Diplomarbeit

Entwicklung eines Frameworks für Web-Anwendungen zur Überwachung von Server-Systemen

Tilman Schneider

5. August 2004

Zusammenfassung

Viele Unternehmen betreiben komplexe heterogene Serversysteme. Vor allem bei hausintern entwickelten Server-Applikationen erfolgt die Administration oft auf systemnaher Ebene, etwa durch den Einsatz von Skripten oder durch die direkte Abfrage von Datenbanken. Dieses Vorgehen birgt viele Probleme in sich: Zum einen ist das Fachwissen an einige wenige Mitarbeiter gebunden und zum anderen fehlt oft die Gesamtübersicht.

Management-Applikationen bieten die Möglichkeit, den aktuellen Zustand der Systeme schnell und übersichtlich über eine einheitliche Oberfläche zu erfassen. So kann gerade im Fehlerfall schneller und effizienter reagiert werden. Die Ansicht sollte je nach Szenario und Verantwortlichkeit des Benutzers angepasst werden können, so dass alle relevanten Daten kompakt ersichtlich sind.

Im Rahmen dieser Arbeit entstand ein vielseitiges, modular aufgebautes Framework, mit dessen Hilfe in kurzer Zeit Web-basierte Management-Applikationen erstellt werden können. Das Framework bietet auf der einen Seite Unterstützung für die Instrumentierung einer bestehenden Applikation durch die Java Management Extension (JMX). Auf der anderen Seite ermöglicht es mit Hilfe einer flexiblen Tag-Library die schnelle und einfache Entwicklung von JSP-basierten Web-Anwendungen, die die zuvor instrumentierten Ressourcen auf vielfältige Weise visualisieren und steuern können.

Die Fähigkeiten dieses Frameworks wurden anhand einer prototypischen Web-Anwendung demonstriert, mit der ein für dm-drogerie markt zentrales Server-System überwacht werden kann.

Abstract

Many companies run complex and heterogeneous server systems. Especially with server applications that are developed in-house, the administration is often very low-level, e.g. by executing shell scripts or by directly querying the database. This practice involves many problems: on the one hand the know-how is bound to few employees and on the other hand there is often a lack of overview.

Management applications offer a possibility to show the current state of the systems quickly and clearly arranged on a uniform user interface. In that way it is possible to react faster and more efficient, particularly in the case of an error. There should be a possibility to adjust the view to the scenario and the responsibility of the user, to make all relevant data compactly apparent.

With this thesis a versatile and modular designed framework evolved that allows to create web-based management applications in short time. On the one hand this framework supports instrumenting existing applications using the Java Management Extension (JMX). On the other hand it provides, with the aid of a flexible tag library, the fast and simple development of JSP based web applications that are able to visualize and control prior instrumented ressources in many ways.

The framework's power was demonstrated with a prototypic web application that monitors a server system that is vital for dm-drogerie markt.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt.

Karlsruhe, den 5. August 2004

_____

Tilman Schneider

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

Kapitel 1

Einleitung

Um im täglichen Betrieb von Applikationen eine hohe Verfügbarkeit zu garantieren, ist es essentiell, mögliche Fehlerquellen frühzeitig zu erkennen. Einfache Mittel der Fehlersuche, wie das Durchsuchen von Log-Dateien, das Ausführen von Skripten, die beispielsweise den Status von Prozessen prüfen, oder direkte Datenbankabfragen sind dabei für kleinere Anwendungen durchaus praktikabel. Sie erfordern jedoch viel Fachwissen und sind sehr zeit- und damit kostenintensiv. Der Administrationsaufwand steigt mit der Komplexität einer Anwendung und wird bei großen Systemen schnell unverhältnismäßig groß.

Es bedarf deshalb einer Management-Applikation, die den aktuellen Zustand der Anwendung schnell und übersichtlich über eine einheitliche Oberfläche darstellt. So kann gerade im Fehlerfall schneller und effizienter reagiert werden. Die Ansicht sollte je nach Szenario und Verantwortlichkeit des Benutzers angepasst werden können, so dass alle relevanten Daten kompakt ersichtlich sind.

Als eines der größten europäischen Handelsunternehmen setzt dm-drogerie markt eine heterogene Systemlandschaft mit verschiedenen Systemen ein, zum Beispiel zur Lager- oder Filialverwaltung, ein zentrales SAP R/3-System oder ein Data Warehouse im Terabytebereich. Um Daten zwischen den Systemen auszutauschen, wird das Enterprise-Application-Integration-Tool Ascential Mercator eingesetzt. Da zahlreiche Prozesse vom Funktionieren des Java-basierten Mercatorsystems abhängen, spielt es in der Systemlandschaft von dm-drogerie markt eine zentrale Rolle.

Ziel dieser Diplomarbeit ist die Entwicklung eines Frameworks, mit dessen Hilfe Web-Anwendungen zur Überwachung beliebiger Server-Systeme erstellt werden können. Auf Basis dieses Frameworks soll prototypisch eine Web-Anwendung zur Überwachung des Mercatorsystem entwickelt werden.

Kapitel 2

Grundlagen

In diesem Kapitel sollen die für diese Diplomarbeit wichtigen Technologien und Verfahren erläutert werden.

2.1 Applikations-Management

Die nachfolgenden Erkenntnisse basieren auf dem Artikel „Das Management von Applikationen und Web-Services“ von Helmut Buchenberger und Stefan Trautmann [AppMan].

In vielen Anwendungen können Fehler, die den reibungslosen Ablauf gefährden oder gar behindern, nur sehr schwer gefunden werden.

Management-Funktionen können hier entgegensteuern, indem sie zur frühzeitigen Fehlerlokalisierung und -korrektur verhelfen. Die wohl einfachste und verbreitetste Management-Funktion ist die Berichterstattung über Logging. Bestimmte Ereignisse werden als Meldung persistent, meist in eine Datei, gespeichert.

Bis zu einem gewissen Grad hin, können Logging-Dateien sehr gut genutzt werden, zumal sich viele Management-Applikationen auf das Herausfiltern der relevanten Informationen aus Logging-Dateien verstehen.

Logging-Dateien beziehen sich jedoch immer auf die Vergangenheit und können nicht den aktuellen Status einer Anwendung wiederspiegeln. Sie eignen sich deshalb zwar zur Fehlersuche, nicht aber zur Steuerung.

Nach [AppMan] ist Managen „die Fähigkeit, administrative und Aufsichtsfunktionen auf einer Anwendung auszuüben und dabei relevante funktionsbezogene Informationen zu erhalten."

Management-Funktionen bieten Mechanismen zur Sicherung des normalen Ablaufs einer Anwendung.

Es gibt folgende Management-Funktionen:

- **Überwachung (Monitoring):** Bestimmte Ereignisse werden erfasst und für Berichterstattung (Reporting) oder Benachrichtigung (Notification) genutzt.
- **Verfolgung (Tracking):** Die Bearbeitung einer Aufgabe wird über mehrere Komponenten hinweg verfolgt (z. B. das Verschicken einer Meldung vom Sender bis zum Empfänger).
- **Steuerung (Controlling):** Die Fähigkeit, das Laufzeitverhalten einer gemanagten Komponente zu verändern (beispielsweise die Lastverteilung zu ändern)

2.2 Servlets

Nachdem der Begriff des „Applikations-Management" definiert wurde, sollen nun die für diese Arbeit wichtigen Technologien vorgestellt werden.

Da Servlets die Grundlage für Java Server Pages (JSP) bilden, welche in dieser Diplomarbeit eine wichtige Rolle spielen, soll die Servlet-Technologie im Folgenden kurz erläutert werden.

Servlets sind Java-Klassen, die eine Anfrage eines Web-Application-Servers bearbeiten und eine Antwortseite generieren. Servlets werden innerhalb des Web-Application-Servers vom sog. Servlet-Container verwaltet. Dieser stellt seinen Servlets verschiedene Dienste, wie z. B. Sessionhandling, zur Verfügung und leitet Anfragen an die Servlets weiter.

Die Schnittstelle zwischen Servlets und Servlet-Container ist in einer von Sun spezifizierten API [Serv] beschrieben. Der Kernpunkt dieser API ist die Schnittstelle `Servlet` bzw. ihre Erweiterung für das HTTP-Protokoll `HttpServlet`. Sie muss von jeder Servlet-Klasse implementiert werden.

Wird das Servlet von einem Browser aus aufgerufen, so übergibt die Servlet-Engine den Kontext des Aufrufs und den Kontext der Antwort über die Servlet-Schnittstelle an das

Servlet. Der Aufrufkontext beinhaltet u. a. die Seitenparameter, die via Get- oder Post-Methode vom Browser übergeben wurden. Der Antwortkontext ermöglicht den Zugriff auf einen `Writer`, über den die Antwortseite geschrieben werden kann.

Das Servlet ist somit in der Lage, eine Anfrage aus dem Netz entgegen zu nehmen, zu bearbeiten und eine Antwortseite zu generieren. Üblicherweise besteht diese Antwortseite aus HTML-Code. Dabei können alle Möglichkeiten der Java-Technologie genutzt werden, insbesondere die mächtige Java-Standardbibliothek, um z. B. über JDBC auf eine Datenbank zuzugreifen, oder eine der zahlreichen externen Bibliotheken, um beispielsweise dynamisch Bilder für Graphen zu generieren. Alle Protokolldetails und die Behandlung bei gleichzeitigen Anfragen werden dabei von der Servlet-Engine übernommen.

2.3 Java Server Pages

Java Server Pages (kurz: JSP) sind ein von Sun spezifizierter Standard [JSP] für die Entwicklung dynamischer Webseiten. Dynamische Webseiten haben trotz ihrer dynamischen Natur oft sehr große, statische Teile. Ein Beispiel: Eine Seite mit einer dynamisch erstellten Tabelle benötigt nur für den Teil Java-Code, der durch die Quelldaten iteriert und die Werte ausliest. Der gesamte Rest, wie z. B. die Kopfzeile mit dem Firmenlogo, die Copyright-Notiz oder ein beschreibender Text, ist statisch.

JSPs drehen daher den Ansatz von Servlets um: Anstatt HTML-Code in eine Java-Klasse einzubinden, wird Java-Code in die HTML-Seite eingebunden. Dies geschieht über spezielle Meta-Tags, die den Java-Code beinhalten. Eine JSP-Seite ist dadurch wesentlich besser lesbar als ein Servlet.

Intern werden JSP-Seiten jedoch wie Servlets behandelt: Beim ersten Aufruf einer JSP-Seite übersetzt die Servlet-Engine die JSP in ein Servlet, compiliert dieses und führt es schließlich aus. Alle weiteren Aufrufe werden dann direkt vom bereits compilierten Servlet bearbeitet.

2.4 Tag-Libraries

Tag-Libraries sind ein Bestandteil der JSP-Spezifikation [JSP]. Mit dem Einsatz von Java Server Pages hat man bereits erreicht, dass nur noch wenig Java-Code nötig ist, um eine dynamische Web-Seite zu erstellen. JSPs sind jedoch trotzdem nicht frei von Java-Code und deshalb für Web-Designer schwer zu bearbeiten.

Mit Hilfe von Tag-Libraries ist es möglich, JSP-Seiten zu entwickeln, die nur noch wenig bis gar keinen Java-Code beinhalten. Solche JSP-Seiten bieten dann die Schnittstelle zwischen dem Web-Designer, der kein Java versteht, und dem Web-Entwickler, der die dynamischen Teile einer Seite entwickelt. Tag-Libraries können zudem in mehreren JSP-Seiten verwendet werden.

Eine Tag-Library besteht aus einer Sammlung von Tag-Klassen und einer Tag-Library-Description (TLD). Tag-Klassen sind Java-Klassen, die eine bestimmte Schnittstelle implementieren. In der TLD ist für jedes Tag aufgeführt, welche Klasse dafür zuständig ist und welche Attribute es bietet. In der JSP können diese speziellen Tags in XML-Notation eingebunden werden, z. B.: `<mylib:mytag myattr1="25" myattr2="xyz"/>`. Der Java-Code ist somit von der JSP-Seite in die Tag-Klasse ausgelagert.

Sobald die Abarbeitung einer JSP-Seite das Start- bzw. End-Tag eines Tags erreicht, ruft die Servlet-Engine bei der Tag-Klasse bestimmte Methoden auf. Die Tag-Klasse kann dann im Java-Code Berechnungen durchführen, Daten von einer Persistenzschicht lesen oder schreiben oder auch zusätzlichen HTML-Code in die Antwortseite schreiben.

2.5 Web Dynpros

Web Dynpros sind ein von SAP entwickeltes Framework zur Entwicklung von Web-Anwendungen nach dem Model-View-Controller-Entwurfsmuster (MVC). Sie sind seit der Version 6.30 Teil des SAP Web Application Servers. Das Web-Dynpro-Framework unterstützt alle Merkmale, die von einer zeitgemäßen Web-Anwendung gefordert werden, wie Eingabeprüfung, Eingabehilfe, Internationalisierung und Fehlerbehandlung.

Web Dynpros setzen als Entwicklungsumgebung das SAP NetWeaver Developer Studio voraus. Damit kann der Entwickler den Workflow der Anwendung und die Anordnung der Bedienelemente mit graphischer Unterstützung zusammenstellen. So entsteht eine Meta-Beschreibung der Web-Anwendung. Aus dieser Meta-Beschreibung generiert NetWeaver im Hintergrund den Laufzeitcode des Controllers. Zur Laufzeit erzeugt der Controller die Bedienelemente (Views) und verbindet sie mit einem Datenmodell (Model). Der Workflow wird dabei durch eine einfache Ereignisbehandlung abgebildet. So wird z. B. eine andere View gezeigt, wenn auf einen bestimmten Knopf gedrückt wurde. An manchen Stellen, z. B. bei einer weitergehenden Ereignisbehandlung, muss dabei jedoch noch von Hand programmiert werden.

Eine Client-seitige Javascript-Bibliothek erlaubt es, dass jeweils nur der Teil einer Seite geladen werden muss, der sich auch geändert hat. Dieses sog. „clientseitige Rendering“ soll laut SAP mit hohen Versionen des Internet Explorer und Netscape Navigators

funktionieren. Bei anderen Browsern, z. B. auf anderen Geräten, soll der HTML-Code komplett serverseitig generiert werden. Nach eigener Erfahrung funktionert es jedoch nur mit dem Internet Explorer wirklich einwandfrei. Da Web-Dynpros noch relativ neu sind, ist zu erwarten, dass die Unterstützung alternativer Browser sich noch verbessern wird.

2.6 JMX

Die Java Management Extension (kurz JMX) ist eine im Rahmen des Java Community Process (JCP) entwickelte Spezifikation [JMX] einer Management-Architektur. JMX hat zwei primäre Ziele: Zum einen soll es mit wenig Aufwand möglich sein, eine eigene Anwendung in JMX zu integrieren und somit verwaltbar zu machen. Zum anderen soll JMX in der Lage sein, sich in bestehende und zukünftige Management-Architekturen einbetten zu können.

Da JMX eine zentrale Rolle in dieser Diplomarbeit spielt, wird im Folgenden auf diese Technologie etwas genauer eingegangen.

Man kann mit ziemlicher Sicherheit davon ausgehen, dass JMX der zukünftige Standard für Management im Bereich Java werden wird oder sogar bereits ist. Praktisch alle namhaften Java-Server-Systeme unterstützen bereits heute JMX. In Java 1.5 ist JMX als fester Bestandteil integriert. Das bedeutet nicht nur, dass JMX Teil der neuen Java Standard-API ist. Die gesamte Java-VM vom Garbage-Collector bis zu Betriebsystemressourcen wie CPU-Last oder Arbeitsspeicher kann über JMX verwaltet werden.

Als kleinste verwaltbare Einheit definiert JMX die Managed Bean, kurz MBean. Eine MBean ist eine Java-Klasse, die eine Ressource verwaltet. Das kann z. B. ein Drucker, ein Rechner, ein Benutzer oder ein Dienst sein.

Die JMX-Architektur teilt sich in drei Ebenen (siehe Abbildung 2.1 auf der nächsten Seite):

- **Der Distributed Services Level:** Hier befindet sich die Management-Applikation (kurz: Manager) sowie die Kommunikationsschicht zwischen dem Manager und dem JMX-Agenten.
- **Agent Level:** Im Agent Level liegt, wie der Name schon sagt, der JMX-Agent, auch MBean-Agent genannt. Er registriert und verwaltet die MBeans und kapselt sie zum Distributed Services Level ab. Ein Manager kann also nicht direkt auf eine MBean zugreifen, sondern muss alle Anfragen an den Agent richten, der sie dann an die MBean weiterleitet.

