wird der darzustellende Teil des Baumes bei jedem Neuzeichnen rekursiv durchlaufen. Dadurch erhält man zu jedem Knoten die Rangordnung und kann somit dessen Bildschirmposition ermitteln sowie das Zeichnen des Knotens und der dazugehörenden Linien veranlassen. Aus Gründen der Effizienz wird vorher geprüft, ob der zu zeichnende Teil auch innerhalb des Viewports[4] liegt.

Problem: Java Swing-Komponenten besitzen die Eigenschaft der horizontalen Ausrichtung. Komponenten können also von links nach rechts oder auch von rechts nach links gezeichnet werden (siehe Abbildung 6-4 und Abbildung 6-5). Somit ist die Position der Knoten nicht nur von der Rangordnung,

---

[3] Hashing ist ein hocheffizientes Suchverfahren. Die Hashtable ist eine Tabelle mit zwei Spalten. In der einen Spalte wird der Wert und in der anderen der dazugehörige Schlüssel abgelegt. Gesucht wird ein Schlüssel über das Hashingverfahren. Java beinhaltet standardmäßig eine Hashtable-Klasse.

[4] Viewport bezeichnet in der Computergrafik einen Blick in ein Dokument oder eine grafische Darstellung, den man mit der Sicht durch ein Fenster vergleichen kann. Er zeichnet sich dadurch aus, dass Teile des Dokuments oder der Grafik, die außerhalb des Zeichenfensters liegen, nicht dargestellt, bzw. abgeschnitten werden.

sondern auch von der Ausrichtung abhängig. Die Koordinate der linken oberen Ecke des Rootknotens ist bei linksbündiger Ausrichtung (0,0). Bei rechtsbündiger Ausrichtung ist die Koordinate der Rootknotens von der Größe des darzustellenden (aufgeklappten) Baumes und von der Größe des Viewports abhängig.

Lösung: Bevor der Baum gezeichnet wird, muss die gesamte Größe als umschließendes Rechteck berechnet werden. Dazu wird der darzustellende Baum vor dem Zeichnen rekursiv durchlaufen. Hierbei werden die Anzahl der Blattknoten (in diesem Fall ist ein nicht aufgeklappter Elternknoten auch ein Blattknoten), die Anzahl der Elternknoten, welche aufgeklappt sind und Geschwister besitzen und die größte Rekursionstiefe ermittelt. Durch die Rekursionstiefe errechnet sich die maximale Höhe des darzustellenden Baumes:

*maximale Rekursionstiefe * (Symbolhöhe + vertikaler Abstand).*

Die maximale Breite errechnet sich wie folgt:

*(Xb * Symbolbreite) + ((Xb - 1) * Xh) + (Xe * 6Xh).*

Xb: Anzahl Blattknoten;
Xh: horizontaler Abstand;
Xe: Anzahl aufgeklappter Elternknoten.

Nun hat man die gesamte Größe des darzustellenden Baumes ermittelt und kann bei rechtsbündiger Ausrichtung die Position des Rootknotens und somit auch aller anderer Knoten bestimmen.

Problem: Ein weiteres kniffliges Thema ist die Bestimmung eines Knotens anhand einer x-y-Koordinate. Wenn der Nutzer auf einen Knoten klickt, muss dieser erkannt und dann die entsprechenden Aktionen durchgeführt werden. Beim `JTree` ist dies kein Problem, da alle Knoten untereinander und in gleichen Abständen zueinander dargestellt werden. Somit kann man einfach den Wert der y-Koordinate durch die Zeilenhöhe teilen. Schon weiß man, in welcher Zeile man sich befindet und kennt somit auch den entsprechenden Knoten.

Lösung: Bei jedem Neuzeichnen des Baumes werden alle darzustellenden Knoten in einer `Hashtable` abgelegt. Dabei ist der Schlüssel das umfassende Rechteck (beinhaltet x-y-Koordinate der linken oberen Ecke, sowohl Höhe und Breite des Rechtecks) des Knotens und als Wert des Schlüssels wird der `TreePath` des entsprechenden Knotens an dieser Stelle hinterlegt. Somit lässt sich sehr schnell der Knoten einer x-y-Koordinate ermitteln. Sobald sich die Baumstruktur ändert oder ein Knoten auf- bzw. zugeklappt wird, wird auch diese `Hashtable` gelöscht und neu geschrieben. Die `Hashtable` wird ebenfalls im der Klasse `TreeStateCache` gespeichert.

Problem: Wie aus Kapitel 5.1.3 zu erkennen ist, ist der Einsatz des SVG-Formats als Symbole für Knoten mit großem Ressourcenbedarf verbunden. Das Rendern von SVG-Dateien ist relativ langsam, selbst wenn alle dazu benötigten Bibliotheken von Batik bereits geladen sind.

Lösung: Die Klasse `DefaultHTreeRenderer` bekommt das Objekt eines Knotens als Parameter übergeben. Das zum Knoten gehörende Symbol (SVG) ist im Objekt enthalten. Damit bei jedem Neuzeichnen nicht auch die SVG neu gerendert werden muss, enthält der Renderer eine `Hashtable`, in der als Schlüssel das Objekt und als Wert die gerenderte Grafik steckt. Somit muss jedes Icon nur einmal gerendert werden. Zu beachten ist, dass die Klasse `DefaultHTreeCellRenderer` ja beliebig austauschbar ist. Es ist also anzuraten, bei einer eigenen Rendererklasse auch darauf zu achten, die Icons in einer `Hashtable` zu speichern.

7.4.2 Der HTree als JavaBean

In Kapitel 4 wird der generelle Aufbau von JavaBeans hinreichend erklärt. In diesem Abschnitt wird die JavaBean-Umgebung (Eigenschaften, Methoden und Events) des `HTree`'s gezeigt. Der HTree ist von der Klasse `JComponent` abgeleitet (vgl. Abbildung 7-5). Es werden hier nur noch die zu `JComponent` zusätzlichen bzw. überschriebenen Eigenschaften, Methoden und Events erläutert.

7.4.2.1 Eigenschaften

Der HTree besitzt alle von JComponent „geerbten" Eigenschaften. Alle zusätzlichen Eigenschaften sind einfache und öffentliche (public) Eigenschaften und werden über die folgenden setter- und getter-Methoden gesetzt.

- **`HTreeCellRenderer getCellRenderer()`**: Liefert die Klasse, welche für das Rendern der Knoten zuständig ist.
- **`void setCellRenderer( HTreeCellRenderer r )`**: Setzt die Klasse, welche zum Rendern der Knoten zuständig ist. Die Klasse `DefaultHTreeCellRenderer` ist standardmäßig als Rendererklasse instantiiert.
- **`int getColumnWidth()`**: Liefert die Spaltenbreite eines Knotens in Bildpunkten.
- **`void setColumnWidth( int w )`**: Setzt die Spaltenbreite. Der Parameter `w` drückt die Breite in Bildpunkten aus. Standardmäßig sind 40 Bildpunkte für die Spaltenbreite gesetzt.
- **`TreeModel getModel()`**: Liefert das benutzte TreeModel.
- **`void setTreeModel( TreeModel m )`**: Setzt das zu benutzende TreeModel.
- **`Dimension getPreferredScrollableViewportSize()`**: Liefert die Größe des benutzten Viewports.
- **`int getRowHeight()`**: Liefert die Zeilenhöhe der Knoten in Bildpunkten.
- **`void setRowHeight( int h )`**.Setzt die Zeilenhöhe. Der Parameter `h` drückt die Höhe in Bildpunkten aus. Standardmäßig sind 40 Bildpunkte für die Zeilenhöhe gesetzt.

- **`HTreeUI getUI()`**: Liefert die Look & Feel-Klasse. `BasicHTreeUI` ist standardmäßig die instantiierte UI-Klasse
- **`void setUI( HTreeUI )`**: Setzt die zu benutzende UI-Klasse.

Zusätzlich können noch der horizontale und vertikale Abstand der Knoten zueinander gesetzt werden. Angepasst an Swing, geschieht dies jedoch nicht in der Klasse `HTree`, sondern man kann die Abstände in der Klasse `HTreeUIManager` wie folgt setzen:

- *Horizontaler Abstand (von 20 Bildpunkten):*
  `HTreeUIManager.setLookAndFeel( HTreeUIManager.Y_GAP, new Integer( 20 ) )`
- *Vertikaler Abstand (von 10 Bildpunkten):*
  `HTreeUIManager.setLookAndFeel( HTreeUIManager.X_GAP, new Integer( 10 ) )`

Über die statische Klasse `HTreeUIManager` lassen sich natürlich auch die Abstände der Knoten zueinander auslesen. Standardmäßig beträgt der horizontale Abstand 20 Bildpunkte und der vertikale Abstand 2 Bildpunkte.

7.4.2.2 Methoden

Alle Methoden sind aus Gründen der Kompatibilität zum `JTree` in Namen und Signatur identisch mit den Methoden des `JTree`.

- **`void collapsePath( TreePath p )`**: Klappt den Knoten, welcher durch den Pfad `p` repräsentiert wird, zusammen.
- **`void expandPath ( TreePath p )`**: Klappt den Knoten, welcher durch den Pfad `p` repräsentiert wird, auf.
- **`TreePath getClosestPathForLocation( int x, int y )`**: Liefert den Pfad des Knotens, welcher den Bildschirmkoordinaten (`x, y`) am nächsten ist.
- **`TreePath getPathForLocation( int x, int y )`**: Liefert den Pfad des Knotens, der an der Bildschirmposition (`x, y`) ist. Befindet sich dort kein Knoten, wird `null` zurückgegeben.
- **`boolean isCollapsed( TreePath p )`**: Zeigt an, ob der Knoten, welcher durch den Pfad `p` repräsentiert wird, zugeklappt ist.
- **`boolean isExpanded( TreePath p )`**: Zeigt an, ob der Knoten, welcher durch den Pfad `p` repräsentiert wird, aufgeklappt ist.
- **`void updateTree()`**: Veranlasst das Neuzeichnen des Baumes.

7.4.2.3 Events

Im `HTree` lösen alle Eigenschaftsänderungen ein Ereignis bei den angemeldeten `PropertyChangedListener` aus. Das An- und Abmelden von Listenern und alle weiteren Events werden komplett von der Superklasse `JComponent` übernommen.

7.4.2.4 Introspektion

Durch Einhaltung des für JavaBeans gültigen Codemusters (vgl. Kapitel 4) erfolgt die Introspektion automatisch. Es wird also keine eigene BeanInfo-Klasse erzeugt.

7.4.3 Der HTree im Kontext

Der `HTree` ist eine eigenständige Komponente. Er kann also beliebig als Baustein eingesetzt und wiederverwendet werden.

Vordergründlich wird die horizontale Baumkomponente als Truppenbaum in dem Projekt JOCCIS eingesetzt. In diesem Zusammenhang wird ein `MouseListener` angemeldet, der ein Kontextdialog öffnet, wenn die rechte Maustaste über einem Knoten gedrückt wird (demonstriert das EventHandling der JavaBean nach außen). Folgende Abbildung zeigt den horizontalen Truppenbaum in JOCCIS.

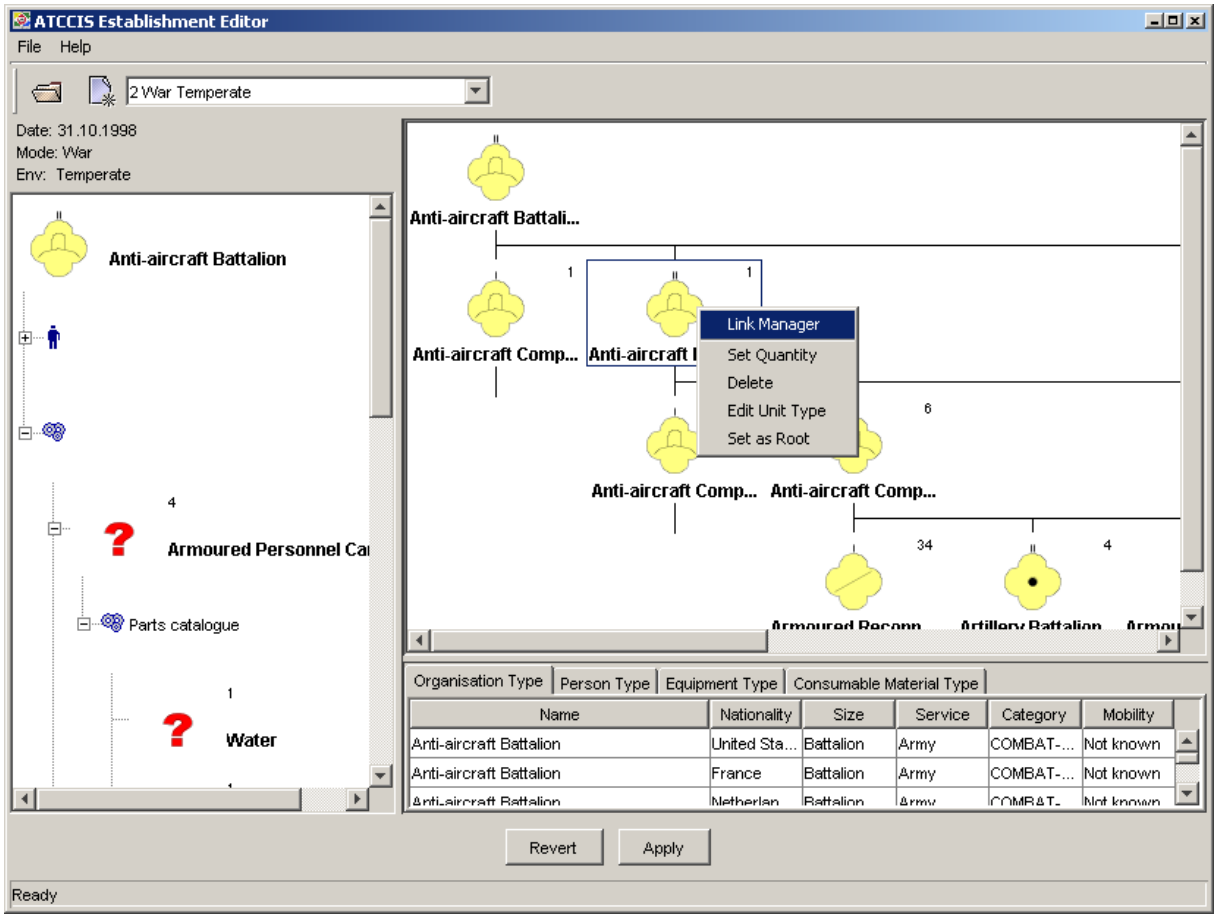


Abbildung 7-6: Horizontaler Truppenbaum in JOCCIS (rechte Maustaste wurde auf einem Knoten geklickt).

Des Weiteren wurde der `HTree` zu Testzwecken als Microsoft Explorer „Ersatz“ verwendet. Die einzelnen Knoten des `TreeModel`'s repräsentieren Dateien oder Ordner. Als Kontextdialog (drücken der rechten Maustaste auf einem Knoten) wurden die Explorer-typischen Funktionen, wie Öffnen, Ausschneiden, Kopieren usw. implementiert (siehe folgende Abbildung).