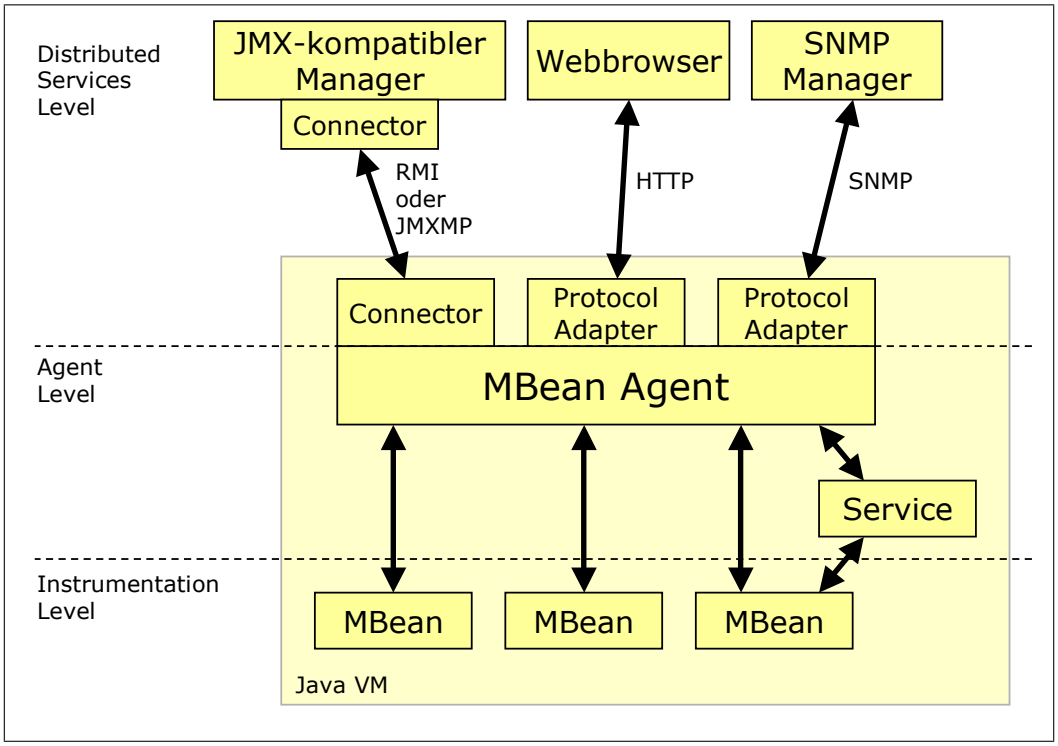

Abbildung 2.1: Die Architektur von JMX [JMX]

- **Instrumentation Level:** In diesem Level befinden sich die MBeans, die den Zugriff auf die Ressourcen bieten, die von einem Manager überwacht und gesteuert werden sollen.

2.6.1 Distributed Services Level

JMX sieht zwei Möglichkeiten vor, wie ein Manager mit einem JMX-Agent kommunizieren kann: Konnektoren und Protokolladapter.

Ein Konnektor arbeitet nach dem Proxy-Entwurfsmuster und hat zur Aufgabe, die Zugriffsschnittstelle zum Agenten netzwerktransparent zu machen. Ein Konnektor besteht daher aus zwei Teilen: Ein Teil befindet sich im Manager und nimmt dort alle Anfragen über die gleiche Schnittstelle entgegen, wie sie ein Agent bietet. Die Anfrage wird über das Netzwerk zum zweiten Teil des Konnektors übertragen, der sie dann an den Agenten weitergibt. Die Antwort des Agenten wird auf dem umgekehrten Weg an den Manager zurückgesendet. Welches Protokoll dabei eingesetzt wird, bleibt dem Konnektor überlassen. Für den Manager besteht kein Unterschied darin, ob er direkt mit einem

lokalen Agenten oder über einen Konnektor mit einem entfernten Agenten kommuniziert.

Ein Protokoll-Adapter bietet über ein bestimmtes Protokoll Zugriff auf den Agenten, indem die Protokollfunktionen in entsprechende JMX-Anfragen übersetzt werden. Im Gegensatz zum Konnektor dient ein Protokoll-Adapter der Anbindung von nicht JMX-fähigen Managern. Mit Hilfe eines SNMP-Protokoll-Adapters beispielsweise sieht ein JMX-Agent für einen SNMP-Manager wie ein normaler SNMP-Agent aus und kann somit in ein bestehendes SNMP-basiertes System integriert werden.

2.6.2 Agent Level

Der Agent stellt das Bindeglied zwischen der durch MBeans instrumentierten Anwendung und dem Manager dar.

Im Folgenden sollen nun drei zentrale Aufgabenbereiche des Agenten beschrieben werden:

- **MBean-Queries:** Sie ermöglichen es einem Manager, eine MBean oder eine Menge von MBeans zu finden.
- **Notification-Mechanismus:** Über Notifications kann eine MBean einen Manager über ein bestimmtes Ereignis informieren.
- **Agent-Services:** Sie übernehmen verschiedene Serviceleistungen für MBeans, wie beispielsweise eine automatische Benachrichtigung in bestimmten Zeitintervallen.

MBean-Queries

MBeans werden innerhalb eines MBean-Agents durch die Domain und durch die Schlüsselattribute eindeutig bestimmt. Die Domain ist dabei ein beliebiger Name, mit dem MBeans von verschiedenen Herstellern oder Applikationen logisch voneinander abgetrennt werden.

Die Domain bildet zusammen mit den Schlüsselattributen den Objektnamen. Ein Beispiel für einen Objektnamen wäre `MyDomain:type=Printer,description=laser`. Vor dem Doppelpunkt steht die Domain, danach folgen durch Komma getrennt die Schlüssel-Wert-Paare der Schlüsselattribute. Die Reihenfolge der Schlüsselattribute ist dabei unerheblich. Mit Hilfe des Objektnamens lässt sich eine MBean also innerhalb eines Agents eindeutig bestimmen.

Um bei einem Agent Mengen von MBeans abzufragen, können Objektnamen mit Wildcards versehen werden, so dass sie auf mehrere MBeans zutreffen können.

So werden bei einer Anfrage nach `*:*` alle MBeans des Agents zurückgegeben. Eine Anfrage nach `mydomain:type=Printer,*` hingegen liefert alle MBeans der Domain `mydomain`, die einen Drucker repräsentieren.

Notification-Mechanismus

Normalerweise folgt der Informationsfluss bei JMX dem Client-Server-Paradigma: Ein Manager schickt eine Anfrage an den Agent, dieser bearbeitet sie und sendet eine Antwort zurück.

JMX sieht jedoch auch eine Möglichkeit vor, wie eine MBean eine Benachrichtitung über ein Ereignis an eine andere MBean oder einen Manager senden kann.

Dieser Notification-Mechanismus hat folgenden Ablauf: Um Benachrichtigungen schicken zu können, muss eine MBean die Schnittstelle `NotificationBroadcaster` implementieren. Über diese Schnittstelle können sich sog. `NotificationListener` registrieren.

Trifft nun ein Ereignis ein, von dem die MBean seine Listener benachrichtigen will, so erzeugt sie ein Objekt der Klasse `Notification` und reicht es an die registrierten Listener weiter.

Wenn ein Listener nur bestimmte Benachrichtigungen erhalten will, so kann er bei der Registrierung einen `NotificationFilter` mitgeben. Dieser entscheidet, ob eine Benachrichtung weitergegeben werden soll oder nicht.

Agent-Services

Agent-Services sind spezielle MBeans, die vom MBean-Agent bereitgestellt werden müssen. Diese Services können sowohl von einem Manager als auch von MBeans genutzt werden.

In der aktuellen Version 1.2 der JMX Spezifikation sind vier Service-Typen vorgeschrieben:

- **Dynamic Class Loading Service:** Dieser Service kann Klassen dynamisch zur Laufzeit nachladen. Dabei kann eine URL mitgegeben werden, so dass sich die Klasse nicht zwingend im aktuellen Classpath befinden muss.

- **Monitor Service:** Der Monitor kann zyklisch Werte von MBeans überwachen, um eine Notification zu versenden, sobald sich der Wert ändert.
- **Timer Service:** Der Timer kann zu einem bestimmten Zeitpunkt oder in einem bestimmten Zeitintervall Notifications erzeugen.
- **Relation Service:** Dieser Service ermöglicht die Definition von Relationen. Wird eine Relation verletzt, so werden die beteiligten MBeans entfernt.

2.6.3 Instrumentation Level

Wie bereits erwähnt, werden Ressourcen mit Hilfe von MBeans für das Management bereitgestellt. Die Implementierung einer MBean für eine Ressource wird Instrumentierung genannt.

Eine MBean bietet Attribute und Operationen. Ein Attribut ist ein Wert, wie z. B. eine Zahl oder eine Zeichenkette. Dabei kann es sich um einen Stammdatenwert wie den Namen der Herstellers handeln oder um einen Bewegungsdatenwert, wie z. B. die aktuelle Auslastung. Attribute können lesbar, schreibbar oder beides sein. Eine Operation ist eine Aktion, die mit der Ressource durchgeführt werden kann, wie z. B. ein Reset. Operationen können Übergabeparameter und einen Rückgabewert haben.

Welche Attribute und Operationen einer MBean nach außen zugänglich und damit verwaltbar sein sollen, wird durch das sog. Management Interface der MBean bestimmt.

JMX bietet vier Möglichkeiten, eine MBean zu definieren. Welche dieser Möglichkeiten genutzt wird, bleibt dem Programmierer überlassen. Dadurch, dass alle MBeans vom Agent gekapselt werden, ist von Seite des Managers nicht erkennbar, um welchen MBean-Typ es sich handelt.

Standard-MBeans

Die einfachste Art, eine MBean zu definieren, ist die Standard-MBean. Der Programmierer implementiert eine Klasse, die die Ressource repräsentiert. Die Namenskonvention für Operationen und Methoden, die auf Attribute zugreifen, folgen dem Java-Bean-Muster: Attribute werden je nach dem, ob sie les- und/oder schreibbar sind mit einer Getter- und/oder einer Setter-Methode zugänglich gemacht. Alle anderen Methoden werden als Operationen behandelt.

Anschließend schreibt er das Management-Interface, indem er eine Schnittstelle erstellt, die den gleichen Namen wie die Klasse hat, mit dem Anhang „MBean“. In die-

ser Schnittstelle definiert er alle Getter, Setter und sonstigen Methoden, die er für das Management bereitstellen will. Schließlich lässt er noch die Klasse von der MBean-Schnittstelle erben (siehe Abbildung 2.2).

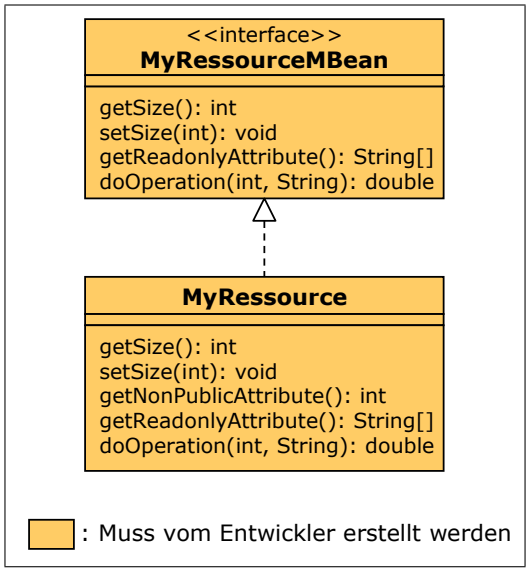

Abbildung 2.2: Klassendiagramm einer Standard-MBean

Dynamic-MBeans

Eine Dynamic-MBean ist eine Klasse, die die Schnittstelle `DynamicMBean` implementiert. Über diese Schnittstelle kann ein Meta-Daten-Objekt abgefragt werden, welches das Management Interface beschreibt. Im Gegensatz zu Standard-MBeans wird diese Beschreibung erst zur Laufzeit erstellt und kann sich sogar ändern. Außerdem bietet die Schnittstelle Methoden zum eigentlichen Zugriff auf Attribute und Operationen (siehe Abbildung 2.3 auf der nächsten Seite).

Model-MBeans

Mit Hilfe von Model-MBeans ist es möglich, Instrumentierung in bestehende Anwendungen zu integrieren ohne die Anwendung selbst ändern zu müssen. Sie dienen als Wrapper für die Objekte der Anwendung, die instrumentiert werden sollen.

Eine Model MBean ist eine Klasse, die die Schnittstelle `ModelMBean` implementiert. Diese erweitert die Schnittstelle `DynamicMBean` um Persistenz und um die Möglichkeit, das gewrappte Objekt neu zu setzen (siehe Abbildung 2.4 auf Seite 13).

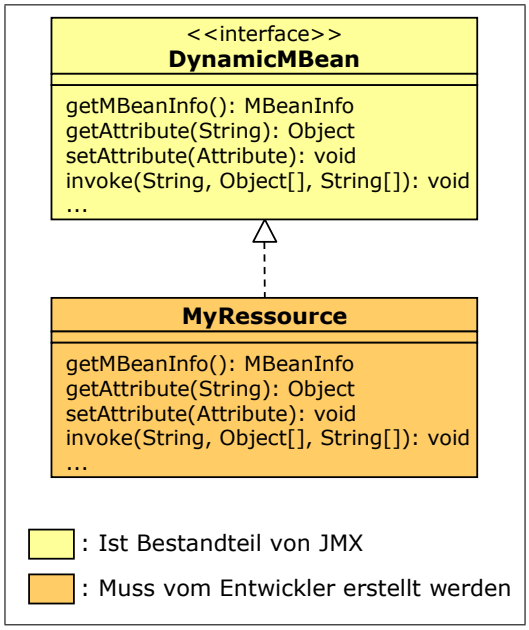

Abbildung 2.3: Beispiel einer Dynamic-MBean

Open-MBeans

Open-MBeans sind Dynamic-MBeans deren Attributwerte, Parameter und Rückgabewerte auf bestimmte Typen, sog. Basic-Types, beschränkt sind. Ein Manager, der alle Basic-Types unterstützt, kann somit automatisch mit jeder Open-MBean arbeiten.

Basic-Types können Skalare, Arrays von Skalaren, Strukturen und Tabellen sein. Skalare können Zeichenketten (Strings), Ganzzahlen, Fließkommazahlen, Zeitstempel oder Bool'sche Werte sein. Strukturen sind Objekte der Klasse `CompositeData` und stellen eine Sammlung von Schlüssel-Wert-Paaren dar, wobei die Schlüssel Zeichenketten und die Werte wiederum Basic-Types sein müssen. Tabellen sind entweder `TabularData`-Objekte oder Arrays von Strukturen gleichen Typs.

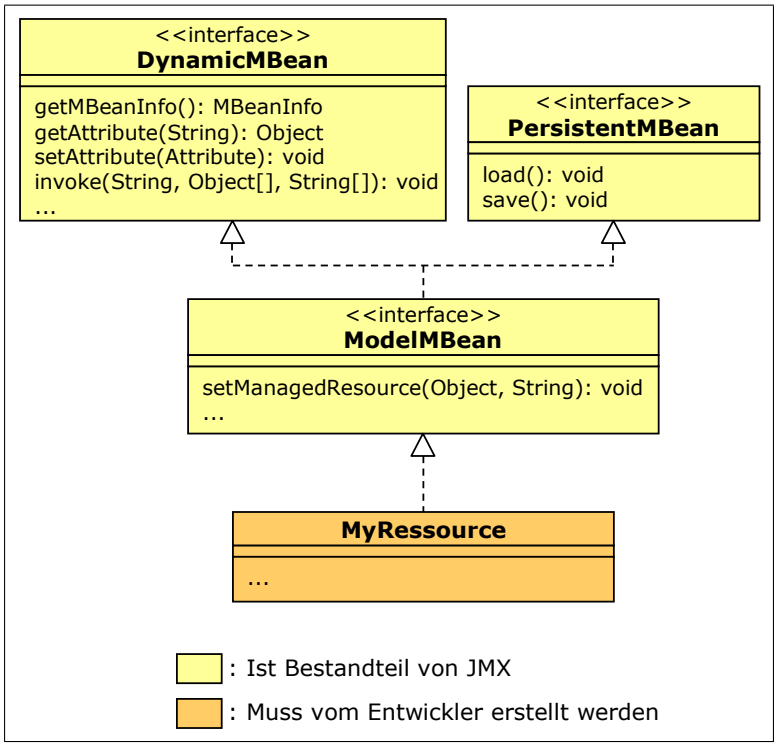

Abbildung 2.4: Beispiel einer Model-MBean

Kapitel 3

Aufgabenstellung

In diesem Kapitel wird zunächst einmal die im Rahmen dieser Diplomarbeit zu bewältigende Aufgabe vorgestellt. Es wird gezeigt, welche Anforderungen bestehen, welche Technologien eingesetzt werden und wie das Ergebnis der Arbeit aussehen soll.

3.1 Das Mercator-System

Der Mercator ist das Server-System, das im Rahmen dieser Diplomarbeit prototypisch überwacht werden soll. Bevor die Probleme bei der Administration erläutert werden, soll dieses System zunächst einmal vorgestellt werden.

Mercator ist ein Enterprise-Application-Integration-System und wird mittlerweile von der Firma Ascential unter dem Namen „Data Stage TX“ vertrieben. Bei dm-drogerie markt ist jedoch der alte Name „Mercator“ geläufiger, so dass ich in dieser Arbeit auch diesen Namen nutzen werde.