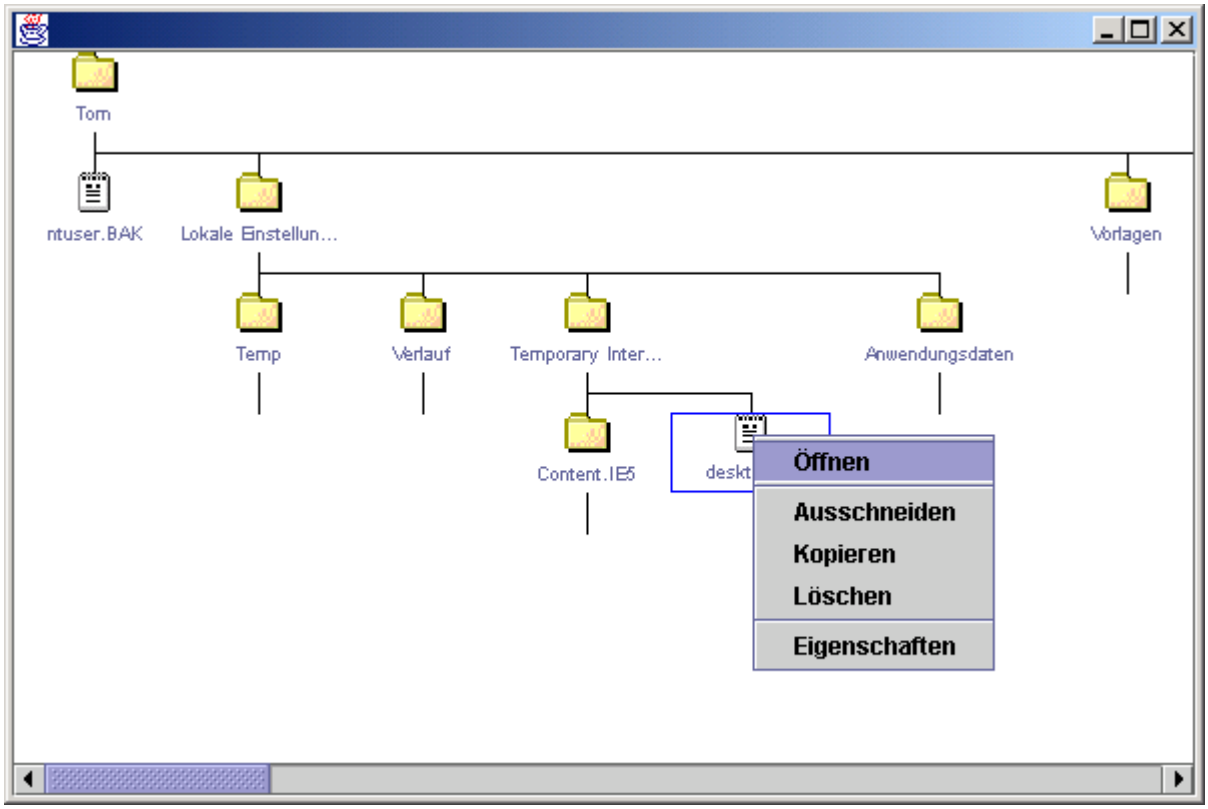


Abbildung 7-7: HTree im Einsatz als Datei-Explorer

Es wurde erreicht, dass der Java `JTree` durch Änderung des Datentyps (aus: `JTree tree = new JTree( model )`, wird: `HTree tree = new HTree( model )`) zu einem horizontalen Baum wird. Benutzt man zur Symboldarstellung fertige Bitmaps (anstelle von SVGs), ist die Geschwindigkeit zur Darstellung des Baumes identisch mit der Geschwindigkeit des `JTree`'s.

Kapitel 8

SCHNITTSTELLE ZUR AUßENWELT VON JAVA

Sinnvollerweise soll das Rad ja nicht für jede Art von Fahrzeugen neu Erfunden werden. Innerhalb der Computerwelt hat diese Weisheit ebenfalls Gültigkeit. Damit man sich also JavaBeans innerhalb anderer Programmiersprachen und Anwendungen zu nutzen machen kann, wurde von Sun eine Schnittstelle geschaffen. Hierbei handelt es sich um die CAS COM Bridge[13].

Auch Microsoft hat sich für die Windowswelt was ausgedacht und hat zur Wiederverwendung von Komponente das ActiveX eingeführt. Es ermöglicht Softwarekomponenten, in einer vernetzten Umgebung miteinander zu kommunizieren, unabhängig von der Programmiersprache, mit der sie entwikkelt wurden. Damit JavaBeans auch durch ActiveX unterstützt werden, hat die Firma Sun die Java Bridge für ActiveX entwickelt.

Basierend auf der ActiveX Technologie gibt es noch ein weiteres Produkt, welches hier untersucht wird. Es handelt sich um ein kommerzielles Produkt mit dem Namen J-Integra der Firma Intrinsyc [14].

8.1 Java Bridge für ActiveX

ActiveX - von Microsoft Mitte der 90er Jahre mit der Intention entwickelt, einen Standard zu schaffen, und heute von der Open Group verwaltet - basiert auf dem Component Object Model (COM) von Microsoft. Derzeit wird ActiveX überwiegend eingesetzt, um interaktive Elemente für das World Wide Web zu entwickeln, obgleich ActiveX auch für Desktopanwendungen und andere Programme verwendet werden kann. ActiveX-Steuerelemente lassen sich in Webseiten einbetten, um Animationen und andere multimediale Effekte, interaktive Objekte und hoch entwickelte Anwendungen herzustellen. ActiveX ist - wie JavaBean - ein Komponentenmodell.

Durch die Java Bridge für ActiveX lassen sich JavaBeans als ActivX-Komponenten in vielen Anwendungen wie Microsoft Office, Internet Explorer oder Visual Basic benutzen. Voraussetzung dafür ist, dass das Java Plug-In in der Version 1.2.x oder 1.3.x auf dem System installiert ist. Das Java Plug-In wird kostenlos mit dem JDK geliefert. In der neuesten Version von JDK (Version 1.4.1) ist zwar das Java Plug-In enthalten, jedoch wird mit dieser nicht (warum auch immer!) die Java Bridge für ActiveX unterstützt.

8.1.1 Von der JavaBean zu einer ActiveX-Komponente

Wenn also eine passende Umgebung geschaffen ist, bedarf es einiger Schritte, um eine JavaBean als ActiveX-Komponente nutzen zu können. Auch hier liefert das JDK ein kostenloses Werkzeug. Im Folgenden wird die Wandlung von der JavaBean zur ActiveX-Komponenten beschrieben:

Die ActiveX-Komponente benötigt eine Registrierungsdatei, damit das Wirtssystem (in diesem Fall Windows) die benötigten Informationen (z.B. Pfad und Objektname) erhält. (Ist eine JavaBean korrekt registriert, so steht sie innerhalb von Windows sehr vielen Anwendungen zur Verfügung.) Weiterhin wird eine so genannte „TypeLibrary“ benötigt. Die TypeLibrary ist die gewandelte JavaBean zur ActiveX-Komponente. Der Packager von Java (im JDK enthalten) erzeugt aus einer JavaBean, die sich in einem Java-Archiv befinden muss, die Registrierungsdatei und die Datei TypeLibrary (siehe Abbildung 8-1).

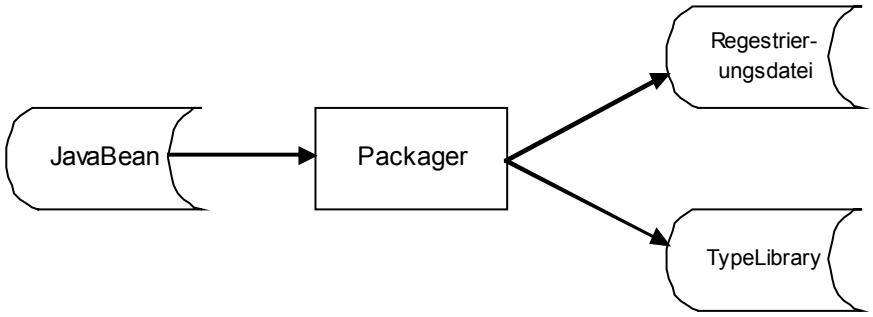


Abbildung 8-1: Funktion des Packager

Um den Packager nutzen zu können, muss man sich erst einmal auf die DOS-Ebene[1] begeben – also eine MSDOS Eingabeaufforderung[2] öffnen. Anschließend wechselt man in das Java Plug-In Installationsverzeichnis (Standardmäßig: `c:\Programme\javasoft\jre\1.3.1`). Von hier aus startet man den Packager mit dem Befehl:

```
bin\java.exe -cp lib\rt.jar;lib\swingall.jar;lib\jaws.jar sun.beans.ole.Packager
```

Wenn Java richtig installiert ist, erscheint nun der Packager wie folgt.

[1] Betriebssystem mit Befehlszeilen Schnittstelle.

[2] Eine Shellumgebung, die auf einer Befehlszeilenaufforderung basiert, wodurch Benutzer die Möglichkeit haben, mit MS-DOS oder einem Betriebssystem zu arbeiten, das MS-DOS emuliert.

Abbildung 8-2: Startfenster des Tools: JavaBeans-Bridge für ActiveX Packager

Dieses Tool teilt sich in fünf einzelne Schritte auf. Erst muss man die JAR-Datei angeben, die die JavaBean enthält, welche man als ActiveX-Komponente nutzen möchte. In Schritt 2 wird eine komplette Liste mit allen enthaltenen JavaBeans angezeigt, denn ein Java-Archiv kann mehrere JavaBeans enthalten. Man wählt nun aus dieser Liste die gewünschte JavaBean aus. In Schritt 3 gibt man den gewünschten Namen und in Schritt 4 noch den gewünschten Pfad der ActiveX-Komponente an. Schritt 5 fragt nur noch ab, ob nun die Registrierungsdatei und die TypeLibrary erstellt und gleichzeitig in Microsoft Windows registriert werden soll. Hat man alle fünf Schritte durchlaufen beendet sich der Packager und die JavaBean steht als ActiveX-Komponente in vielen verschiedenen Anwendungen zur Verfügung. Dies können sowohl visuelle als auch nicht-visuelle Beans sein. Wird Beispielsweise eine visuelle JavaBean innerhalb von Visual Basic durch die ActiveX Bridge importiert, so lässt sie sich in derselben Weise benutzen, wie die Standardsteuerelemente von Visual Basic.

Die Bridge für ActiveX ist nur eine Brücke in eine Richtung. Es können mit ihr also keine ActiveX-Komponenten in Java Applikation Builder importiert werden, jedoch gibt es dafür auch Lösungen (z.B. das JACOB, eine Java-COM Bridge).

Ist eine Komponente in Windows registriert, bleibt sie auch nach einem Neustart registriert. Manchmal kann es erforderlich sein, die Registrierung einer Komponente wieder rückgängig zu machen. Dazu bietet JDK ein Tool welches von der DOS-Ebene mit `UnregBean [Paramter] <ProgID>` gestartet wird und den Eintrag in der Windows Registrierungsdatei löscht.

8.1.2 Schlussbetrachtung

Durch den Packager wird aus einer JavaBean eine vollwertige ActiveX-Komponente. Das bedeutet auch, dass sie über das Internet nutzbar ist. Jedoch gelten auch hier für jeden Internet Client die gleichen Bedingungen wie auf der lokalen Maschine – es muss das Java Plug-In in der Version 1.2.x oder

1.3.x installiert sein. In der neusten Version von JDK (Version 1.4.x) wird die ActiveX Bridge aus Gründen, die ich nur raten kann, nicht mehr unterstützt. Es gibt über die Java Bridge für ActiveX ein Fehlerforum[12], in dem Entwickler ihre Erfahrungen veröffentlichen und in dem für oder gegen die ActiveX Bridge gestimmt werden kann. Vermutlich wurde in der Version 1.4.1 gegen die weitere Unterstützung der ActiveX Bridge gestimmt. Jedoch soll in den neueren Versionen diese Brücke wieder unterstützt werden. Dies belegt das folgende Zitat von Jim Melvin, entnommen aus dem Fehlerforum:

> *A new, high quality Active-X Bridge is being implemented on the 1.4.x codebase for the Java 1.4.2 release due to ship in Spring 2003.*
>
> *Jim Melvin*
> *Java Software*
> *Sun Microsystems*

Ein Bedarf an der Schnittstelle zur „Außenwelt“ von Java ist also durchaus vorhanden. So wirklich ausgereift scheint aber die Java Bridge für ActiveX in der aktuellen Version noch nicht zu sein. Es bleibt also abzuwarten, ob in der neuen JDK Version 1.4.x diese Bridge wieder implementiert ist und wie stabil sie funktioniert. Aber wie sieht es mit Alternativen aus? Sun Microsystems bietet ja noch die CAS COM Bridge an. Ob dies aber eine wirkliche Alternative zur ActiveX Bridge ist, wird in dem folgenden Abschnitt untersucht.

8.2 J2EE CAS COM Bridge

Die Client Access Services (CAS) COM Bridge stellt einen Interprozess COM-Server dar, der Zugriffe auf Java-Objekte ermöglicht. Die Bridge unterstützt COM-Komponenten[3] bzw. -Anwendungen bei dem Zugriff auf lokale Java-Bibliotheken. Es können somit Java-Klassen genauso instantiiert werden, wie JavaBeans.

Die CAS COM Bridge gibt es als kostenlosen download[13] bei der Firma Sun. Das Entwiklungstool beinhaltet die benötigten Libraries, eine Dokumentation, Tools und Beispielanwendungen.

Windows-Applikationen, welche die CAS COM Bridge benutzen, greifen durch die OLE-Automation Technik auf Java-Objekte zu. Im Zusammenhang mit OLE bezeichnet „Automation“ (früher auch unter dem Begriff „OLE-Automation“ bekannt) eine von Microsoft entwickelte Technologie, die es einer Anwendung ermöglicht, Objekte und deren Eigenschaften so darzulegen, dass die Objekte von

[3] COM-Komponenten sind für den Einsatz unter Microsoft Windows-Plattformen konzipiert und können in einer Vielzahl von Programmiersprachen entwickelt werden. Das Entfernen von COM-Komponenten ist während der Laufzeit möglich, ohne dass dazu das Programm erneut kompiliert werden muss. COM stellt die Grundlage für die Spezifikationen OLE (Object Linking and Embedding), ActiveX und DirectX dar.

einer anderen Anwendung genutzt werden können. Beispielsweise kann ein Tabellenkalkulationsprogramm einem Textverarbeitungsprogramm den Zugriff auf ein Diagramm gewähren. Das Textverarbeitungsprogramm kann dann das Diagramm anzeigen und verändern. Die Anwendung, die das Objekt für die Nutzung darlegt, wird als „Server“ bezeichnet. Analog dazu wird die Anwendung, die das Objekt ändert, als „Client“ bezeichnet. Die Automation kann dabei sowohl lokal auf dem eigenen Computer erfolgen als auch in Verbindung mit entfernten Systemen (z.B. auf Computern, die über ein Netzwerk miteinander verbunden sind). Die Automation über OLE ist hauptsächlich für den Einsatz unter Hochsprachen wie Microsoft Visual Basic und Microsoft Visual C++ konzipiert.

8.2.1 Aufbau der CAS COM Bridge

Bevor eine COM-Anwendung via CAS COM Bridge auf Java-Objekte zugreifen kann, startet sie die Java Virtual Machine (JVM). Während der kompletten Ausführungszeit einer Java-Methode wird aus der COM-Anwendung eine echte Java-Anwendung. Dies ermöglicht ihr den Zugriff auf Java in der gleichen Weise, wie Java-Anwendungen dies tun. Somit sind auch Enterprise JavaBeans[4] (EJB) und RMI[5] nutzbar.

Sämtliche Features der CAS COM Bridge sind in drei Ebenen angesiedelt (siehe Abbildung 8-3).

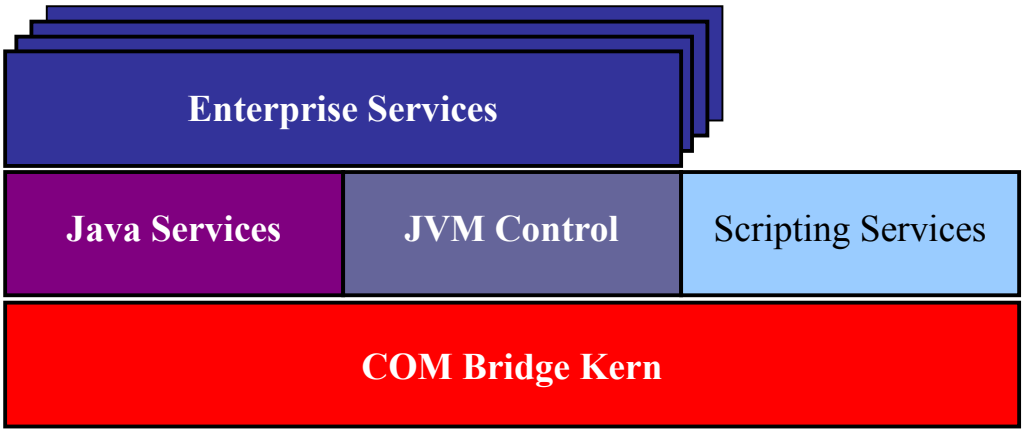


Abbildung 8-3: Funktionsebenen der CAS COM Bridge

Die Basisebene ist die COM Bridge selbst. Sie ermöglicht den Zugriff auf die grundlegenden Funktionen der JVM. Die Service Module auf der zweiten Ebene stellen vereinfachte Methoden für alle üblichen Aufgaben der JVM zur Verfügung. Die Aufgaben umfassen Dinge, wie Klassensuche, Objekte instantiieren oder das „Typecasting“. Die oberste Schicht ist eine Sammlung von Enterprise Services. Sie bietet eine einheitliche Schnittstelle zu einer Vielzahl von gängigen Applikation Servern.

[4] Eine Anwendungsprogrammierschnittstelle (API) zur Erweiterung des Komponentenmodells JavaBeans auf plattformübergreifende Serveranwendungen, die auf den unterschiedlichen Systemen einer Unternehmensumgebung ausgeführt werden können.

8.2.2 COM Bridge Kern

Der COM Bridge Kern besteht aus drei COM-Klassen – Java Virtual Machine Starter (JVMStarter), Java Virtual Machine (JVM) und JavaProxy. Von diesen Dreien kann nur der JVMStarter direkt instantiiert werden. Der JVMStarter lädt die JVM und liefert als Ergebnis ein Objekt der gestarteten JVM. Vorher können noch Eigenschaften wie beispielsweise der zu benutzende Klassepfad der JVM gesetzt werden.

Die Java Virtual Machine stellt Methoden für die folgenden Aufgaben zur Verfügung. Klassensuche, Generierung von Java-Objekten, Zugriff auf statische Klassen und das „Typecasting“. (Diese Methoden werden nochmals vereinfacht von den Service Modulen der CAS COM Bridge zur Verfügung gestellt).