Der Mercator ist ein zentrales Server-System, das in der Infrastruktur von dm-drogerie markt eine wichtige Rolle übernimmt: Es konvertiert die Daten, die von einem System in ein anderes transferiert werden sollen. Der grobe Ablauf ist dabei stets der Folgende: Die Daten werden Mercator übergeben, indem sie in ein Eingangsverzeichnis geschrieben werden. Mercator passt die Datenstruktur an das Zielsystem an und schreibt sie in ein Ausgangsverzeichnis. Von dort werden sie schließlich an das Zielsystem übergeben.

Wie die Daten in die Struktur des Zielsystems umzuwandeln sind, wird dabei durch so genannte „Mappings“ definiert (siehe Abbildung 3.1 auf der nächsten Seite). Ein

Mapping enthält die Umwandlungsregeln von einem Quellformat in ein Zielformat. Die Entwicklung der Mappings erfolgt in einer speziellen Entwicklungsumgebung. Mercator erkennt anhand des Dateinamens, welches Mapping auf eine konkrete Eingangsdatei anzuwenden ist.

Das SAP-System nimmt dabei eine Sonderrolle ein: Während die Daten aller anderen Systeme über die gerade vorgestellte Dateischnittstelle an Mercator übergeben bzw. von Mercator abgeholt werden, kann Mercator Daten in Form von IDocs vom SAP-System direkt lesen bzw. sie direkt ins SAP-System schreiben. Für Datentransfers aus dem SAP-System heraus wird in das Eingangsverzeichnis lediglich eine leere Datei gelegt, die als Auslöser der Konvertierung fungiert.

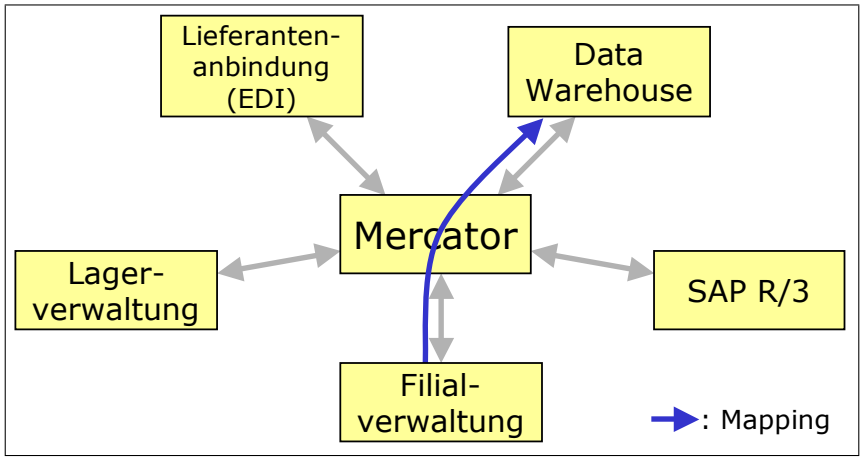

Abbildung 3.1: Kopplung der IT-Systeme bei dm-drogerie markt

dm-drogerie markt arbeitet prinzipiell mit einem dreistufigen Systemverbund: Entwicklungs-, Konsolidierungs- und Produktionssytem. Für jede dieser Stufen steht ein separates Mercator-System zur Verfügung. Das Entwicklungssystem wird für die eigentliche Entwicklung genutzt, im Konsolidierungssystem wird die Änderung getestet und im Produktionssystem findet schließlich der eigentliche Betrieb mit den realen Daten statt.

Für jedes Land, in dem dm-drogerie markt vertreten ist, wird es in Zukunft den oben genannten dreistufigen Systemverbund geben. Momentan befindet sich dm-drogerie markt noch in einer Aufbauphase, so dass noch nicht alle Systeme auf die einzelnen Länder aufgeteilt sind. Das Mercator-System existiert momentan nur für Deutschland und Österreich. Es gibt also ingesamt bereits 6 Systeme. Ab 2005 soll der Aufbau der Mercatorsysteme auch für die Länder Italien, Kroatien, Ungarn, Slowenien, Slowakei und Tschechien abgeschlossen sein. Dadurch werden dann in der Summe 24 Mercatorsysteme im Einsatz sein.

3.2 Problemstellung

In den produktiven Mercator-Systemen existieren etwa 200 Mappings, die von insgesamt 20 bis 30 Mapping-Betreuern gepflegt werden. Auf jeden Mapping-Betreuer fallen zwischen einem und zwanzig Mappings. Täglich werden von diesen Mappings insgesamt etwa 20 000 Datensätze bearbeitet. Die technische Seite wird von zwei Administratoren betreut.

Durch fehlerhafte Eingangsdaten oder nicht erfüllte Vorbedingungen im Zielsystem kommt es regelmäßig zu Mapping-Abbrüchen. Selbst bei einer relativ kleinen Fehlerquote von 0,3% bedeutet das ca. 60 Fehlerfälle pro Tag. Der jeweilige Mapping-Betreuer muss nun das Problem analysieren, beheben und den betroffenen Datensatz erneut von Mercator bearbeiten lassen.

Zur Fehleranalyse prüft der Mapping-Betreuer im Quell- und Zielsystem, welche Daten abgeschickt bzw. empfangen wurden. Falls er mit der Analyse nicht weiterkommt, wendet er sich an einen der Administratoren. Dieser analysiert den Fehler momentan noch sehr systemnah durch:

- Einsehen des Dateisystems
- Aufruf von Shellskripten
- Direktes Absetzen von Datenbankabfragen
- Abfragen innerhalb des SAP-Systems

Die Administration erfordert daher viel Fachwissen und verursacht hohe Kosten wegen des hohen Zeitaufwands. Den Mapping-Betreuern fehlt eine Übersicht über den Zustand ihrer Mappings und den Adminstratoren eine Gesamtübersicht, die schnell einen Überblick über den aktuellen Zustand des gesamten Systems sowie einzelner Mappings bietet.

3.3 Anforderungen

Die zu entwickelnde Anwendung besteht aus zwei Teilen: Zum einen soll ein Framework entstehen, das es ermöglicht, sehr schnell eine Management-Applikation zu erstellen. Zum anderen soll dieses Framework genutzt werden, um prototypisch eine Applikation zu entwickeln, mit deren Hilfe die Mercator-Systeme überwacht werden können.

3.3.1 Anforderungen an das Framework

Das Framework soll einer Management-Applikation, die damit erstellt wird, möglichst viel Entwicklungsaufwand abnehmen. Dabei soll es trotzdem allgemein bleiben.

Die Anforderungen:

- *allgemeines Framework:* Es soll ein allgemeines Framework entstehen, das nicht auf die Überwachung der Mercator-Systeme spezialisiert ist, sondern möglichst universell für die Überwachung jeglicher Server-Systeme genutzt werden kann.
- *Web-basierte Benutzerschnittstelle:* Um nicht bei jedem Mapping-Betreuer und Administrator ein Clientprogramm installieren und aktuell halten zu müssen, soll die Benutzerschnittstelle Web-basiert gestaltet werden. Dies erlaubt auch eine Einbindung in das Intranet-Portal.
- *Multi-User-Fähigkeit:* Als Framework für Webanwendungen müssen natürlich gleichzeitige Zugriffe mehrerer Benutzer ermöglicht und unabhängig voneinander bearbeitet werden.
- *Erweiterbarkeit:* Die Architektur des Frameworks soll auf allen Ebenen Erweiterungen zulassen, sowohl von Seiten des Frameworks als auch um spezialisierte Detaillösungen einer Management-Applikation zuzulassen.
- *Modularisierung:* Das Framework soll in Module aufgeteilt sein, die sich austauschen und möglichst auch unabhängig von den anderen Module einsetzen lassen.
- *Skalierbarkeit:* Die Last soll mit steigendem Umfang einer Management-Applikation oder mit steigender Benutzerzahl nicht überproportional steigen. Außerdem soll eine Lastverteilung auf mehrere Rechner oder eine Lastminderung durch geeignete Caching-Strategien möglich sein.
- *Verteilbarkeit:* Der Client und die Management-Applikation können sich durch die Web-basierte Benutzerschnittstelle bereits auf verschiedenen Rechnern befinden. Das Framework soll jedoch auch eine Verteilung der Module zulassen. Insbesondere soll es möglich sein, dass die Management-Applikation auf einem anderen Rechner wie die zu überwachende Anwendung läuft. Außerdem soll auch ermöglicht werden, mit einer Management-Applikation Anwendungen auf mehreren Rechnern zu überwachen.
- *Fehlerbehandlung:* Das Framework soll auftretende Ausnahmen abfangen und bei Folgefehlern die Ursache mit angeben (Nested Exceptions). Eine Ausnahme soll sich durch alle Schichten hinweg verfolgen lassen (Exception Tracking).
- *Nutzung vorhandener Technologien:* Soweit möglich sollen vorhandene Technlogien eingesetzt werden. Das Framework soll nicht das „Rad neu erfinden“, sondern es soll sich in die genutzten Technologien einbetten.

3.3.2 Anforderungen an den Prototyp

Für die Mercator-Systeme soll der Einsatz und die Fähigkeiten des Frameworks anhand eines Prototyps gezeigt werden. Dieser Prototyp wird InterfaceCockpit heißen und soll zwei Seiten der Mercator-Systeme beleuchten:

- Die Betriebssystemseite: Das beinhaltet beispielsweise die CPU-Auslastung, den belegten Arbeitsspeicher oder die Anzahl der Dateien in bestimmten Verzeichnissen. Diese Daten müssen vom System (eine IBM AIX) abgefragt werden, beispielsweise über die üblichen Kommandos wie `ps` oder `ls`.
- Die Anwendungsseite: Hierbei sind Mercator-spezifische Daten gemeint, wie z. B. Daten über ein Mapping. Diese Daten kommen zum Teil von Shell-Skripten, welche sie aus Logdateien oder Konfigurationsdateien auslesen, oder sie werden über RFC-Aufrufe am SAP-System abgefragt, da das Mercator-System Status- und Protokollinformationen im SAP-System ablegt.

Das InterfaceCockpit kennt folgende Anwender-Use-Cases:

- Ein Anwender möchte Information über das Interface-Cockpit selbst bekommen. (Hilfe-Funktion)
- Ein Anwender möchte persönliche Einstellungen im Interface-Cockpit ändern.
- Ein Mapping-Betreuer möchte den Status seiner Mappings abfragen.
- Ein Mapping-Betreuer möchte Details zu einem Mapping ansehen.
- Ein Administrator möchte einen Bericht über den Status des Mercator-Servers.

Anforderungen:

- *Personalisierbarkeit:* Das InterfaceCockpit soll personalisierbar sein, so dass ein Benutzer beispielsweise einstellen kann, welches System das InterfaceCockpit beim Start zeigen soll.
- *Wenig Umfang:* Da das InterfaceCockpit den Teil der Entwicklungsarbeit repräsentiert, der bei jeder Entwicklung einer neuen Management-Applikation erbracht werden muss, soll es möglichst schlank sein. Die Hauptleistung soll vom Framework erbracht werden. Das InterfaceCockpit soll nur die Mercator-spezifische Entwicklung beinhalten.
- *Übersichtlichkeit:* Ein Anwender soll sich in wenigen Navigationsschritten zur gewünschten Information bewegen können. Die Ansichten sollten auf die Use-Cases zugeschnitten sein und die Informationen zeigen, die im jeweiligen Use-Case gewünscht sind.

- *Filterbarkeit:* Dieser Punkt zielt vor allem auf die Menge der Mappings. Ein Mercator-System kennt ca. 200 Mappings. Ein Mapping-Betreuer ist jedoch jeweils nur für etwa 10 Mappings zuständig. Ein Mapping-Betreuer soll daher eine Möglichkeit bekommen, „seine“ Mappings herauszufiltern, so dass nur diese angezeigt werden.
- *Ständige Systemanzeige und schneller Systemwechsel:* Auf verschiedenen Mercator-Systemen existieren oft die gleichen Mappings. Wegen der Verwechslungsgefahr ist es deshalb wichtig, auf jeder Seite anzuzeigen, welches System gerade betrachtet wird. Über diese Anzeige soll zusätzlich die Möglichkeit geboten werden, schnell von einem System auf ein anderes zu wechseln.

Kapitel 4

Technologieanalyse

In diesem Kapitel sollen zwei alternative Lösungswege der Diplomarbeit, die jeweils auf verschiedenen Technologien beruhen, vorgestellt, analysiert und miteinander verglichen werden. Abschließend wird eine Auswahl nach bester Eignung in Bezug auf die Anforderungen getroffen.

4.1 JMX mit JSPs

Eine Möglichkeit, eine Management-Applikation Web-basiert zu erstellen, wäre, die Technologien JSP und Tag-Libraries zusammen mit JMX einzusetzen.

JMX liefert dabei die Verwaltung der Ressourcen (MBeans) und die Netzwerktransparenz zwischen Webserver und dem Rechner, auf dem die zu überwachende Anwendung läuft.

Die eigentliche Darstellung der Werte wird von JSPs übernommen. Um den Entwicklungsaufwand der Web-Anwendung zu minimieren, kann das im Rahmen dieser Diplomarbeit zu entwickelnde Framework eine Tag-Library anbieten, die den Zugriff auf MBeans und die Darstellung von MBean-Werten übernimmt.

Auf der anderen Seite müssen MBeans entwickelt werden, die die Daten über die Ressourcen anbieten und aufbereiten. Dies kann das Framework durch eine Hilfsbibliothek erleichtern, die Probleme bewältigen kann, die für einen bestimmten Ressourcentyp typisch sind. Ein Beispiel hierfür wäre eine Abbildung von SAP-Funktionsbausteinen auf

MBeans oder eine Unterstützung, um Kommandozeilenskripten auszuführen und deren Ausgabe zu parsen.

Die Entwicklung einer neue Management-Applikation würde sich auf das Erstellen der JSP-Seiten und die Entwicklung der MBeans beschränken, wobei beides in großen Teilen vom Framework unterstützt werden kann (siehe Abbildung 4.1).

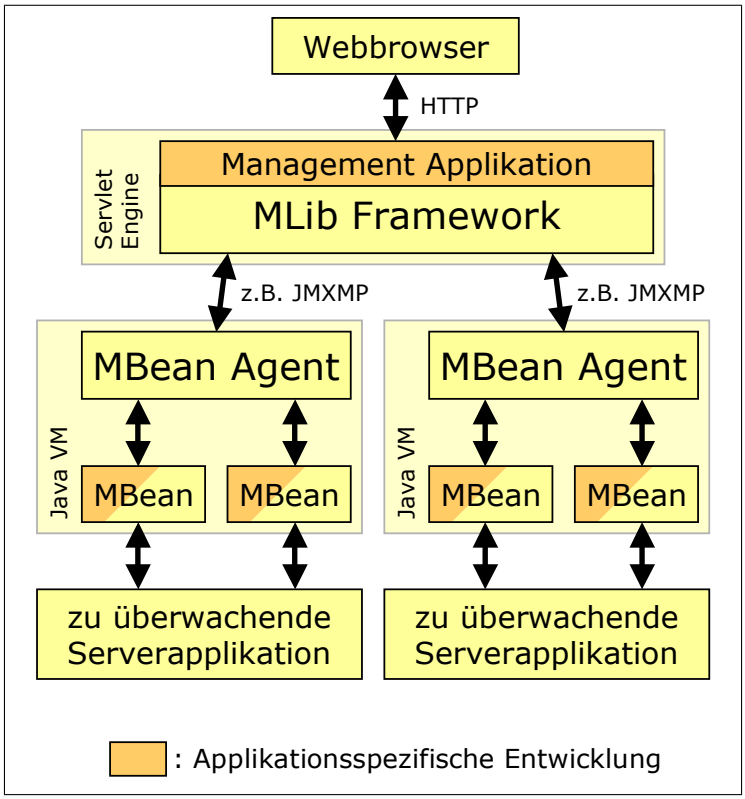

Abbildung 4.1: Architekturentwurf mit JMX und JSPs

Vorteile von JMX in Verbindung mit JSPs:

- *Offener Standard*: Sowohl JMX als auch JSP sind offene Standards. Für beide gibt es auch bereits verschiedene, voneinander unabhängige Implementierungen. Viele davon sind kostenlos und Open Source.
- *JMX setzt sich durch*: Es gibt bereits viele Java-Projekte, die für JMX instrumentiert sind. Darunter bezufindet sich auch die neue Java Version 1.5. JMX ist in Java 1.5 integriert und die Java VM kann zu großen Teilen über JMX verwaltet werden und bietet sogar Daten über Betriebsystemressourcen, wie CPU oder den Arbeitsspeicher. Auch die meisten großen Java-Serversysteme wie z. B. BEA WebLogic, JBoss oder IBM WebSphere sind ebenfalls bereits für JMX instrumentiert. Somit

kann ein großer Teil der Java-Welt in Management-Applikationen eingebunden werden, ohne dass zusätzlicher Entwicklungsaufwand entsteht.

- *Wiederverwendbarkeit:* Einmal über JMX erschlossene Ressourcen lassen sich auch von anderen Management-Applikation ansprechen. Durch JMX-Protokoll-Adapter können dazu sogar Management-Applikation genutzt werden, die gar keine JMX-Unterstützung bieten (siehe Abschnitt 2.6.1 auf Seite 8).
- *Know-How bereits vohanden:* Da der Internetauftritt und das Intranet von dm-drogerie markt auf JSP-Technologie basiert, ist bereits reichlich Know-How im Bereich JSP bei dm-drogerie markt vorhanden.

Nachteile von JMX in Verbindung mit JSPs:

- Komplexere Widgets, wie Tabellen, Baumansichten o. ä. müssen selbst erstellt werden.

4.2 SAP WebDynpros

Ein Teil der Daten, die im InterfaceCockpit angezeigt werden sollen, kommt aus dem SAP-System. Warum also nicht SAP-eigene Mittel nutzen, um sie anzuzeigen? SAP liefert mit dem SAP Web Application Server (kurz: SAP WAS) ein brandneues Werkzeug zur Erstellung von Webapplikationen.

Mit Hilfe von SAP WebDynpros erhält man eine einfache Anbindung an das SAP-System und eine recht mächtige Widgetsammlung, die mit einem GUI-Builder zu einer Oberfläche zusammengestellt werden kann.

Andere Quellen neben dem SAP-System können angebunden werden, indem sie als Webservice zugänglich gemacht werden. Das zu entwickelnde Framework kann ähnlich wie im vorigen Abschnitt beschrieben eine Hilfsbibliothek bieten, die die Erschließung der Ressourcen vereinfacht. Diese Hilfsbibliothek muss sich dabei zusätzlich um die Verwaltung und Veröffentlichung der Ressourcen kümmern, da SAP eine Anbindung über WebServices voraussetzt. Auch hier kann JMX eingesetzt werden, wobei ein Protokoll-Adapter entwickelt werden müsste, der einen JMX-Agenten als WebService zur Verfügung stellt.

Leider sieht die SAP WebDynpro-Architektur keine Möglichkeit vor, eigene Widgets zu entwickeln, so dass es keine Möglichkeit gibt, auf WebDynpro-Ebene eine Framework-Unterstützung anzubieten, die beispielsweise eine besondere Darstellung der Daten als spezielles Diagramm erlauben würde. Man ist auf die Widgets angewiesen, die von SAP

gestellt werden. Auch eine Anbindung anderer alternativer Quellen neben WebServices ist nicht möglich.

Die Entwicklung einer neuen Management-Applikation würde daher Folgendes erfordern: Die Erstellung von Funktionsbausteinen im SAP-System für die Anbindung von SAP-eigenen Daten, die Implementierung von Java-Klassen für die Anbindung von Ressourcen, die nicht im SAP-System liegen und eine komplette Entwicklung der Web-Applikation mit SAP WebDynpros (siehe Abbildung 4.2).

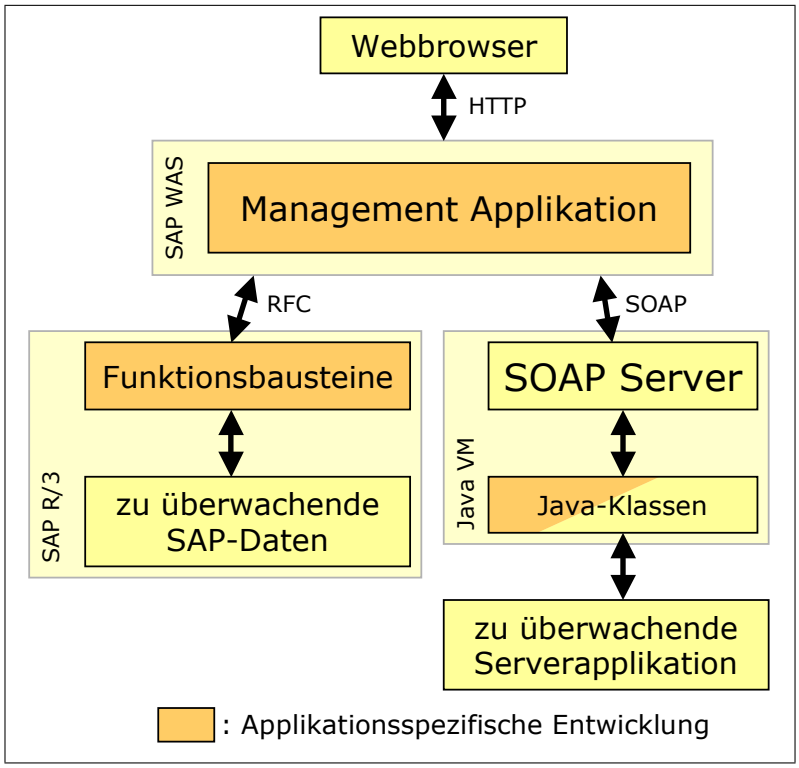

Abbildung 4.2: Architekturentwurf mit SAP WebDynpros

Vorteile von WebDynpros:

- *Mächtige Widgets:* WebDynpros bieten bereits recht mächtige Widgets, wie z. B. eine Tabelle, die Sortierung und Paging unterstützt.
- *Sehr gute SAP-Integration:* Daten aus dem SAP-System sind leicht zugänglich.

Nachteile von WebDynpros:

- *Proprietär*: Man ist an einen Hersteller gebunden.

- *Nur ein Entwicklungstool*: Man ist an ein Entwicklungstool gebunden: Das SAP NetWeaver Developer Studio.
- *Nur im SAP-Umfeld sinnvoll einsetzbar*: WebDynpros bringen vor allem dann Vorteile, wenn sie im SAP-Umfeld eingesetzt werden. Es macht wenig Sinn eine Webapplikation mit WebDynpros zu erstellen, die keine SAP-Daten nutzt.
- *SAP WAS vorausgesetzt:* WebDynpros laufen nur auf dem SAP Web-Application-Server (Kosten: ca. 40.000 Euro pro CPU)
- *Keine Erweiterungsmöglichkeiten:* Erweiterungen von WebDynpros, wie die Entwicklung eigener Widgets, sind nicht möglich.
- *Kein Know-How vorhanden:* dm-drogerie markt hat momentan noch keine Erfahrungen mit der WebDynpro-Technologie.

4.3 Vergleich und Auswahl

Um eine Auswahl zu treffen, werden im Folgenden beide Lösungswege mit den zuvor definierten Anforderungen abgeglichen und auf die Eignung geprüft.

Die Anforderungen an das Framework:

- *allgemeines Framework:* Die Forderung nach einem universell einsetzbaren Framework wird von der Lösung JMX/JSP erfüllt. Alles was aus Java heraus ansprechbar ist, läßt sich instrumentieren. Die Lösung WebDynpro erfüllt diese Forderung auch, jedoch ist die Anbindung von Ressourcen außerhalb des SAP-Systems wesentlich aufwändiger.
  (JMX/JSP: +, WebDynpro: +/-)
- *Web-basierte Benutzerschnittstelle:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)
- *Multi-User-Fähigkeit:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)
- *Erweiterbarkeit:* Was die Oberfläche angeht, so erlaubt nur die Lösung JMX/JSP Erweiterungen. Auch bei der Anbindung verschiedener Ressourcen ist JMX wesentlich flexibler, da die komplette Kommunikationsschicht sehr leicht austauschbar ist.
  (JMX/JSP: +, WebDynpro: -)

- *Modularisierung:* Diese Forderung hängt mehr von der konkreten Implementierung ab und kann mit beiden Lösungen erfüllt werden.
  (JMX/JSP: +, WebDynpro: +)

- *Skalierbarkeit:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

- *Verteilbarkeit:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

- *Fehlerbehandlung:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

- *Nutzung vorhandener Technologien:* Hier erlaubt die Lösung JMX/JSP eine weitaus flexiblere Einbindung von anderen Technologien. Es gibt unzählige Projekte, die beispielsweise Tag-Libraries für alle möglichen Belange liefern. Die Spanne reicht von kleinen Lösungen für kleine Probleme, etwa das Zeichnen eines Graphs bis zu Gesamtlösungen, wie sie z. B. von Projekten wie Struts oder Standards wie Java Server Faces gegeben sind. JSPs erlauben hier auch eine Auswahl, die der Größe des Projekts angemessen sind. WebDynpros erlauben nur das, was von SAP vorgesehen ist.
  (JMX/JSP: +, WebDynpro: -)

Die Anforderungen an das InterfaceCockpit:

- *Personalisierbarkeit:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

- *Wenig Umfang:* Die Entwicklung einer Anwendung mit Hilfe der WebDynpro-Lösung ist wesentlich aufwändiger, da die WebDynpro-Architektur keine Erleichterung seitens des Frameworks für die Erstellung der Oberfläche zulässt. Zudem ist die Bereitstellung einer Ressource als WebService wesentlich aufwändiger als die Implementierung einer MBean.
  (JMX/JSP: +, WebDynpro: -)

- *Übersichtlichkeit:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

- *Filterbarkeit:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

- *Ständige Systemanzeige und schneller Systemwechsel:* In beiden Lösungen gegeben.
  (JMX/JSP: +, WebDynpro: +)

Sowohl die Listen der Vor- und Nachteile, als auch der Vergleich mit dem Anforderungskatalog zeigt eine deutlich bessere Eignung der Lösung JMX/JSP. Sie lässt sich wesentlich flexibler einsetzen, kann an den Umfang der Management-Applikation angepasst werden, und ist nicht abhängig von einem Hersteller.

Kapitel 5

Entwurf

Durch den Einsatz von JMX ist bereits die Verwaltung der MBeans geregelt. Neben der einfachen Verwahrung beinhaltet das auch eine Anfragemöglichkeit, die mit einer einfachen SQL-Anfrage zu vergleichen ist. Man kann sich beispielsweise alle Drucker-MBeans geben lassen, die einen Laserdrucker repräsentieren.

Hinzu kommt, dass MBeans netzwerktransparent sind, so dass sich die zu überwachende Applikation nicht auf dem selben Rechner wie der Servlet-Container befinden muss, lediglich ein schlanker MBean-Agent ist nötig. Das ist insbesondere wichtig, wenn über eine Web-Oberfläche mehrere Anwendungen überwacht werden sollen.

Betrachtet man die JMX-Architektur in Abbildung 5.1 auf der nächsten Seite, so übernimmt der JMX-Agent den kompletten Agent-Level und den Teil des Distributed-Services-Level, der die Netzwerktransparenz ermöglicht.

Es müssen nun noch zwei Lücken gefüllt werden: Zum einen müssen die einzelnen Werte der zu überwachenden Applikation geeignet angezeigt werden, was dem Distributed-Services-Level in der JMX-Architektur entspricht. Zum anderen müssen diese Werte über MBeans nach außen zugänglich gemacht werden, entsprechend dem Instrumentation-Level der JMX-Architektur.

Im Distributed-Services-Level gilt es, eine Möglichkeit zu bieten, in einer JSP-Seite Werte über JMX zu laden und auf verschiedenste Weise anzuzeigen - sei es als Graph, als Tabelle oder einfach nur als reiner Text. Da JSP-Seiten oft von Webdesignern entworfen werden, ist es am komfortabelsten, all diese Aufgaben mit Hilfe einer Tag-Library zu lösen, welche über die für Webdesigner gewohnte Tag-Notation angesprochen werden kann. Die JSP-Seite kann so von Java-Code frei gehalten werden und vereint HTML-

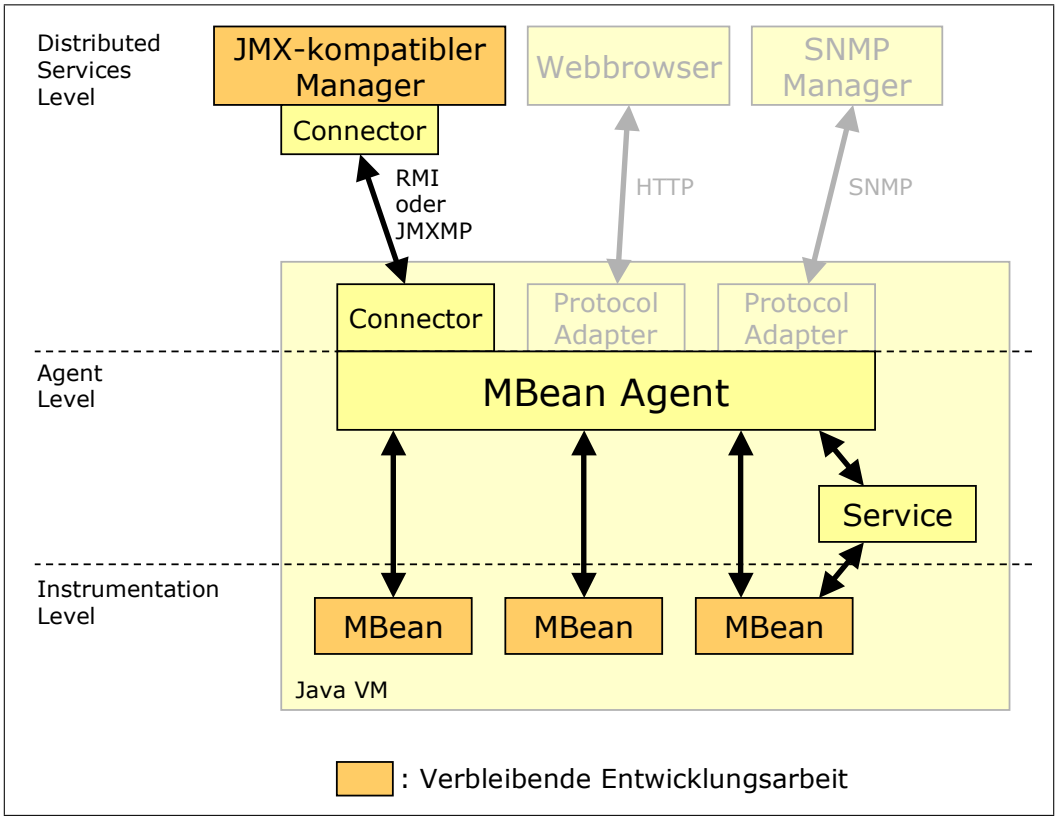

Abbildung 5.1: Verbleibende Entwicklungsarbeit aus Sicht der JMX-Architektur

Tags mit Tag-Library-Tags auf eine einheitliche Weise. Diese Tag-Library wird im weiteren Verlauf der Arbeit als „MLib“ bezeichnet.

Im Instrumentation-Level fallen eine Vielzahl von Problemen an, die sehr stark von der zu lösenden Aufgabe abhängen. So müssen hier Shell-Skripten aufgerufen und deren Ausgabe geparst werden, es müssen Daten vom SAP-System gelesen oder oft benötigte JMX-Abläufe vereinfacht werden.

All diese Hilfestellungen für die MBean-Entwicklung kann man sehr gut in einer modular aufgebauten Bibliothek unterbringen. Wobei jedes Modul in einem eigenen Unterpaket abgegrenzt ist. Diese Bibliothek wird im Folgenden als „MTools“ bezeichnet.

5.1 InterfaceCockpit

Das InterfaceCockpit repräsentiert den Teil, der bei jeder Entwicklung einer neuen Management-Applikation geleistet werden muss. Ein Ziel dieser Diplomarbeit ist es, diesen Aufwand zu minimieren. Diese Entwicklungsarbeit teilt sich in zwei Ebenen: Die MBean-Ebene und die Web-Applikations-Ebene.

Auf MBean-Ebene (Instrumentation-Level) wird sehr viel Arbeit von den MTools abgenommen. Diese übernehmen alle Aufgaben, die sich für den jeweiligen Ressourcentyp verallgemeinern lassen. Entwurfsarbeit ist seitens des InterfaceCockpit für diese Ebene nicht notwendig, da es nur die Konzepte der MTools anwendet.

Auf der Web-Applikations-Ebene (Distributed-Services-Level) wird die Verwaltung und die eigentliche Darstellung der MBeans und ihrer Werte von der MLib übernommen. Das Screendesign und die Benutzerführung hängen jedoch von der konkreten Anwendung ab. Der Entwurf dieser obersten Schicht soll nun vorgestellt werden.

Das InterfaceCockpit teilt sich in drei Bereiche (siehe Abbildung 5.2 auf der nächsten Seite):

- Einen allgemeinen Teil, der sich mit dem InterfaceCockpit selbst befasst. Dieser Teil deckt Hilfe und Einstellungen ab.
- Einen Mapping-Teil, der sich mit den Mappings und den Mapping-Filtern befasst. Die Filter werden später erläutert.
- Ein Ressourcen-Teil, der dem Administrator Auskunft über den Zustand des Mercator-Servers gibt.

Hinweis zur Abbildung 5.2 auf der nächsten Seite: Im InterfaceCockpit werden „Mappings“ als „Schnittstellen“ bezeichnet. Im Rahmen dieser Ausarbeitung wurde wegen der Verwechslungsgefahr zu Programmierschnittstellen die englische Bezeichnung gewählt. Die Menüpunkte heißen deshalb „Schnittstellen-Übersicht“ und „Schnittstellen-Filter“, anstatt „Mapping-Übersicht“ und „Mapping-Filter“.