Generierte Objekte werden durch so genannte JavaProxies repräsentiert. JavaProxies sind COM-Objekte, welche eine Referenz auf die zugrunde liegenden Java-Objekte beinhalten und somit den Zugriff auf die entsprechenden „public“ Methoden mit COM ermöglichen. Für die Parameterübergabe müssen die Datentypen konvertiert werden. Folgende Tabelle beinhaltet die Datentyp Konvertierung von Java zu COM.

Java Typ	COM (variant) Typ	Visual Basic Typ	Anmerkung
byte	VT_UI1	Byte	
short	VT_I2	Integer	
int	VT_I4	Long	
Long	VT_DECIMAL	Decimal	
Char	VT_I4	Long	
Float	VT_R4	Single	
Double	VT_R8	Double	
Boolean	VT_BOOLEAN	Boolean	
String	VT_BSTR	String	
Object	VT_DISPATCH	Object	Java-Objekt wird zu COM Java-Proxy gewandelt.
Byte	VT_UI1	Byte	
Short	VT_I2	Integer	
Integer	VT_I4	Long	
Long	VT_DECIMAL	Decimal	
Character	VT_I4	Long	
Float	VT_R4	Single	

[5] Rechnerübergreifendes Objekt-zu-Objekt Kommunikationskonzept in Java.

Double	VT_R8	Double	
Boolean	VT_BOOLEAN	Boolean	

Tabelle 8-1: Java – COM Datentyp „mapping“

8.2.3 Service Module

Die Service Module beinhalten Methoden für den vereinfachten Zugriff auf Java Objekte, die Anbindung von Visual Basic Script[6] (VBScript) und die Unterstützung von Applikationsservern in Verbindung mit Enterprise JavaBeans.

In den Java Services sind die grundlegenden Funktionen der JVM nochmals realisiert. Es erscheint zwar widersprüchlich, die gleiche Funktionalität - Klassensuche, -instantiierung und –casting - nochmals auf einer anderen Ebene anzubieten, jedoch reichen die hier angebotenen Funktionen für den Zugriff auf die Java Standardumgebung vollkommen aus und vereinfachen die Benutzung der CAS COM Bridge innerhalb der COM-Applikation.

Das Scripting Service Objekt bietet Dienste für die Benutzung der CAS COM Bridge für VBScript an. Hauptsächlich dient dieser Service den Active Server Pages[7] (ASP), welche VBScript für ihre Funktionalität benutzen.

Die Enterprise Services vereinfachen den Zugriff auf EJBs. Obwohl die Enterprise Services nicht für den Zugriff auf EJBs benötigt werden, vereinfachen sie doch die Aufgabe. Der Programmierer braucht den Umgang mit EJB – Serveranbindung, Home-Interface finden und Typecasting - nicht extra zu codieren. Standardmäßig werden Module für die folgenden Applikationsserver angeboten:

- iPlanet IAS 6.0 SP2 (IasServices) der Firma Sun Microsystems.
- Sun Microsystems J2EE 1.2.1 + RI der Firma Sun Microsystems.
- SilverStream 3.7 der Firma SilverStream Inc. / Novell.
- WebLogic 5.1 und 6.0 der Firma Bea Systems, Inc.
- WebSphere 3.5 der Firma IBM.

Jedoch kann man auch Serveranbindungen für weitere Applikationsserver über Sun Microssystems beziehen.

[6] Ein Teilbereich der Programmiersprache „Visual Basic for Applications“, der für das Programmieren in einer Webumgebung konzipiert ist. Der Code für Visual Basic Script wird wie bei JavaScript in HTML-Dokumente eingebettet.

[7] Eine Webtechnologie, die von Microsoft entwickelt wurde, um die Entwicklung von Skripten auf der Serverseite zu ermöglichen.

8.2.4 Tools

Zur Hilfe bei der Entwicklung mit Grafischen Entwicklungstools (z.B. Visual Basic) gibt es so genannte „Type Libraries“. Sie zeigen die Methoden Namen und Eigenschaften der einzelnen Java-Klassen bzw. JavaBeans an. Zum erstellen der Type Libraries gibt es das Tool `GenTypeLib`. Dies erstellt aus einer kompilierten Java-Klasse das Type Library und trägt es in die Windows Registry-Datei ein. Beispielsweise wird aus der Java-Klasse `java.util.Date` die Type Library Datei `java_util_Date` erstellt. Entwickler können nun Variable vom Datentypen `java_util_Date` festlegen. Um Type Libraries auch auf anderen Computern nutzen zu können, reicht es aus, die gewünschten Dateien der Type Libraries auf den Computer zu kopieren und mit einem Tool - `RegTypeLib` - in die dortige Windows Registry-Datei einzutragen.

8.2.5 Schlussbetrachtung

Die CAS COM Bridge ist ein sehr umgängliches Werkzeug. Die Benutzung von Java Standardklassen sowie JavaBeans innerhalb einer COM-Applikation läuft sehr stabil es bedarf wenig Aufwand zur Objektinstantiierung, wie das folgende Beispiel zeigt.

Beispiel: Benutzung der Java Standardklasse `java.util.Date` in Visual Basic.

```
Private Sub Form_Load() {
  'Variable theDate wird vom Type java_util_Date definiert
  Dim theDate as java_util_Date
  'Objekt instantiieren
  Set theDate = JavaNew("java.util.Date")
  'Ausgabe des Datums in der Messagebox
  MsgBox theDate.toString
}
```

Bevor man dieses kleine Beispielprogramm zum Laufen bekommt, müssen ein paar grundsätzliche Schritte erledigt werden. Die CAS COM Bridge muss auf dem gewünschten Arbeitsplatz installiert sein und in der Visual Basic Umgebung muss auf die J2EECAS Services 1.0 Type Library (Service Modul der CAS COM Bridge) und auf die Type Library java_util_Date verwiesen werden (Die nötigen Schritte sind genau in der Dokumentation zur CAS COM Bridge beschrieben). Das Ergebnis dieses kleinen Beispielprogramms sieht man in Abbildung 8-4.

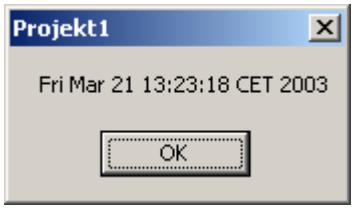


Abbildung 8-4: Ergebnis des obigen Beispielprogrammes in Visual Basic

Als Nachteil kann man die Rückwärtskommunikation bewerten. Leider funktioniert die CAS COM Bridge nur in eine Richtung. Dies bedeutet, dass ich aus COM-Anwendungen beliebig Java-Objekte instantiieren und benutzen kann, jedoch bietet es keine Möglichkeiten, von den Java-Objekten mit einem „callback“[8] wieder auf die COM-Anwendung zugreifen zu können. Deswegen ist es auch das Listener-Konzept von Java zur COM-Anwendung nicht möglich. Dieses Manko lässt sich jedoch entweder durch zusätzliche Benutzung des Open Source Projektes JaCOB von Dan Adler oder durch das Projekt J-Integra der Firma Intrinsyc Software beheben.

8.3 J-Integra

J-Integra ist eine vollwertige Java-Com-Bridge. Dieses Tool ermöglicht den Zugriff auf COM-Komponenten in der Weise, als wären es Java-Objekte und es erlaubt den Umgang mit Java-Objekten, als wären es COM-Komponenten.

J-Integra ist ein kommerzielles Produkt. Es gibt jedoch bei der Firma Intrinsyc eine kostenlose Testversion mit einer Laufzeit von 60 Tagen [14].

Das Hauptaugenmerk der Untersuchungen wurde auf das Listener-Konzept gerichtet. D.h., von einer COM-Anwendung werden Java-Objekte instantiiert, welche Ereignisse auslösen, die dann wiederum in der COM-Anwendung registriert werden sollen. Erst wird aber das ganze Paket von J-Integra vorgestellt.

8.3.1 Aufbau und Leistungsvermögen von JIntegra

J-Integra ist so konzeptioniert, dass die Kommunikation zwischen Java- und COM-Anwendungen von jeder Plattform aus erfolgen kann. Dabei spielt auch der lokale Standort der verschiedenen Applikationen keine Rolle (siehe Abbildung 8-5). Auf der untersten Ebene benutzt die Laufzeitumgebung von J-Integra die Standard Java-Networking Klassen. Diese basieren auf TCP/IP. Darauf aufgebaut sind RPC’s und darauf wiederum ist DCOM aufgebaut. Vereinfacht gesagt: Für die Java - COM Kommunikation wird DCOM über TCP/IP benutzt.