Das Screendesign nimmt das „Look and Feel“ der Intranet-Anwendungen von dm-drogerie markt auf und nutzt die bei Webanwendungen häufig genutzte T-Form. Im linken Bereich befindet sich die Navigation, die in die genannten drei Teile gruppiert ist. Im oberen Bereich wird der Pfad angezeigt, in dem sich der Benutzer innerhalb des Navigationsbaums gerade befindet. Rechts oben wird das System angezeigt, das der Benutzer gerade betrachtet. Dort kann auch auf ein anderes System gewechselt werden. In der Mitte wird der eigentliche Inhalt der aktuellen Seite angezeigt.

Der Login zum InterfaceCockpit passiert automatisch. Eine bei dm-drogerie markt bereits im Einsatz befindliche ActiveX-Komponente erkennt den gerade angemeldeten

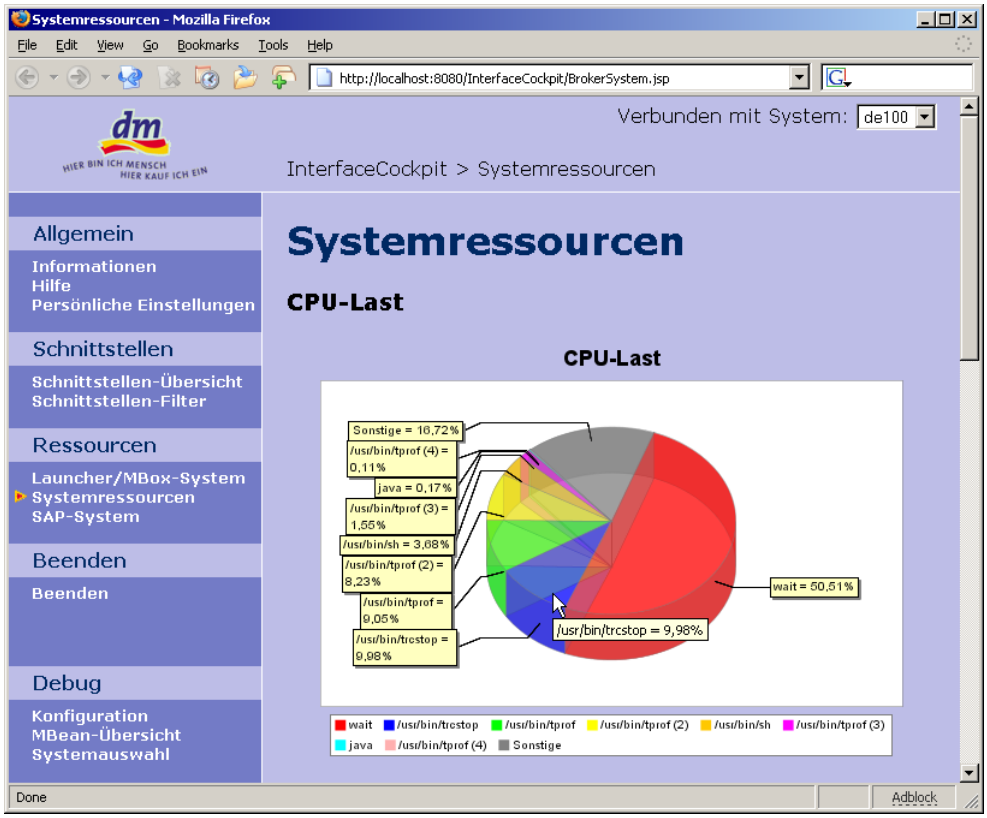

Abbildung 5.2: Das InterfaceCockpit.

Windows-User und meldet diesen der Web-Applikation. Dieser Vorgang läuft für den Anwender vollkommen transparent ab, so dass er sofort die Mapping-Übersicht zu sehen bekommt.

Eine besondere Rolle spielen die Mapping-Filter. Damit der Benutzer in der Mapping-Übersicht nicht jedes Mal alle 200 Mappings zu Gesicht bekommt und daraus „seine“ Mappings heraussuchen muss, sieht das InterfaceCockpit einen flexiblen Filtermechanismus vor. Mit Hilfe eines Filters kann man die Menge der angezeigten Mappings auf bestimmte Mappings reduzieren. Zusätzlich besteht die Möglichkeit, nur solche Mappings anzuzeigen, die gerade aktiv bzw. inaktiv sind oder deren Zustand kritisch bzw. unkritisch ist. So kann sich jeder Mapping-Betreuer einen oder mehrere Filter zusammenstellen, die die Ansicht auf „seine“ Mappings beschränken. In den Einstellungen kann jeder Anwender einen Filter wählen, der nach dem Login vorausgewählt werden soll.

5.2 MLib

Technologien, die es erlauben, in kurzer Zeit wiederverwendbare und leicht wartbare JSP-Seiten zu erstellen, gibt es bereits. Als Beispiel seien hier Struts und Java Server Faces genannt.

Wenn es gelingt, eine Tag-Library zu entwickeln, die sich voll auf das einfache und generische Laden und Darstellen von Objekten und ihren Werten konzentriert, so kann es dem Entwickler der Web-Anwendung überlassen werden, welche weiteren Technologien eingesetzt werden sollen. Die MLib soll also nicht die Lösung für alles sein, sondern sie soll vielmehr in der Lage sein, sich modular in bereits bestehende und bewährte Konzepte einzufügen.

Die MLib bietet daher eine Möglichkeit, Objekte von verschiedenen Quellen so zu abstrahieren, dass sie auf die selbe Art und Weise angesprochen werden können. Darauf bauen dann schließlich Tags auf, die Werte eines solchen Objekts auf unterschiedliche Art und Weise anzeigen können.

Im weiteren Verlauf dieser Diplomarbeit werden Attribute von Tag-Library-Tags als „Parameter“ bezeichnet, um sie von Objekt-Attributen besser zu unterscheiden.

Außerdem werden Attributwerte eines Objekts und Rückgabewert einer Operation eines Objekts unter dem Begriff „Objektwert“ zusammengefasst.

5.2.1 ObjectWrapper

Da die MLib sich voll auf die Darstellung von Werten konzentrieren können soll, gilt es zunächst, eine Abstraktion für Daten zu entwickeln, die sich auf möglichst beliebige konkrete Objekte und Objektstrukturen abbilden läßt.

Hinter einem solchen Objekt kann sich letztendlich eine einfache Datenstruktur (z. B. ein `CompositeData`-Objekt oder Hashtabelle), eine MBean oder auch ein beliebiges Java-Bean verbergen.

Der Zugriff auf die Attribute und Operationen von konkreten Objekten hängt jedoch vom jeweiligen Objekttyp ab. So werden bei Java-Beans beispielsweise Attribute über Getter und Setter gelesen bzw. geschrieben und Methoden durch direkte Methodenaufrufe ausgeführt. Bei MBeans hingegen läuft der Zugriff generisch über Methoden der Schnittstelle `MBeanServerConnection` (siehe Abbildung 5.3 auf der nächsten Seite).

Um den Zugriff zu vereinheitlichen, muss daher ein Objekt-Wrapper dazwischengeschalten werden. Dieser handelt nach dem Entwurfsmuster „Wrapper“ (auch als „Proxy“

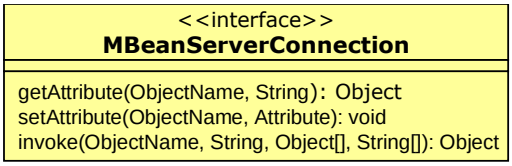

Abbildung 5.3: Die Schnittstelle MBeanServerConnection

bekannt). Er nimmt also alle Anfragen an das Objekt stellvertretend entgegen und richtet sie dann an das eigentliche Objekt unter Beachtung der jeweiligen Eigenarten des Objekttyps. Eine Abfrage eines Attributwerts wird so bei einem Java-Bean durch den Aufruf eines Getters durchgeführt, bei einer MBean durch den Aufruf der Methode `getAttribute` (siehe Abbildung 5.3) und bei einer Hashtabelle durch Lesen des entsprechenden Wertes.

Ein ObjectWrapper hat außerdem die Möglichkeit, neben den „normalen“ Attributen eines Objekts, sog „Wrapper-Attribute“ anzubieten. Dies sind Attribute, die nicht vom Objekt, sondern vom Wrapper angeboten werden, um beispielsweise eine Möglichkeit zu bieten, weitere Informationen über das Objekt abzufragen. Objekte des Typs `MBean` haben z. B. das Wrapper-Attribut `(name)`, das den eindeutigen Namen (die ID) der MBean verrät.

Um Mengen von Objekten abzubilden, können Listen von ObjectWrappern gebildet werden. Eine Tabelle läßt sich beispielsweise durch eine Liste von ObjectWrappern darstellen, die jeweils eine Zeile repräsentieren. Die MLib kennt zwei Arten von Tags: Manche arbeiten mit einzelnen ObjectWrappern, andere mit ganzen Listen von ObjectWrappern. Beispiele hierzu siehe Anhang A auf Seite 56.

5.2.2 Lebenszyklus von ObjectWrappern

Bevor Daten eines ObjectWrapper angezeigt werden können, muss erst der ObjectWrapper selbst erzeugt und geladen werden. Der Ablauf muss also der folgende sein: Erst laden, dann darstellen.

Ein Wrapper wird beim Laden im sog. „Seitenkontext“ (Page Context) abgelegt. Dieser wird vom Servlet-Container bereitgestellt, und kann Daten aufnehmen, die temporär zum Aufbau einer Antwortseite benötigt werden. Der Seitenkontext ist von allen anderen Tags der selben Seite aus erreichbar.

Gerade wenn man mit ganzen Listen von ObjectWrappern arbeitet, möchte man die Liste bevor man sie darstellt noch manipulieren können, d. h. sie sortieren, um weitere

ObjectWrapper erweitern oder ähnliches. Zwischen dem Laden und der Darstellung muss daher noch ein optionales Manipulieren möglich sein.

Da es wahrscheinlich ist, dass von einem ObjectWrapper viele Werte angezeigt werden sollen, wäre es unkomfortabel, wenn man bei jedem Wert neben dem Attributenamen bzw. dem Namen und den Parametern der Methode auch noch jedes Mal den ObjectWrapper angeben müsste. Es macht daher Sinn, eine Art Selektionsmechanismus vorzusehen. D. h. man kann einen ObjectWrapper selektieren, so dass alle Leseoperationen auf diesen ObjectWrapper bezogen werden - Es sei denn, man gibt explizit einen anderen an. Der Ablauf erweitert sich dadurch um: Laden, evtl. Manipulieren, Selektieren und dann Darstellen (siehe Abbildung 5.4).

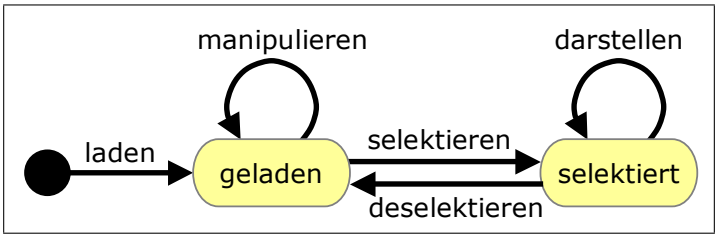

Abbildung 5.4: Zustandsdiagramm ObjectWrapper

5.2.3 Erzeugung von ObjectWrappern

Wie bereits in Abschnitt 5.2.1 auf Seite 31 gesagt, wird die unterschiedliche Art und Weise, wie auf Attribute und Operationen bestimmter Objekttypen zugegriffen wird, von ObjectWrappern vereinheitlicht. Da diese Wrapper in der MLib eine zentrale Rolle spielen, soll nun zunächst darauf eingegangen werden, wie für ein vorhandenes Objekt der passende ObjectWrapper erzeugt wird. Dieser Schritt wird zur Laufzeit von den Tags vollkommen transparent durchgeführt, so dass der Entwickler der Management-Applikation sich nicht darum kümmern muss.

Zur Erzeugung von ObjectWrappern wird eine Erweiterung des Fabrik-Entwurfmusters genutzt. Zur Erinnerung: Eine Fabrik ist ein Singleton mit mindestens einer Fabrikmethode. Ein Singleton ist eine Klasse, die nur eine Instanz hat. Eine Fabrikmethode ist eine Methode, die ein Objekt erzeugt, wobei je nach gegebenen Parametern oder durch Ableitung Instanzen verschiedener Klassen erzeugt werden können, was beim direkten Aufruf eines Konstruktors nicht möglich ist.

Die `ObjectWrapperFactory` bietet zwar die Fabrikmethode, die eigentliche Erzeugung wird jedoch an sog. `ObjectWrapperBuilder` delegiert. Ein Builder hat zwei Me-

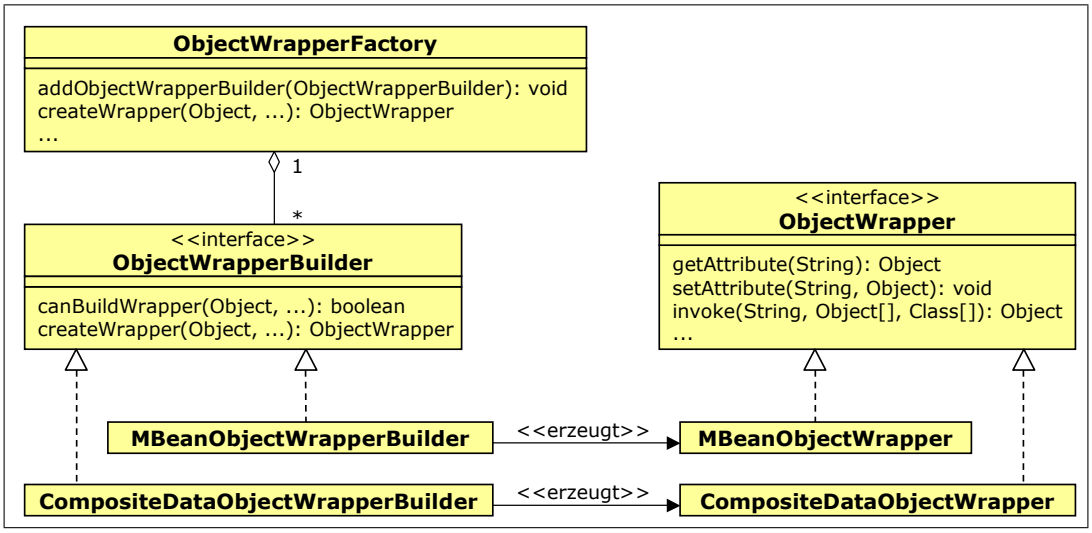

Abbildung 5.5: Klassendiagramm zur ObjectWrapperFactory

thoden: Die erste kann für ein gegebenes Objekt sagen, ob der Builder in der Lage ist, dafür einen Wrapper zu erzeugen. Die zweite erzeugt schließlich den Wrapper.

Für jeden unterstützten Objekttyp gibt es einen Builder. Momentan sind das die Typen `MBean` und `CompositeData`. Diese Builder müssen bei der Fabrik angemeldet werden (siehe Abbildung 5.5).

Die Erzeugung eines ObjectWrapper funktioniert nun wie folgt: Die Fabrik fragt einen Builder nach dem anderen, ob er einen Wrapper für das gegebene Objekt erzeugen kann. Der erste Builder, der die Erzeugung übernehmen kann, wird aufgefordert, dieses zu tun. Falls sich kein zuständiger Builder findet, wird das Objekt in einen `JavaObjectWrapper` gepackt, welcher Zugriff auf die Attribute und Operationen des Objekts nach dem Java-Bean-Standard ermöglicht.

Dank dieser Technik ist es möglich, dynamisch weitere Objekttypen zu unterstützen, ohne die vorhandene Implementierung ändern zu müssen. Es muss nur ein entsprechender Builder geschrieben werden, der dann bei der Fabrik angemeldet wird.

5.2.4 Tag-Syntax zum Lesen von Objektwerten

Viele Tags verarbeiten Werte von ObjectWrappern. Dabei kann es sich um einen Attributwert oder um den Rückgabewert einer Operation handeln. Was mit diesem Wert letztendlich passiert, hängt natürlich vom jeweiligen Tag ab. Das Tag `textRenderer`

gibt den Wert beispielsweise als einfachen Text aus, das Tag `loadObject` speichert ihn in den Seitenkontext.

Als Spezialfall sei das Tag `call` genannt: Es ruft eine Operation einfach nur auf und verwirft danach den Rückgabewert. Dieses Tag kann man einsetzen, wenn man eine Operation aufrufen will, die keinen Rückgabewert hat oder deren Rückgabewert nicht benötigt wird.

Um die Nutzung der Tags konsistent zu halten, ist die Syntax bei jedem dieser `ValueReaderTags` die gleiche. Diese Syntax soll nun im Folgenden vorgestellt werden.

```
<!-- Den Wert '5721' fest vorgeben -->
<m:aValueTag value="5721" />

<!--
 | Den Wert des Attributs 'ID' des gerade selektierten ObjectWrapper nutzen.
 +-->
<m:aValueReaderTag attribute="ID" />

<!--
 | Den Wert des Attributs 'Name' des ObjectWrappers nutzen, der unter dem
 | Schlüssel 'user' im Seitenkontext gespeichert ist.
 +-->
<m:aValueReaderTag attribute="ID" fromKey="user" />

<!--
 | Den Rückgabewert der Operation 'int calculateSize(String name, int max)' des
 | gerade selektierten ObjectWrapper verwenden.
 +-->
<m:aValueReaderTag>
  <m:operation name="calculateSize">
    <m:operationParam type="String" value="Otto"/>
    <m:operationParam type="int" value="1024"/>
  </m:operation>
</m:aValueReaderTag>
```

Quelltext 5.1: Syntax von ValueReaderTags

Die einfachste Möglichkeit, einem `ValueReaderTag` einen Wert zu geben, ist, einen festen Wert zuzuweisen. Das wird durch den Parameter `value` ermöglicht (siehe Quelltext 5.1, Zeile 2).

Will man einen Attributwert verwenden, muss man den Namen des Attributs mit dem Parameter `attribute` angeben. Optional kann man dabei den Schlüssel angeben, unter dem der ObjectWrapper im Seitenkontext gespeichert ist, dessen Attribut man lesen möchte (siehe Quelltext 5.1, Zeile 7 bzw. 13).

Um Rückgabewerte von Operationen zu nutzen, muss ein eingebetteter `operation`-Tag verwendet werden. Das ist nötig, weil eine Operation viele Parameter haben kann,

die jeweils als eingebettete `operationParam`-Tags angegeben werden (siehe Quelltext 5.1 auf der vorherigen Seite, Zeilen 19 bis 24). Auch beim `operation`-Tag kann über den optionalen Parameter `fromKey` der zu verwendende ObjectWrapper angegeben werden.

5.2.5 Die Tag-Typen der MLib

Die einzelnen Tags der MLib lassen sich analog zum im Abschnitt 5.2.2 auf Seite 32 vorgestellten Paradigma von Laden, Manipulieren, Selektieren und Darstellen in vier Kategorien einordnen: Lage-Tags, Manipulations-Tags, Selektions-Tags und Darstellungs-Tags.

Dabei wird jeweils zwischen Tags unterschieden, die einen einzelnen ObjectWrapper nutzen und solchen, die mit einer Liste von ObjectWrappern arbeiten.

Eine Auflistung der wichstigsten Tags mit Anwendungsbeispielen befindet sich in Anhang A auf Seite 56. Die Abbildung A.5 auf Seite 69 zeigt eine Übersicht über die Zusammenarbeit der Tags und Klassen der MLib.

Lade-Tags

Die Aufgabe der Lade-Tags ist die Erzeugung von ObjectWrappern und deren Bereitstellung im Seitenkontext. Dieser Vorgang wird im Folgenden als „Laden“ bezeichnet. Sie sind die einzigen Tags, die auf bestimmte Objekttypen spezialisiert sind. Sie abstrahieren ein bestimmtes Objekt mit Hilfe eines Wrappers, so dass alle weiteren Tags eine einheitliche Zugriffsmöglichkeit haben.

Manipulations-Tags

Manipulations-Tags ändern einen zuvor geladenen Wrapper bzw. eine Liste von Wrappern. Diese Manipulation kann beispielsweise das Sortieren oder Erweitern einer Liste sein.

Dazu nehmen sie den betreffenden Wrapper oder Liste aus dem Seitenkontext, manipulieren ihn und legen ihn an der selben Stelle wieder im Seitenkontext ab.

Selektions-Tags

Es gibt genau zwei Selektions-Tags: Eines für die Selektion eines einzelnen Object-Wrappers und eines für die sequentielle Selektion aller ObjectWrapper einer Liste.

Durch die Selektion kann sich der Entwickler der JSP-Seite die explizite Angabe des Wrapper bei jedem ValueReaderTag ersparen.

Die Selektion verläuft folgendermaßen:

1. Der momentan selektierte Wrapper wird zur späteren Wiederherstellung in einer Variablen zwischengespeichert.
2. Der angegebene Wrapper wird selektiert, indem er im Seitenkontext unter einem speziellen Selektionsschlüssel gespeichert wird.
3. Der Tag-Body wird abgearbeitet. Alle ValueReaderTags im Body können über den Selektionsschlüssel auf den selektierten Wrapper zugreifen.
4. Im schließenden Tag wird die letzte Selektion wieder hergestellt, d. h. der anfangs zwischengespeicherte Wrapper wird selektiert.

Durch das Speichern und Wiederherstellen der alten Selektion können Selektions-Tags beliebig geschachtelt werden.

Bei der Selektion einer Liste wird nacheinander jeder Wrapper in der Liste selektiert und der Body des Selektion-Tags einmal für diesen Wrapper abgearbeitet. Innerhalb des Tag-Body ist also jeweils nur ein Wrapper selektiert, nicht die gesamte Liste.

Darstellungs-Tags

Unter diese Kategorie fallen die meisten Tags. Dabei werden Darstellungs-Tags, die einzelne Wrapper nutzen als „Renderer“ bezeichnet. Tags, die mit ganzen Listen arbeiten, werden „Builder“ genannt.

Renderer stellen meist einen Wert eines Wrappers dar, sei es als einfacher Text, als auf eine bestimmte Weise formatierte Zahl oder als Bild. Diese Tags sind als ValueReader-Tags realisiert.

Da Builder ganze Listen von Wrappern darstellen können, erzeugen sie automatisch komplexere Widgets, wie Tabellen oder Graphen. Ähnlich wie das Listen-Selektions-Tag iterieren sie dabei durch die Liste und fügen für jeden Wrapper ein Element zum

Widget hinzu. Ein solches Element kann im Falle einer Tabelle eine Zeile, bei einem Tortendiagramm ein Tortenstück und bei einem Liniendiagramm ein Punkt der Linie sein.

Um die dazu nötigen Werte letztendlich auszulesen, werden eingebettete ValueReader-Tags genutzt.

Jedes Mal, wenn ein Wert von einem Wrapper gelesen wird, sei es durch einen Renderer oder innerhalb eines Builder, passiert dies durch ein ValueReaderTag. Dadurch wird eine einheitliche und konsistente Nutzung nach dem im Abschnitt 5.2.4 auf Seite 34 beschriebenen Prinzip sichergestellt.

Dadurch ist eine hohe Wartbarkeit und eine schnelle Entwicklung von JSPs gewährleistet, weil der Entwickler einer JSP immer nach dem selben Muster vorgehen kann. Außerdem wird eine geringe Fehleranfälligkeit ermöglicht, da bei der Implementierung eine gemeinsame abstrakte Oberklasse genutzt werden kann, wodurch Code-Redundanz vermieden wird.

5.3 MTools

In den vorigen Abschnitten wurde gezeigt, wie es mit Hilfe der MLib möglich ist, Werte von MBeans oder anderen Objekten auf verschiedenste Weise anzuzeigen. Im Folgenden wird nun der zweite Teil des Frameworks vorgestellt: Die MTools.

Der Aufwand des Schreibens einer MBean hängt sehr stark von der Herkunft der Werte ab, sprich von der zu überwachenden Applikation. Handelt es sich um eine Java-Applikation, so liegen die Werte meist bereits vor, sie müssen nur z. B. über einen Methodenaufruf abgefragt werden. Eine MBean zu schreiben, die dieses macht, ist entsprechend einfach.

Kommen die Werte jedoch von der Ausgabe eines Shell-Skriptes oder vom SAP-System, dann muss erst die umgeleitete Ausgabe des Skriptes geparst werden bzw. ein enfernter Funktionsaufruf (RFC) zum SAP-System durchgeführt werden.

Die MTools haben daher die Aufgabe, die Entwicklung von MBeans möglichst weitgehend zu vereinfachen. Um die einzelnen Problemgruppen klar voneinander abzugrenzen, soll für jedes Aufgabengebiet ein eigenes Unterpaket entstehen. So besteht auch die Möglichkeit, die MTools zu einem späteren Zeitpunkt ohne großen Aufwand in einzelne Module aufzuteilen, wenn der wachsende Umfang es erfordern würde.

Die Unterstützung der MBean-Entwicklung kann verschieden stark vorrangetrieben werden. Die erste Stufe wäre eine Hilfsklasse, die statische Methoden für verschiedene,

häufig gebrauchte Prozeduren enthält. Die zweite Stufe wäre eine Sammlung von Hilfsklassen, die eine ganze Teilaufgabe übernehmen. Die dritte Stufe könnte schließlich die gesamte Implementierungsarbeit abnehmen, indem sie die Beschreibung von MBeans über eine Konfigurationsdatei ermöglicht. Diese Datei würde beim Laden des Agents genutzt, um vollkommen generisch die gewünschten MBeans zu erzeugen.

5.3.1 ConfigMBeans

Eine wichtige Rolle bei der Entwicklung einer Anwendung spielt die Konfiguration. Sie legt wichtige Parameter fest und passt eine Anwendung an die lokalen Besonderheiten an.

Das Einlesen der Konfiguration ist ein typisches Bootstrap-Problem: Da die Applikation beim Start zunächst einmal nichts weiss, muss es eine feste Anlaufstelle geben, an der die Konfiguration abgelegt ist. In den meisten Fällen wird hierzu eine Konfigurationsdatei mit einem festen Namen genutzt. Im Falle der MLib muss darin zumindest abgelegt werden, wie und wo welcher MBean-Agent zu erreichen ist. Theoretisch könnten dann alle weiteren Informationen von MBeans bezogen werden.

Das würde jedoch bedeuten, dass die Konfiguration nicht mehr zentral an einer Stelle wäre, sondern in eine Grundkonfiguration und eine über MBeans bezogene Konfiguration aufgeteilt wäre. Außerdem muss die Konfiguration irgendwo persistent abgelegt werden, was dann die MBeans übernehmen müssten. Aus diesem Gesichtspunkt wäre es also besser, wenn sich die gesamte Konfiguration zentral in einer Datei befände.

Auf der anderen Seite wäre es wiederum sehr hilfreich, wenn die Konfiguration über MBeans erreichbar wäre, weil man sie dann innerhalb der Webanwendung anzeigen und ändern könnte.

Warum also nicht beide Vorteile vereinen? Warum nicht eine Konfiguration bieten, die in einer festen lokalen Datei abgelegt ist, die über generisch erzeugte MBeans les- und änderbar ist? Genau das bieten die `ConfigMBeans`.

`ConfigMBeans` sind eine Stufe-Drei-Unterstützung für die Konfiguration, d. h. es muss kein Java-Code geschrieben werden, um sie einzusetzen. Die Erstellung einer XML-Konfigurationsdatei reicht aus. Und wie sich zeigen wird, ist noch nicht einmal das unbedingt nötig.

Eine `ConfigMBean` hält die Daten für ein bestimmtes Objekt, also beispielsweise für einen Benutzer oder für einen MBean-Agent.

```
<?xml version="1.0"?>
<configuration>
  <typeList>
    <type name="MBeanAgent" description="A connection to a MBean agent">
      <attribute name="Host" type="String" defaultValue="null" isKey="false"
                 description="The host name where the agent is running" />
      <attribute name="Protocol" type="String" defaultValue="null" isKey="false"
                 description="The protocol to use to connect to the agent" />
      <attribute name="AgentName" type="String" defaultValue="null" isKey="true"
                 description="The name to use to identify the agent in JSPs" />
      <attribute name="Port" type="Integer" defaultValue="null" isKey="false"
                 description="The port where the agent is running" />
    </type>
    ...
  </typeList>

  <MBeanAgentList>
    <MBeanAgent>
      <Host>localhost</Host>
      <Protocol>JmxMp</Protocol>
      <AgentName>mercator</AgentName>
      <Port>5001</Port>
    </MBeanAgent>
    ...
  </MBeanAgentList>
  ...
</configuration>
```

Quelltext 5.2: Auszug aus einer XML-Konfiguration von ConfigMBeans

Jede `ConfigMBean` ist einem Typ zugeordnet, der Schlüsselattribute und normale Attribute definiert. Für die Beschreibung eines MBean-Agent käme als Schlüsselattribut z. B. ein Name, als normale Attribute Hostname, Port und Protokoll in Frage.

Eine `ConfigMBean` wird durch ihren Typ und die Schlüsselattribute eindeutig bestimmt, z. B.: `Type=MBeanAgent,Name=mercator`. Aus diesem Grund sind Schlüsselattribute nur lesbar, während normale Attribute sowohl les- als auch schreibbar sind.

Als Attributwerte können beliebige BasicTypes genutzt werden (siehe Abschnitt 2.6.3 auf Seite 12).

Die Attributwerte werden, wie bereits erwähnt, in einer XML-Datei abgelegt (siehe Quelltext 5.2). Der Zugriff auf diese Datei ist durch eine abstrakte Persistenzschicht gekapselt, so dass es jederzeit möglich ist, die Daten in einer Datei mit einem anderen Format oder auch in einer Datenbank abzulegen.

Die zentrale Verwaltung wird von einer MBean mit dem Namen `Configuration` übernommen. Diese MBean liest bei ihrer Erzeugung die Konfiguration von der Persistenzschicht und legt die darin definierten `ConfigMBeans` an.

Wird der Wert einer `ConfigMBean` geändert, so wird diese Änderung an die Persistenzschicht weitergegeben, die sie in unserem Fall in die XML-Datei schreibt - nicht ohne vorher eine Sicherung der alten Konfiguration angelegt zu haben. Auf diese Weise werden die letzten fünf Stände gesichert.

Neben der initialen Erzeugung der `ConfigMBeans` können über die `Configuration`-MBean auch zur Laufzeit neue Typen, neue Attribute und neue `ConfigMBean`-Instanzen erzeugt werden. Da es sich um eine MBean handelt, sind all diese Operationen auch von der Web-Anwendung aus möglich.

Mit den ConfigMBeans erhält man also eine Konfiguration, die lokal persistent abgelegt ist und die sich vollständig in die JMX-Architektur integriert. Durch diese Integration ist die Konfiguration automatisch von einer MLib-basierten Web-Anwendung nutzbar, ohne dass dafür konfigurationsspezifischer Code nötig wäre.

Die MLib legt beim Start automatisch einen lokalen MBean-Agent mit einer `Configuration`-MBean an, die die Konfiguration aus der Datei `config.xml` liest. Damit ist auch das erwähnte Bootstrap-Problem gelöst.

5.3.2 SAP-MBean-Mapping

Wie in vielen anderen großen Unternehmen so spielen auch bei dm-drogerie markt SAP-Systeme eine zentrale Rolle. Sehr weite Teile der Unternehmensplanung und -steuerung werden über SAP abgewickelt. Es ist daher essentiell, eine einfache und zugleich leistungsfähige Zugangsmöglichkeit zu Daten aus dem SAP-System zu bieten.

Um Daten vom SAP-System nach außen hin anzubieten werden sog. „remotefähige Funktionsbausteine“ eingesetzt. Diese sind dann über Remote Function Calls (RFC) erreichbar. Im Zuge ihres Engagements im Bereich Java, hat die SAP AG den Java-Connector (kurz: JCo) entwickelt, mit dessen Hilfe von Java aus auf Funktionsbausteine zugegriffen werden kann.

Funktionsbausteine sind jedoch prozedural, MBeans objektorientiert. Funktionsbausteine sind globale Funktionen innerhalb des SAP-Systems, die Tabellen und flache Strukturen (Records) als Eingabe- und Ausgabeparameter nutzen. MBeans dagegen repräsentieren jeweils eine Ressource mit Attributen und Operationen.

Die Abbildung von MBeans auf SAP-Funktionsbausteine

Die einfachste Möglichkeit, SAP-Funktionsbausteine über JMX zugänglich zu machen, wäre, eine SAP-MBean anzubieten, die alle gewünschten Funktionsbausteine als Operationen enthält. Das widerspräche jedoch der Natur einer MBean, da die SAP-MBean nicht eine Ressource, sondern eine Quelle von Ressourcen repräsentieren würde. Man würde dadurch die einfache Handhabung von Ressourcen aufgeben, man könnte auch nicht mehr das Schema „Laden, Manipulieren, Selektieren und Darstellen“ anwenden. Es würde im Laufe der Zeit für jedes SAP-System ein wahres Monstrum von MBean entstehen.

Es wäre also wünschenswert, wenn die Ressourcen im SAP-System auch jeweils von einer MBean repräsentiert würden. Dazu wäre eine Abbildung von MBeans auf Funktionen nötig, also quasi ein „objekt-prozedurales Mapping“. Da abzusehen ist, dass bei dm-drogerie markt die meisten MBeans Ressourcen aus dem SAP-System darstellen werden, kann dieser Abbildung eine zentrale Rolle innerhalb der MTools zugesprochen werden. Das Mapping sollte daher eine Stufe-Drei-Unterstützung bieten, das Mapping sollte also komplett in einer XML-Datei beschreibbar sein.