Durch den Einsatz von J-Integra kann ein Java-Programmierer COM-Komponenten wie Java-Objekte benutzen. COM-Eigenschaften, -Methoden und -Events werden in gleicher Weise wie Java-Eigenschaften, -Methoden und -Events benutzt. Wenn also eine COM-Komponente Eventfähig ist, kann man in Java einen Listener für den Event anmelden, welcher dann beim Eintritt des Ereignisses benachrichtigt wird.

[8] Aufrufendes Objekt gibt eine Referenz dem aufgerufenem Objekt eine Referenz von sich selber mit, damit das aufgerufene Objekt einen „Rückaufruf“ starten kann.

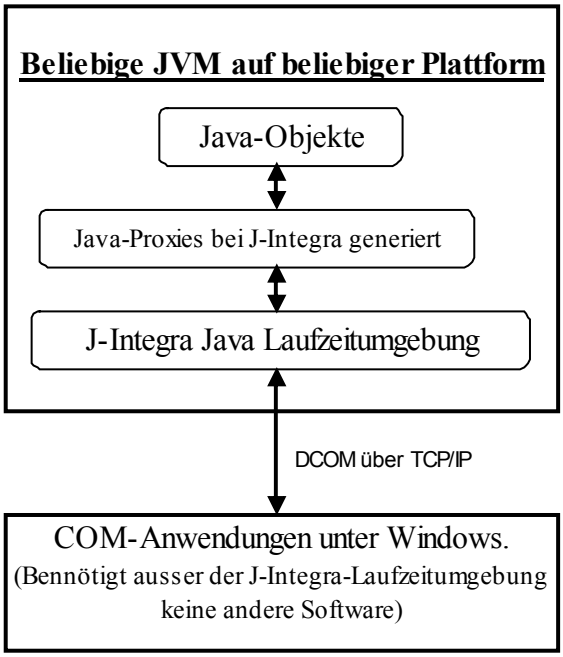


Abbildung 8-5: Konzeptioneller Aufbau von J-Integra

Da J-Integra eine bidirektionale Java-Com-Bridge ist, kann der Programmierer von COM-Anwendungen (in VisualBasic, C++, VBA, usw.) auch auf Java-Objekte bzw. -Komponenten zugreifen. Dabei stehen dem Entwickler alle öffentlichen (public) Methoden und Variablen von Java-Objekten zur Verfügung. Auch hier ist es möglich, Events abzufangen. Die COM-Komponente kann sich als Listener an ein Java-Objekt anmelden und wird beim Eintreten eines Ereignisses benachrichtigt.

Durch die Komplexität von J-Integra ist es unmöglich, das ganze Können von J-Integra zu Beschreiben, ohne in Details einzusteigen. Da jedoch der Zugriff von COM-Anwendungen auf Java-Objekte im Vordergrund dieser Untersuchungen steht, wird hier auf dieses Thema im folgenden Abschnitt genauer eingegangen.

8.3.2 Zugriff von COM-Anwendungen auf Java-Objekte

J-Integra erlaubt COM-unterstützenden Programmiersprachen den Zugriff auf Java-Objekte so, als wären sie COM-Komponenten. Um dies zu ermöglichen, muss auf der Seite des COM-Client-PC's eine Referenz auf eine JVM registriert werden. Dabei unterscheidet J-Integra zwischen den folgenden drei Möglichkeiten:

(1) **DCOM Modus**: Hier können sich die JVM und die Java-Objekte auf einem beliebigen Server befinden (siehe folgende Abbildung). Beim Registrieren der JVM auf dem Windows-Client-PC werden der Hostname („localhost" bei lokalen Anwendungen) und die Portnummer angegeben.

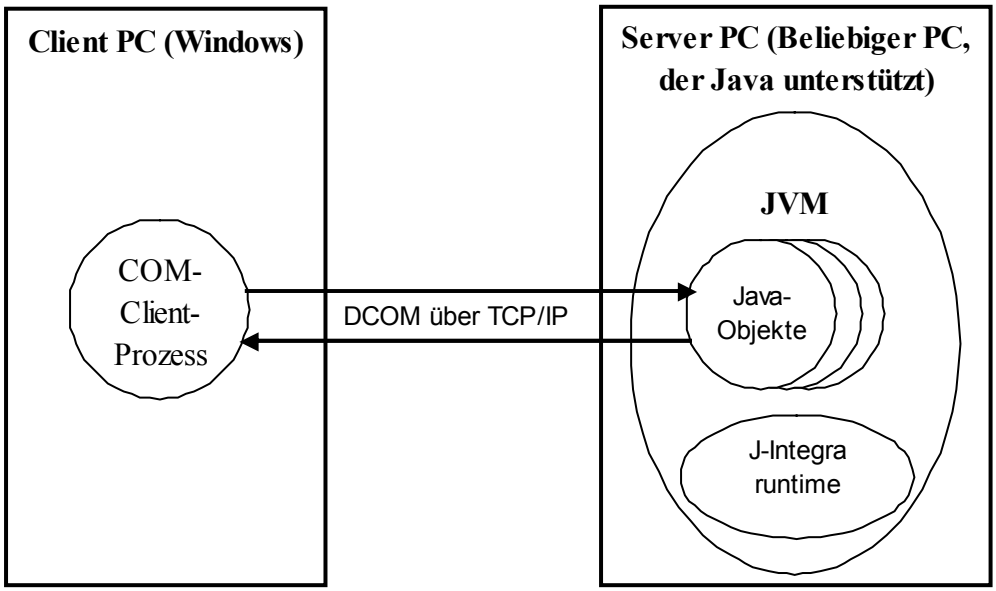


Abbildung 8-6:Kommunikation einer COM-Anwendung mit Java-Objekten im DCOM-Modus

Auf der Serverseite muss die JVM vor dem Zugriff auf die Java-Objekte gestartet werden. Die JVM muss `com.linar.jintegra.Jvm.register("<jvm id>")` in der Main-Methode aufrufen, wobei <jvm id> die ID ist, welche vom Nutzer bei der Registrierung der JVM vergeben wird. Die JVM wird gestartet, indem sie ein Java-Programm ausführt. Das Java-Programm könnte wie folgt aussehen:

```
public class Main {
  public static void main(String[] args) {
    //JVM läuft und ist bereit, remote Anfragen zu empfangen.
    com.linar.jintegra.Jvm.register("myJVM");
    //Thread "schläft" für eine Stunde
    Thread.sleep(600000);
  }
}
```

Die JVM muss zusätzlich mit dem Parameter `JINTEGRA_DCOM_PORT=<“port nr“>` gestartet werden (z.B.: `java -DJINTEGRA_DCOM_PORT=1234 Main`).

(2) **Native Modus (out of process)**: Dieser Modus funktioniert nicht auf verteilten Systemen. Dies bedeutet, dass die JVM und die Java-Objekte sich auf dem selben Rechner befinden müssen (siehe folgende Abbildung). Auch hier gilt (wie beim DCOM Modus), dass die JVM vorher gestartet werden und dann den Aufruf `com.linar.jintegra.Jvm.register("<jvm id>")` tätigen muss. Zusätzlich wird hier die JVM mit dem Parameter JINTEGRA_NATIVE_MODE gestartet (z.B.: `java -DJINTEGRA_NATIVE_MODE Main`).

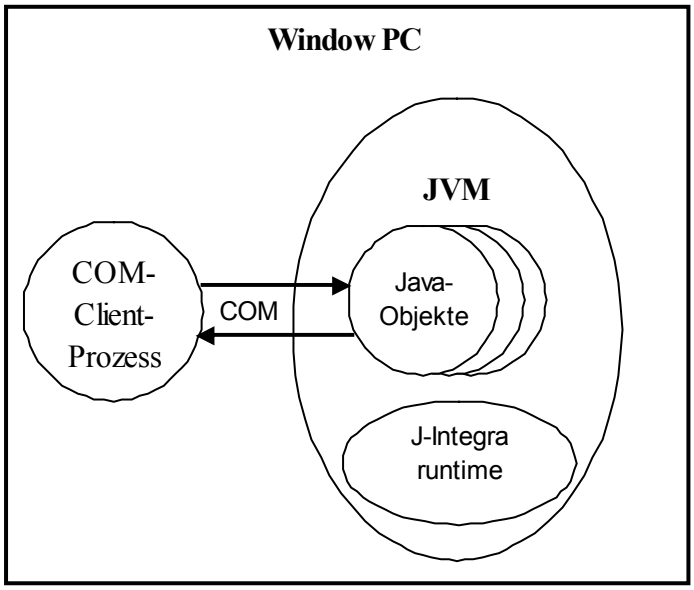


Abbildung 8-7: Kommunikation einer COM-Anwendung mit Java-Objekten im Native Modus (out of process)

(3) **Native Modus (in process)**: Dieser Modus erlaubt es, Java-Objekte im selben Prozess wie die COM-Anwendung laufen zu lassen. Das setzt voraus, dass sich die JVM und die Java-Objekte auf demselben Computer wie die COM-Anwendung befinden (siehe folgende Abbildung).

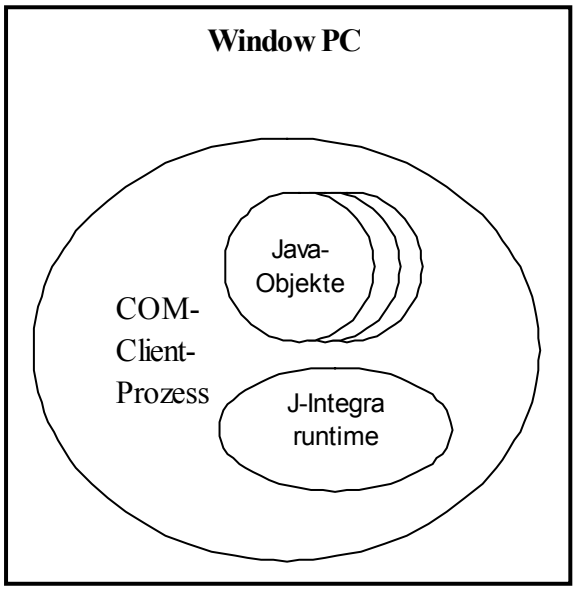


Abbildung 8-8: Kommunikation einer COM-Anwendung mit Java-Objekten im Native Modus (in process)

In diesem Modus muss die JVM nicht vorher manuell gestartet werden, sondern sie wird automatisch beim Instantiieren von Java-Objekten gestartet.

Desweiteren unterscheidet J-Integra noch zwischen zwei Möglichkeiten der Bindung von Java-Objekten. Java-Objekte können vor dem Erstellen einer COM-Anwendung an diese gebunden werden (early Binding) oder können zur Laufzeit der COM-Anwendung (late Binding) registriet und benutzt werden. Bei der Methode der frühen Bindung müssen die im folgenden Beispiel (nächster Abschnitt)

kurz erklärten Schritte der Type-Library[9]-Erstellung und -Registrierung durchgeführt werden. Im Falle der späten Bindung ist dies nicht erforderlich. Der hauptsächliche Nachteil der späten Bindung liegt darin, dass **COM-Anwendungen nicht als Listener an Java-Anwendungen registriert** werden können, da es zur Entwicklungszeit die COM-Anwendung nicht die benötigten Typen der Java-Objekte kennt.

8.3.3 Beispielanwendung

Im folgenden Beispiel wird der Ablauf zur Benutzung von Java-Objekten aus einer COM-Anwendung in Visual Basic erklärt. Der Zugriff auf die JVM erfolgt im Native Modus in process (siehe Abschnitt 8.3.2 (3)).

Bevor auf Java-Klassen zugegriffen werden kann, müssen einige Vorbereitungen getroffen werden. Alle benötigten Schritte werden ausführlich in der zu J-Integra mitgelieferten Dokumentation beschrieben und werden hier nur ganz kurz aufgeführt.

- Umgebung konfigurieren (CLASSPATH und PATH richtig setzen).
- Java-Klassen analysieren, Erstellen der Java-Wrapper-Klassen[10] und der passenden IDL[11]-Datei mit Hilfe des J-Integra-Tools „java2com“.
- Kompilieren der Wrapper-Klassen und der IDL-Datei und Registrieren der erstellten Type-Library.
- Die JVM richtig registrieren (Native Mode, in process). Dazu kann das J-Integra-Tool „regjvm“ benutzt werden.

Wenn all diese Schritte durchgeführt sind, sollte die Java-COM-Anwendung lauffähig sein.

In der Beispielanwendung handelt es sich um ein Fenster (in Visual Basic programmiert, siehe Abbildung 8-10), dass einen Text anzeigt. Um den Text zu editieren wird ein Dialog-Fenster (in Java programmiert, siehe Abbildung 8-9) geöffnet. Die COM-Anwendung erlaubt das Ab- und Anmelden von Listenern bei der Java-Anwendung.

Die Java-Klasse (`DialogBox`) beinhaltet die folgenden Methoden:

`public void setName( String n )`: Setzt den Namen in der Anzeige.

`public String getName()`: Liefert den eingegebenen bzw. geänderten Namen.

---

[9] Beschreibt Schnittstellen und Klassen in einem COM-Server / einer COM-Umgebung

[10] „Eingepackte“ Java-Klassen. In diesem Falle werden die Java-Klassen DCOM-konform umhüllt.

[11] Beschreibt die Schnittstelle der aufzurufenden Programme bzw. Objekte.

`public String getOK()`: Liefert den Text des O.k.-Buttons.

`public String getCancel()`: Liefert den text des Cancal-Buttons.

`public void show( boolean b )`: Öffnet bzw. schließt das Dialog-Fenster.

An den beiden Schaltflächen (`O.k.` und `Cancel`) ist ein Listener angemeldet. Beim Kicken auf einen Button, wird bei allen an `DialogBox` angemeldeten `ActionListener` die Methode `actionPerformed( ActionEvent e )` aufgerufen.

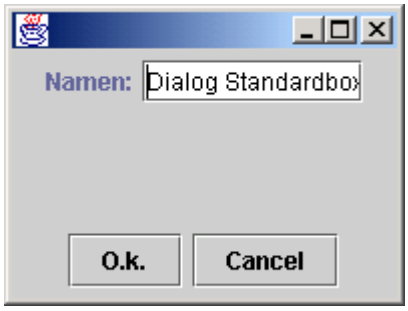


Abbildung 8-9: DialogBox (Java-Anwendung)

Die COM-Anwendung ist in Visual Basic erstellt. Bei einem Mausklick auf den Text, wird das Java-Dialog-Fenster zum Editieren des Textes geöffnet. Ist die COM-Anwendung als Listener bei dem Dialog-Fenster angemeldet (durch Drücken des Reg.Listener-Buttons), wird beim Drücken des o.k.-Buttons der Text („Dialog Standardbox“) geändert und das Dialog-Fensters geschlossen. Wird der Cancel-Button gedrückt, bleibt der Text unverändert und das Dialog-Fenster wird geschlossen.

Abbildung 8-10: Visual Basic Anwendung (steht in Interaktion mit einer Java-Anwendung)

Die COM-Anwendung beinhaltet die folgende Klasse in Visual Basic (mit dem Namen: `Class1`):

```
' Wichtig ist das Schlüsselwort "WithEvents". Dies zeigt an, dass auf Events der
' Java-Klasse reagiert wird

Dim WithEvents dia As DialogBox

Dim isreg As Boolean

' Dient zum Anmelden der COM-Anwendung als Listener. Der Parameter evs ist eine
' Instanz der Java-Klasse Dialog-Box
```

```
Public Sub addEventSource(ByVal evs As DialogBox)
  isreg = True
  Form1.Label2.Caption = "Listener registriert"
  Set dia = evs
End Sub

' Meldet die COM-Anwendung als Listener bei der Java-Anwendung ab.
Public Sub removeEventSource(ByVal evs As DialogBox)
  If (dia = evs) Then
    Set dia = Nothing
    Form1.Label2.Caption = "kein Listener registriert"
    isreg = False
  End If
End Sub

' Wird in der Java-Anwendung die Methode actionPerformed( ActiobEvent e ) auf-
' gerufen, wird diese Methode in der COM-Anwendung ausgeführt.
Public Sub dia_actionPerformed(ByVal p1 As JavaAwtEventActionEvent)
  but = p1.getActionCommand
  If but = dia.getOk Then
    Form1.Label1.Caption = dia.getName
  Else
    MsgBox "Edit wurde abgebrochen !"
  End If
  dia.show(false)
End Sub
```

Die Instantiierung der Java-Klasse (`DialogBox`) und das Instantiieren der Visual Basic-Klasse (`Class1`) geschieht beim Öffnen (`on_Load`) des Formulares (Fensters). Das Anzeigen des Dialog-Fensters wird in der Methode `Label1_Click()` ausgeführt. Das Registrieren eines Listeners geschieht in der Methode `Command1_Click()` und das Abmelden eines Listeners geschieht in der Methode `Command2_Click()`.

```
Dim btn As DialogBox ' btn als Datentyp festlegen
Dim cl1 As New Class1 ' Class1 wird durch den operator "New" instantiiert.

Private Sub Form_Load()
  Set btn = GetObject("inproc:DialogBox") ' btn wird instantiiert.
  Label1.Caption = btn.getName
End Sub

Private Sub Command1_Click()
  cl1.addEventSource btn ' Hinzufügen von class1 als Listener an btn
End Sub
```

```
Private Sub Command2_Click()
  cl1.removeEventSource btn ' Entfernen von class1 als Listener an btn
End Sub

Private Sub Label1_Click()
  btn.Show(true) ' Anzeigen des Dialog-Fensters
End Sub
```

Wichtig ist bei der Eventsteuerung nur die richtige Namensgebung. Wenn eine beliebige Java-Klasse ein Event `xyEvent` als Ereignis besitzt und beim Eintritt dieses Ereignisses die Methode `xyPerformed( xyEvent e )` bei allen angemeldeten Listenern aufruft, so muss in der Visual Basic-Klasse die Methode zum Aufruf `btn_xyPerformed(ByVal p1 As xyEvent)` heißen.