In dieser XML-Datei müssen die Verbindungseinstellungen zu den einzelnen SAP-Systemen und die eigentlichen Abbildungen für jeden Ressourcentyp beschrieben werden.

Wie kann eine solche Abbildung nun aussehen? Prinzipiell gibt es folgende Probleme zu lösen:

- Es müssen die einzelnen Instanzen identifiziert werden.
- Attributwerte müssen gelesen werden können.
- Attributwerte müssen geändert werden können.
- Operationen müssen ausgeführt werden.

Jede dieser Aufgaben kann von jeweils einem Funktionsbaustein gelöst werden. Damit die Abbildung einheitlich bleibt und nicht unnötig kompliziert wird, bietet es sich an, für jede dieser Aufgaben ein Entwurfsmuster zu definieren. Ein Funktionsbaustein, der diese Aufgabe übernehmen will, muss sich an die Vorgaben des Entwurfsmuster halten. Die einzelnen Entwurfsmuster sollen nun vorgestellt werden.

Der **Initialisierer** hat die Aufgabe, die einzelnen Ressourcen-Instanzen zu identifizieren. Er wird zur Laufzeit dazu genutzt, für jede Instanz eine MBean mit den dazugehörigen Schlüsselattributen anzulegen.

Ein Initialisierer braucht dazu keine Eingangsparameter und stellt als Ausgabe eine Tabelle bereit, die die Schlüsselattribute als Spalten und die Instanzen als Zeilen enthält. Dabei können weitere Attributwerte als Spalten mitgegeben werden, um damit schon einmal den Cache zu befüllen.

Der **Attribut-Getter** hat die Aufgabe, die Werte eines oder mehrerer Attribute einer Instanz zurückzugeben.

Als Eingangsparameter können beliebige Attributwerte übergeben werden, das schließt insbesondere Schlüsselattribute ein, die der Getter benötigt, um die Instanz zu identifizieren, auf die sich die Anfrage bezieht. Um zyklische Abfragen von Gettern zu vermeiden, werden hierzu jedoch lediglich Werte aus dem Cache genutzt. Es können also nur langlebige Attributwerte genutzt werden.

Als Ausgangsparameter liefert der Getter die gewünschten Attributwerte.

In der XML-Beschreibung kann zu jedem Getter ein maximales Alter angegeben werden. Dieses Alter gibt an, wie lange die Attributwerte im Cache behalten werden sollen, bevor sie erneut abgefragt werden.

Der **Attribut-Setter** hat die Aufgabe, einen Attributwert einer Instanz zu setzen.

Als Eingangsparameter kann der Setter, wie bereits beim Getter beschrieben, beliebige Attributwerte annehmen. Zusätzlich bekommt er noch den neuen Attributwert übergeben. Ausgangsparameter kennt der Setter keine.

Die **Operation** hat die Aufgabe, eine Operation einer Instanz auszuführen.

Sie kann dazu als Eingangsparameter wie der Attribut-Getter und -Setter beliebige Attributwerte entgegen nehmen. Zusätzlich können beliebig viele weitere Parameter mitgegeben werden. Diese Parameter müssen dann vom Manager beim Aufruf der Operation mitgegeben werden. Als Ausgabeparameter kann entweder eine Tabelle, eine Struktur (Record) oder ein Wert aus einer Struktur (Skalar) zurückgegeben werden.

Die Erläuterung der XML-Beschreibung finden Sie in Anhang B auf Seite 70.

Die Architektur auf JMX-Seite

Im vorigen Abschnitt wurde gezeigt, wie es möglich ist, MBeans auf Funktionsbausteine abzubilden und wie diese Funktionsbausteine auf SAP-Seite auszusehen haben. Im Folgenden wird erläutert, wie diese Abbildungen genutzt werden, wie die entprechenden MBeans entstehen und wie sie arbeiten.

Die Abbildungen und die Verbindungen zu den SAP-Systemen werden von einem zentralen Singleton verwaltet, dem `SapMappingManager`. Dieser liest beim Start des MBean-Agent die XML-Beschreibung ein und erzeugt daraus für jedes SAP-System einen Connection-Pool und für jede Abbildung einen SAP-MBean-Typ. Die Attribute und Operationen eines SAP-MBean-Typs definieren sich wie folgt:

- Für jedes Attribut des Initialisierers wird ein Schlüsselattribut, bzw. ein normales Attribut angelegt, je nach dem ob der Parameter `isKey` gesetzt wurde.
- Für jeden Attribut-Getter und -Setter werden alle darin enthaltenen Attribute angelegt.
- Alle Operationen werden angelegt.

Der SAP-MBean-Typ bezieht also alle Informationen über Attribute und Operationen von den entsprechenden Funktionsabbildungen. Ist ein Attribut im Initialisierer oder in einem Getter enthalten, so ist es lesbar. Ist ein Attribut in einem Setter enthalten, so ist es schreibbar. Ist beides der Fall, so ist es les- und schreibbar.

Alle Typinformationen und Beschreibungen werden dabei vom jeweiligen Funktionsbaustein im SAP-System abgefragt. Es ist daher nicht nötig, in der Mapping-Datei anzugeben, von welchem Datentyp ein Attribut ist.

Nachdem alle SAP-Connection-Pools eingerichtet und alle SAP-MBean-Typen angelegt sind, wird von jedem SAP-MBean-Typ der Initialisierer aufgerufen. Für jede Instanz einer Ressource, die vom Initialisierer geliefert wird, wird eine SAP-MBean angelegt.

Nach dieser Initialisierungsphase existiert also für jede Ressource im SAP-Sytem eine stellvertretende SAP-MBean. Hierbei zeigt sich auch ein Nachteil der Abbildung: Da es keine Möglichkeit gibt vom SAP-System aus eine Verbindung zum MBean-Agent aufzubauen (Push-Verfahren), müssen alle Aktionen vom MBean-Agent ausgehen (Pull-Verfahren). Das betrifft insbesondere den Fall, wenn sich die Menge der Ressourcen-Instanzen ändert. Wird im SAP-System eine Ressource hinzugefügt, oder fällt eine weg, so bekommt der MBean-Agent dieses nicht mit. Die entsprechende SAP-MBean existiert daher immer noch, bzw. eine SAP-MBean fehlt.

Der `SapMappingManager` ist daher selbst auch eine MBean. Über die Operation `refresh` wird der Bestand der SAP-MBeans überprüft, um ggf. MBeans zu entfernen bzw. neue zu erstellen, falls die jeweilige Ressource im SAP-System entfernt bzw. neu hinzugekommen ist. Momentan muss diese Operation noch von Hand ausgelöst werden, in einer späteren Ausbaustufe wird sie jedoch automatisch in einem festen Intervall ausgeführt.

Über eine SAP-MBean ist eine Management-Applikation nun in der Lage, Attributwerte zu lesen und zu schreiben und Operationen aufzurufen. Für jede dieser Anfragen wird gemäß der Abbildung der jeweils richtige SAP-Funktionsbaustein aufgerufen.

Eine Ausnahme hierbei nimmt das Lesen von Attributwerten ein. Das Lesen von Attributwerten ist die häufigste Anfrage an MBeans. Damit nicht jedes Mal, wenn ein Attributwert gelesen werden soll, eine Anfrage beim SAP-System abgesetzt wird, bieten SAP-MBeans ein Attribut-Caching an. Nur wenn der Cache-Eintrag ein maximales Alter überschritten hat, wird tatsächlich auf den Attribut-Getter im SAP-System zugegriffen.

Dieses maximale Alter wird in der Mapping-Beschreibung eines Attribut-Getters mit dem Parameter `maxAge` in Sekunden angegeben. Für Stammdaten kann das Alter sehr hoch, z. B. auf einen Tag oder mehr, gesetzt werden, für Bewegungsdaten auf einen geringeren Wert, z. B. eine Minute.

5.3.3 Skriptunterstützung

Eine schlanke und einfache Möglichkeit an Daten heranzukommen, die mit Java-Mitteln nicht oder nicht einfach erreichbar sind, ist das Ausführen eines Kommandozeilenbefehls oder -skripts. Mit Hilfe eines solchen Kommandos könnte man z. B. betriebsystemnahe Daten, wie die aktuelle Prozessorlast abfragen.

Bei dm-drogerie markt gibt es bereits vorhandene, fertige Administratorskripten, die bestimmte Daten beipielsweise aus Konfigurations- oder Log-Dateien extrahieren. Mit einer Skriptschnittstelle könnte man diese einfach nutzen und müsste das Rad nicht neu erfinden. Dabei ist jedoch zu beachten, dass diese Lösung den Nachteil hat, dass dabei der Vorteil der Plattformunabhängigkeit von Java aufgegeben wird, da die Skripten im Allgemeinen nicht auf anderen Plattformen laufen.

Das einfache Ausführen eines Kommandos ist unter Java nicht besonders schwierig. Die Klasse `System` sieht dafür mehrere Methoden vor, mit deren Hilfe man ein Kommando samt Aufrufparameter ausführen kann.

Weniger trivial ist es jedoch, an die Ausgabe des Kommandos heranzukommen. Die MTools bieten daher eine Methode, die ein Kommando aufruft, die Ausgabe umleitet und als String-Array zurückgibt. Ein String in diesem Array entspricht einer Zeile der Ausgabe des Kommandos.

Mit Hilfe dieses Arrays ist es nun sehr einfach, die gewünschten Daten zu extrahieren. Im InterfaceCockpit wurden dazu reguläre Ausdrücke verwendet. Es handelt sich zwar nur um eine Stufe-Eins-Unterstützung, der verbleibende Programmieraufwand ist jedoch nicht sehr hoch.

5.3.4 MBean-Queries

Die JMX-Spezifikation [JMX] sieht für die Anfrage von Mengen von MBeans zwei Parameter vor: Den Objektnamen und ein Query-Expression-Objekt.

Der Objektname beinhaltet eine Domain und eine Liste von Key-Attributen, die auf eine MBean zutreffen müssen, damit sie in die Ergebnisliste aufgenommen wird. Dabei ist auch die Angabe von Wildcards möglich, um beispielsweise alle Domains oder alle Werte mancher Key-Attribute zuzulassen. Der Objektname `config:Type=User,*` findet beispielsweise alle Benutzer-MBeans des Agenten `config`.

Das Query-Expression-Objekt wird häufig als „die WHERE-Klausel einer MBean-Query“ bezeichnet. Es kann für jede MBean entscheiden, ob sie in die Ergebnisliste aufgenommen werden soll oder nicht. Dazu ruft der JMX-Agent für alle MBeans, die nach dem Objektnamenstest übrig geblieben sind, eine Methode des Query-Expression-Objekts auf. Diese Methode nimmt jeweils eine MBean entgegen und entscheidet dann, ob es diese MBean akzeptieren will oder nicht. Auf diese Weise kann bereits auf Agent-Seite die Liste der MBeans eingeschränkt werden, so dass Netzwerklast gespart wird.

Leider gibt es weder in der JMX-Implementierung von Sun noch in MX4J konkrete Implementierungen von Query-Expression-Objekten. Die MTools gleichen dies nun aus, und definieren vier Klassen von Query-Expression-Objekten:

- `AttributeQueryExp`: Entspricht einem Test auf Gleichheit. Bei der Erzeugung bekommt sie einen Attributnamen und einen Wert bzw. eine Liste von Werten gegeben. Bei der Anwendung prüft sie, ob der Wert des Attributs einem Wert aus der Liste entspricht.

- `OperationQueryExp`: Funktioniert analog zur `AttributeQueryExp` mit Operationen. Dazu bekommt sie bei der Erzeugung neben dem Operationsnamen eine Liste von Parametern gegeben.

- `NegatingQueryExp`: Entspricht einem logischen NICHT und kann ein Query-Expression-Objekt schachteln. Bei der Anwendung führt sie die geschachtelte Query-Expression aus und gibt das negierte Ergebnis zurück. Hat die geschachtelte Query-Expression eine konkrete MBean akzeptiert, so wird die `NegatingQueryExp` sie nicht akzeptieren und umgekehrt.

- `QueryExpGroup`: Entspricht einem logischen UND bzw. ODER. Bei der Erzeugung bekommt sie eine Liste von Query-Expression-Objekten und einen Flag, welcher bestimmt, ob sie als UND oder als ODER fungieren soll. Bei der Anwendung akzeptiert sie eine konkrete MBean je nachdem, wie der Verhaltens-Flag gesetzt wurde: Wenn entweder alle geschachtelten Query-Expressions diese MBean akzeptieren (UND) oder wenn eine Query-Expression diese MBean akzeptiert (ODER).

Mit Hilfe dieser Query-Expression lassen sich beliebig geschachtelte logische Ausdrücke erstellen. Dadurch, dass die Container-Query-Expressions `NegatingQueryExp` und `QueryExpGroup` beliebige Query-Expressions aufnehmen können, lassen sich beliebige Tests zusammenstellen.

Kapitel 6

Implementierung

Zur Implementierung wurde die Entwicklungsumgebung Eclipse in Verbindung mit einem Ant-Skript eingesetzt. Für die Formularvalidierung des InterfaceCockpit wurde das Framework Struts eingesetzt.

Getestet wurde auf dem schlanken Servlet-Container jetty und mit Hilfe von JUnit-Tests.

6.1 Die MLib im Einsatz

Um ein besseres Bild von der Flexibilität und von der einfachen Anwendung der MLib zu vermitteln, soll ihr Einsatz anhand zweier Beispiele gezeigt werden.

Die Beispiele zeigen, wie sich mit wenig JSP-Code eine recht komplexe Anzeige realisieren läßt.

6.1.1 Beispiel: Einfache Anwendung der MLib

Die Funktionsweise und das Zusammenspiel der MLib-Tags soll im Folgenden anhand eines Beispiels erläutert werden. Es soll eine Liste von CPUs gezeigt werden. Für jede CPU sollen die Top-Ten-Prozesse einmal als Tabelle ausgegeben und einmal als Tortendiagramm visualisiert werden, ähnlich wie es bereits in Abbildung 5.2 auf Seite 30 gezeigt wurde.

Die Nutzung der Tags verläuft stets nach dem im Abschnitt 5.2.2 auf Seite 32 vorgestellten Paradigma von Laden, Manipulieren, Selektieren und Darstellen.

```
<!-- Laden aller CPU-MBeans -->
<m:loadMBeanList toListKey="cpuList"
                 mbeanAgent="mercator"
                 mbeanQuery="*:type=CPU,*" />

<!-- Manipulieren der Liste. Hier: sortieren nach dem Namen -->
<m:sortList fromListKey="cpuList" attribute="Name"/>

<!-- Sequentielles Selektieren der CPUs in der Liste -->
<mlib:selectLoop fromListKey="cpuList">

  <!-- Darstellen des Attributs 'Name' -->
  <h3>CPU: <mlib:textRenderer attribute="Name"/></h3>

  ...

</mlib:selectLoop>
```

Quelltext 6.1: Anwendungsbeispiel der MLib

Im Quelltext 6.1 wird der erste Teil des Beispiels gezeigt, das Laden, Sortieren und Iterieren der CPU-MBeans.

In den Zeilen 3 bis 5 wird die Liste der CPU-MBeans geladen. Das Tag bedeutet, dass alle MBeans des Typs `CPU` vom MBean-Agent `mercator` im Seitenkontext unter `cpuList` gespeichert werden sollen. Das Tag führt dabei Folgendes aus: Es holt sich von der Konfiguration eine Verbindung zum MBean-Agent `mercator`. Bei diesem setzt es eine Query nach den gewünschten MBeans ab. Der MBean-Agent liefert daraufhin eine Liste mit den vollen Namen aller MBeans, die der Anfrage entsprechen. Aus dieser Namensliste wird eine Liste von `MBeanObjectWrappern` erzeugt und im Seitenkontext gespeichert.

Ab diesem Zeitpunkt werden die MBeans wie jeder andere ObjectWrapper behandelt. Alle nachfolgenden Tags wissen nicht, dass es sich um MBeans handelt und dass eine MBean entfernt angesprochen werden muss, dies wird vom `MBeanObjectWrapper` gekapselt. Es könnte sich genausogut um lokale Java-Objekte oder einfache Datenstrukturen handeln. Der Zugriff auf die Attribute und Operationen der MBean verläuft also vollkommen transparent. Tags zum Laden anderer ObjectWrapper sind in Abschnitt A.1 auf Seite 56 aufgeführt.

In der Zeile 8 findet eine Manipulation der gerade geladenen Liste statt. Sie soll nach dem Attribut `Name` sortiert werden. Das Tag nimmt die Liste aus dem Seitenkontext, sortiert sie und legt sie unter dem selben Schlüssel wieder im Seitenkontext ab.

In den Zeilen 11 bis 18 findet die Selektion statt. Das `selectLoop`-Tag iteriert durch eine Liste. Die CPU-MBeans in der Liste werden nacheinander selektiert und der Inhalt des `selectLoop`-Tag wird für jede CPU-MBean einmal abgearbeitet. Der jeweilige ObjectWrapper wird dabei im Seitenkontext unter einem bestimmten Schlüssel abgelegt. Die Anzeige-Tags können über diesen Schlüssel den momentan selektierten Wrapper abrufen und alle nötigen Werte davon abfragen.