8.3.4 Schlussbetrachtung

J-Integra ist eine sehr leistungsstarke und vor allem vollwertig bidirektionale Java-COM-Bridge. Für die Untersuchung des vollen Leistungsumfang dieses Tools bedarf es einer separaten Diplomarbeit.

Probleme gab es innerhalb des untersuchten Rahmens nur bei der Registrierung der JVM im Native Modus (in process). Die Beispielanwendung ist nicht auf allen JVMs, die auf dem PC installiert sind, gelaufen. Die Gründe, warum es mit der einen JVM funktionierte und mit einer Anderen nicht, sind unergründlich. Ein weiteres, kleines Manko ist zur Zeit noch die Dokumentation zu J-Integra. Es wird zwar der ganze Leistungsumfang dokumentiert, aber in vielen Bereichen (auch im Bereich des Java-COM-Eventhandlings) ist die Beschreibung sehr dürftig. Vieles kann man jedoch über Foren (z.B.: YahooGropus unter http://groups.yahoo.com/group/jintegra/) im Internet recherchieren und erfahren.

Die Vorteile wie Benutzerfreundlichkeit und Stabilität überwiegen deutlich die Nachteile. J-Integra liefert genau das, worauf es bei den Untersuchungen angekommen ist (Eventhandling und Wechseln zwischen Java- und COM-Anwendung zur Laufzeit). Der vermeindliche Nachteil, dass dieses Tool Lizenzpflichtig ist, wird durch den guten Service und die Produktqualität egalisiert.

ABBILDUNGSVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

API	Application Programming Interface	Schnittstellen(-beschreibung) zur Programmierung von Anwendungen.
ASP	Active Server Pages	Zu Deutsch: aktive Seiten auf dem Server. Eine Webtechnologie, die von Microsoft entwickelt wurde, um die Entwicklung von Skripten auf der Serverseite zu ermöglichen.
AWT	Abstract Window Toolkit	Standardpaket mit Oberflächenkomponenten für Java.
CAD	Computer Assisted Drafting	Computer unterstützte (technische) Zeichnung.
COM	Component Object Model	Allgemeines Objektmodell. Dabei handelt es sich um eine Spezifikation, die die Entwicklung von Softwarekomponenten beschreibt, welche sich in Programme einbauen oder auch in bestehende Programme hinzufügen lassen.
CSS	Cascading Style Sheet	Dokumentvorlage für HTML und XML Dokumente. Enthalten typografische Informationen in Bezug auf das Erscheinungsbild der Seite.
DCOM	Distributed COM	Die Version der COM- Spezifikation von Microsoft, die festlegt, wie die Komponenten mit Windows basierten Netzwerken kommunizieren.
DOM	Document Object Model	Eine vom W3C vorgestellte Technik, um auf die Struktur eines Dokumentes zuzugreifen.
DTD	Document Type Definition	Ein gesondertes Dokument, das formale Definitionen aller Datenelementcodes einer Gruppe von XML- oder SGML-Dokumenten enthält.
EJB	Enterprise JavaBeans	Eine Anwendungsprogrammierschnittstelle (API) zur Erweiterung des Komponentenmodells JavaBeans auf plattformübergreifende Serveranwendungen.
GUI	Graphical User Interface	Grafische Oberfläche, welches als Frontend für den Benutzer dient.
GVT	Graphic Vector Toolkit	„Low Level“ Modul von Batik, welches den SVG DOM Baum in ein zum Rendern passendes Format wandelt.

HTML	Hyper Text Markup Language	Hypertextauszeichnungssprache. Auszeichnungssprache, die für Dokumente im World Wide Web verwendet wird.
IDL	Interface Definition Language	Beschreibt die Schnittstelle der aufzurufenden Programme bzw. Objekte.
JDK	Java Developer's Kit	Kostenloses Softwarewerkzeuge, welches von Sun Microsystems zum Schreiben von Java-Applets oder Anwendungen entwickelt wurden. Das Kit enthält u.a. einen Java-Compiler, Interpreter, Debugger, einen Viewer für Applets und eine Dokumentation.
JVM	Java Virtual Machine	Umgebung, in der Java-Programme laufen. Die JVM ist Software und steht plattformunabhängig zur Verfügung.
MVC	Model View Controller	Bestimmtes Muster, indem das Modell, die graphische Oberfläche und die Steuerung in getrennten Klassen sind.
OLE	Object Linking and Embedding	Verknüpfen und Einbetten von Objekten. Eine Technologie zum Austausch und zur gemeinsamen Nutzung von Daten zwischen verschiedenen Anwendungen.
RMI	Remote Methode Invocation	Entfernter Methodenaufruf. Ein rechnerübergreifendes Objekt-zu-Objekt Kommunikationskonzept in Java.
SAX	Simple API for XML	Stellt für einen Parser von XML-Dateien eine API zur Verfügung, mit der eine Objektorientierte Sprache (z.B. Java oder C++) diese verarbeiten kann.
SGML	Standard Generalized Markup Language	Verallgemeinerte und standardisierte Textauszeichnungssprache. SGML beschreibt ein abstraktes Verfahren zur Bereitstellung plattform- und anwendungsunabhängiger Dokumente, bei dem Formatierung, Indizierung und verknüpfte Informationen erhalten bleiben.
SVG	Scalable Vector Graphics	Skalierbare Vektorgrafik. Ist eine XML Anwendung.
UML	Unified Modeling Language	Vereinheitlichte Modellierungssprache. Sie kann zur Bestimmung, Herstellung und Dokumentation von Software- und Nichtsoftwaresystemen (z.B. geschäftlichen Modellen) genutzt werden.
W3C	World Wide Web Consortium	Zusammenschluss von Interessenten des World Wide Web (HP, IBM, Siemens und viele Andere).

XML	Extensible Markup Language	Erweiterbare Textauszeichnungssprache. Eine reduzierte Variante von SGML, die als Nachfolgerin von HTML für Internetanwendungen entwikkelt wurde.

QUELLEN UND LITERATURVERWEISE

[1] Webseite zu „W3C Math Home“, <http://www.w3.org/Math/>

[2] Webseite zu „Web 3D Consotium - X3D“, <http://www.web3d.org/x3d.html>

[3] Webseite zu „W3C Synchronized Multimedia“, <http://www.w3.org/AudioVideo/>

[4] Webseite zu „W3C Scalable Vector Graphics“, <http://www.w3.org/TR/SVG/>

[5] Webseite zu Batik, <http://xml.apache.org/batik/>

[6] Vincent J. Hardy. *JAVA 2D API Graphics*. Prentice Hall, Januar 2000

[7] Dr. Satyaraj Pantham, *Pure JFC Swing, A Code- Intensive Premium Reference*, Techmedia, erste Auflage, 1999

[8] J. David Eisenberg, *SVG Essential*, O‘Reilly & Associates, erste Auflage, Februar 2002

[9] Webseite bei Sun, „Swing JDC Chat“, < http://java.sun.com/products/jfc/tsc/articles/chat1/>

[10] Webseite zu Jazz, <http://www.cs.umd.edu/hcil/jazz/>

[11] Sun Microsystems: The JavaBean V1.01 API specification.

[12] Webseite bei Sun. Fehler-Forum zur Java Bridge für ActiveX (nur für registrierte Benutzer), <http://developer.java.sun.com/developer/bugParade/bugs/4616580.html>

[13] Webseite bei Sun. (J2EE CAS) COM Bridge 1.0 early access (nur für registrierte Benutzer), <http://developer.java.sun.com/developer/earlyAccess/j2eecas/download-com-bridge.html>

[14] Webseite zu JIntegra bei Intinsyc, <http://www.intrinsyc.com/index.asp>

KOPIERRECHTE UND LIZENZEN