In Zeile 14 findet eine einfache Darstellung statt. Das Attribut `Name` der selektierten CPU-MBean wird als Überschrift dargestellt.

In Zeile 16 erfolgt die Darstellung der Prozessliste einmal als Tabelle und einmal als Tortendiagramm. Dieser Schritt wird im nächsten Beispiel erläutert.

6.1.2 Beispiel: Fortgeschrittene Anwendung der MLib

In diesem fortgeschrittenen Beispiel soll nun eine Liste mit zehn Prozessen von der CPU-MBean geladen werden, die auf der CPU die größte Last erzeugen. Diese Prozesse sollen einmal als Tabelle und einmal als Tortendiagramm ausgegeben werden.

Die CPU-MBean hat für diesen Zweck eine Operation, die die Prozessliste zurückgibt. Die Operation bekommt dabei die maximale Länge der Liste übergeben. In unserem Fall also eine `10`. Die Liste enthält flache Strukturen, die jeweils die Daten über einen Prozess beinhalten (siehe Abbildung 6.1).

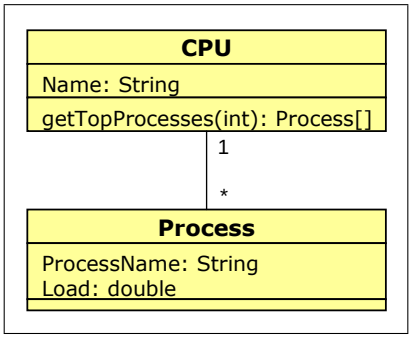

Abbildung 6.1: Klassendiagramm von CPU-MBean und Prozess-Struktur

Im Quelltext 6.2 auf der nächsten Seite wird gezeigt, wie die Prozessliste geladen und dargestellt wird. Dieser Quelltext baut auf dem Quelltext des ersten Beispiels auf und fügt sich an der Stelle ein, die dort mit drei Punkten gekennzeichnet ist (siehe Quelltext 6.1 auf der vorherigen Seite, Zeile 16).

```
<!-- Laden der Prozessliste -->
<m:loadList toListKey="processList">
  <m:operation name="getTopProcesses">
    <m:operationParam type="int" value="10" />
  </m:operation>
</m:loadList>

<!-- Sequentielles Selektieren der Prozesse in der Liste -->
<m:tableBuilder fromListKey="processList" cssClass="overview">
  <!-- Darstellen des gerade selektierten Prozesses als Tabellenzeile -->
  <m:column title="Prozessname" attribute="ProcName" />
  <m:column title="CPU-Last">
    <div align="right">
      <m:numberRenderer attribute="Load" pattern="0.00 %" />
    </div>
  </m:column>
</m:tableBuilder>

<!-- Sequentielles Selektieren der Prozesse in der Liste -->
<m:pieChartBuilder fromListKey="processList" title="CPU-Last"
                   fillSliceLabel="Sonstige" fillSliceValue="1">
  <!-- Darstellen des gerade selektierten Prozesses als Tortenstück -->
  <m:map to="label" attribute="ProcessName"/>
  <m:map to="value" attribute="CpuLoad"/>
</m:pieChartBuilder>
```

Quelltext 6.2: Fortgeschrittenes Beispiel

In den Zeilen 2 bis 6 wird die Liste geladen. Dazu wird vom gerade selektierten ObjectWrapper, in diesem Fall also der CPU-MBean, die Operation `getTopProcesses` mit einem Integer mit dem Wert `10` als Parameter aufgerufen. Der Rückgabewert der Operation (also die Prozessliste) wird im Seitenkontext unter `processList` abgelegt.

Auch hier muss das `loadList`-Tag nicht wissen, dass die Operation letztendlich bei einer MBean aufgerufen wird. Es könnte auch ein lokales Java-Objekt mit einer entsprechenden `getTopProcesses`-Methode oder ein beliebiges anderes Objekt sein.

In den Zeilen 9 bis 17 wird die Tabelle dargestellt. Alle Builder-Tags arbeiten ähnlich wie das `selectLoop`-Tag: Sie selektieren nacheinander alle ObjectWrapper einer Liste und arbeiten dabei jeweils ihren Tag-Inhalt einmal ab. Bei Builder-Tags muss jedoch der Tag-Body zum jeweiligen Builder-Tag passen. Die Builder-Tags übernehmen also die Selektion, ihr Body die Darstellung.

Im Falle des `tableBuilder`-Tags muss der Tag-Body aus `column`-Tags bestehen. Bei jedem Durchlauf des `tableBuilder`-Tags wird eine Tabellenzeile erzeugt, wobei jedes `column`-Tag jeweils die Zelle erzeugt, die zu ihrer Spalte gehört.

In den Zeilen 20 bis 25 wird das Tortendiagramm dargestellt. Ein `pieChartBuilder`-Tag fügt bei jedem Durchlauf ein Tortenstück zum Diagramm hinzu. Die Werte, die jeweils als Beschriftung und Größe der Tortenstücks verwendet werden sollen, werden durch `map`-Tags zugewiesen.

Kapitel 7

Ergebnisse

Das Framework ist sogar umfangreicher geworden, als es das InterfaceCockpit erfordert hätte, so gibt es beispielsweise eine veränderbare, persistente Konfiguration (siehe Abschnitt 5.3.1 auf Seite 39), ein Darstellungs-Tag für Tortendiagramme (siehe Abschnitt A.5 auf Seite 65) und zahllose weitere Detaillösungen, die vom InterfaceCockpit gar nicht genutzt werden, bzw. nicht angefordert waren.

Das InterfaceCockpit ist größtenteils ebenfalls fertiggestellt. Die Oberfläche ist vollständig fertiggestellt und es wird stichprobenartig mit echten Daten gearbeitet, die sowohl aus dem SAP-System als auch von Kommandos stammen. Es hat also seine Aufgabe als Prototyp erfüllt.

Zur vollkommenen Fertigstellung, die eine Produktivnahme zulässt, fehlen nur noch die folgenden Punkte:

- Die Implementierung einiger SAP-Funktionsbausteine. Dieser Punkt ist von dm-drogerie markt zu leisten.
- Die Umlagerung der Benutzereinstellungen in das SAP-System (momentan werden `ConfigMBeans` genutzt). Dieser Punkt wurde von dm-drogerie markt gewünscht, da sie es vorziehen, die Benutzer im SAP-System zu verwalten und nicht in einer XML-Datei. Für die eigentliche Konfiguration der MLib werden weiterhin `ConfigMBeans` genutzt.
- Die Anbindung eines Skriptes.
- Weitere Detailänderungen.

Kapitel 8

Ausblick

Der wohl größte Vorteil der MLib ist ihre Flexibilität und Erweiterbarkeit. Durch die ObjectWrapper ist eine Abstraktion geschaffen, die sich praktisch auf jede Art von Objekten abbilden läßt. Durch diese Entkopplung von Daten und Darstellung lassen sich weitere Tags erstellen, die Daten dieser Objekte auf jede nur erdenkliche Art und Weise darstellen oder ändern können. Auf der anderen Seite lassen sich weitere Objekt-Typen anbinden, die automatisch zu jedem dieser Darstellungs-Tags kompatibel sind.

Theoretisch ließe sich die MLib auch zur Darstellung von Daten nutzen, die gar nichts mit JMX oder Management im Allgemeinen zu tun haben. Es wäre beispielsweise möglich, Daten aus einer relationalen Datenbank anzuzeigen. Alles was dazu notwendig wäre, ist ein weiterer ObjectWrapper-Typ für Datenbankzeilen, der Spalten einer Datenbankzeile als Attribute bereitstellt und ein weiteres Lade-Tag, mit dessen Hilfe eine Datenbankabfrage abgesetzt und die Ergebnistabelle als Liste von ObjectWrappern in den Seitenkontext stellt. Damit ließen sich schon jegliche Daten einer Datenbank auf verschiedenste Weise anzeigen. Wenn der Datenbank-ObjectWrapper es zulassen würde, Attributwerte zu ändern und wenn er Insert- oder Update-Operationen anbieten würde, könnte sogar schreibend auf eine Datenbank zugegriffen werden.

Durch die MTools lassen sich viele Datenquellen sehr einfach erschließen.

Mit den ConfigMBeans ist eine Konfiguration gegeben, die sich nicht nur vollständig in die restliche Architektur einbettet, indem sie sich in Form von MBeans selbst instrumentiert, sondern sogar zur Laufzeit aus der Webanwendung heraus veränderbar ist.

Durch das SAP-MBean-Mapping ist eine flexible Kopplung zu SAP-Systemen möglich. Die Daten und Operationen des SAP-Systems werden dabei sogar noch zu Objekten zusammengefasst und zwischengespeichert (Caching). Und alles was dazu nötig ist,

um weitere SAP-Daten anzubieten, ist das Erstellen von Funktionsbausteinen, die die gewünschten Daten und Operationen bereitstellen (sofern sie nicht bereits existieren) und die Erweiterung eine XML-Datei.

Für die Zukunft könnte man sich beispielsweise vorstellen, der MLib weitere Darstellungs-Tags zur Verfügung zu stellen, mit deren Hilfe z. B. weitere Diagrammtypen unterstützt werden könnten (z. B. ein Linien- oder Balkendiagramm).

Außerdem könnte man das Notification-Konzept von JMX für Webanwendungen zugänglich machen. So könnte ein Anwender z. B. einen Warnhinweis bekommen, wenn sich der Zustand eines Mappings auf „kritisch“ ändert.

Die naheliegendste Weiterentwicklung liegt jedoch nicht in der Erweiterung, sondern vielmehr in der Nutzung des Frameworks. Das Framework schafft eine Basis, auf der mit relativ wenig Aufwand weitere Management-Applikationen entwickelt werden können. Dieser Aufwand hat sich schnell amortisiert, wenn man bedenkt, wie viel Arbeit den Administratoren damit abgenommen werden kann.

Anhang A

Die Tags der MLib

In diesem Anhang sollen die wichtigsten Tags der MLib vorgestellt werden. Eine vollständige Auflistung aller Tags, inklusive der Hilftags, bietet die Tag-Library-Dokumentation (TLDdoc) der MLib.

Eine Übersicht über die Zusammenarbeit der Tags und Klassen der MLib finden Sie in Abbildung A.5 auf Seite 69.

A.1 Lade-Tags

Wie bereits aus Abschnitt 5.2.4 auf Seite 34 ersichtlich, muss für ein Objekt, dessen Werte man anzeigen oder anderweitig verarbeiten will, zunächst ein `ObjectWrapper` erzeugt und in den Seitenkontext geladen werden. Das wird von sog. Lade-Tags übernommen.

Dabei wird zwischen Lade-Tags unterschieden, die einen einzelnen Wrapper laden und solchen, die eine ganze Liste von Wrappern unter einem Schlüssel im Kontext ablegen.

Das Tag loadMBean

Das Tag `loadMBean` lädt eine MBean in den Kontext. Genaugenommen wird ein `MBeanWrapper` erzeugt, der als Proxy zur entfernten MBean dient. Es müssen dazu drei Parameter angegeben werden:

- `toKey`: Der Schlüssel unter dem der `MBeanWrapper` im Kontext abgelegt werden soll.
- `mbeanAgent`: Gibt den MBean-Agent an, von dem die MBean geladen werden soll.
- `mbeanQuery`: Der Name der gewünschte MBean.

```
<!-- Lädt die User-MBean mit der ID 1642 vom agent config. -->
<m:loadMBean toKey="user" mbeanAgent="config"
             mbeanQuery="JMX:type=User,id=1642"/>
```

Quelltext A.1: Beispiel zum Tag `loadMBean`

Das Tag loadMBeanList

Dieses Tag lädt eine ganze Liste von MBeans. Dazu sind folgende Parameter anzugeben:

- `toListKey`: Der Schlüssel unter dem die Liste im Kontext abgelegt werden soll.
- `mbeanAgent`: Gibt den MBean-Agent an, von dem die MBean geladen werden soll.
- `mbeanQuery`: Der Name der gewünschte MBean. Hier können die Wildcards `*` und `?` genutzt werden (siehe Abschnitt 2.6.2 auf Seite 8).
- `filterKey` (optional): Der Schlüssel unter dem der zu nutzende Query-Filter liegt (siehe Abschnitt 5.3.4 auf Seite 46).

Das Tag loadObject

Das Tag `loadObject` ist ein `ValueReaderTag`. Der gelesene Wert wird im Kontext unter dem Schlüssel abgelegt, der durch den Parameter `toKey` angegeben wurde.

```
<!-- Lädt alle User-MBeans vom agent config. -->
<m:loadMBeanList toListKey="userList" mbeanAgent="config"
                 mbeanQuery="*:type=User,*"/>
```

Quelltext A.2: Beispiel zum Tag `loadMBeanList`

```
<!--
 | Lädt das Attribut 'owner' vom Wrapper 'mapping' und speichert es unter
 | 'mappingOwner' im Kontext.
 +-->
<m:loadObject toKey="mappingOwner" fromKey="mapping" attribute="owner" />
```

Quelltext A.3: Beispiel zum Tag `loadObject`

Das Tag loadList

Dieses Tag funktioniert analog zum `loadObject`-Tag mit dem Unterschied, dass er eine Liste von Wrappern lädt, nicht einen einzelnen. Der entsprechende Attribut- oder Rückgabewert muss daher als Liste interpretierbar sein, was beispielsweise bei Arrays der Fall ist oder bei Implementierungen der Schnittstelle `java.util.Collection`, auch Java-Collections genannt.

```
<!--
 | Lädt das Attribut 'indicatorList' vom Wrapper 'mapping' und speichert es
 | unter 'list' im Kontext.
 +-->
<m:loadList toListKey="list" fromKey="mapping"
            attribute="indicatorList" />
```

Quelltext A.4: Beispiel zum Tag `loadList`

A.2 Manipulation-Tags

Wie bereits in Abschnitt 5.2.2 auf Seite 32 erläutert, kann nach dem Laden eine optionale Manipulation vorgenommen werden.

Im Rahmen der Diplomarbeit wurden zwei Arten der Manipulation implementiert: Das Sortieren und das Erweitern von Listen.

Das Tag sortList

Wie bereits gesagt, dient dieses Tag der Sortierung von Listen. Die Liste wird dabei durch den Parameter `fromListKey` angegeben, das Attribut nach dem sortiert werden soll durch den Parameter `attribute`. Durch den optionalen Parameter `order` kann man die Sortierreihenfolge festlegen: `ascending` für aufsteigendes Sortieren, was der Standardeinstellung entspricht und `descending` für absteigendes Sortieren.

```
<!-- Eine Liste von Benutzern wird nach dem Namen sortiert. -->
<m:sortList fromListKey="userList" attribute="name" />
```

Quelltext A.5: Beispiel zum Tag `sortList`

Das Tag addToList

Dieses Tag ist ein `ValueReaderTag`. Der gelesene Wert wird einer Liste angefügt.

```
<!--
 | Liest einen Benutzer vom Attribut 'owner' und fügt ihn einer Benutzerliste
 | hinzu.
 +-->
<m:addToList toListKey="userList" attribute="owner" />
```

Quelltext A.6: Beispiel zum Tag `addToList`

A.3 Selektions-Tags

Wie bereits in Abschnitt 5.2.2 auf Seite 32 beschrieben, gibt es die Möglichkeit, einen Wrapper zu selektieren, so dass sich alle Operationen innerhalb der Selektion implizit auf diesen Wrapper beziehen. Eine explizite Angabe des zu verwendenden Wrapper ist mit Hilfe des Parameters `fromKey` natürlich trotzdem jederzeit möglich (siehe Abschitt 5.2.4 auf Seite 34).

Auch bei der Selektion wird wieder zwischen der Selektion eines einzelnen Wrappers und der einer ganzen Liste von Wrappern unterschieden.

Das Tag select

Dieses Tag selektiert einen einzelnen Wrapper.

```
<!--
 | Selektiert einen Benutzer. Die ValueReaderTags innerhalb der Selektion müssen
 | daher nicht mehr angeben, von welchem Wrapper sie ein Attribut lesen wollen.
 +-->
<m:select fromKey="user">
  ID:   <m:textRenderer attribute="Id" /><br/>
  Name: <m:textRenderer attribute="Name" />
</m:select>
```

Quelltext A.7: Beispiel zum Tag `select`

Das Tag selectLoop

Dieses Tag selektiert eine Liste. D. h. es wird durch die Liste iteriert, indem nacheinander jeder Wrapper der Liste selektiert und der Body des `selectLoop`-Tag einmal für diesen Wrapper abgearbeitet wird. Innerhalb des Tag-Body ist also jeweils nur ein Wrapper selektiert, nicht die gesamte Liste.

```
<!--
 | Ausgabe einer Benutzerliste. Für jeden Benutzer in der userList wird ein
 | li-Tag mit seinem Namen erzeugt.
 +-->
<ul>
  <m:selectLoop fromListKey="userList">
    <li><m:textRenderer attribute="Name" /></li>
  </m:select>
</ul>
```

Quelltext A.8: Beispiel zum Tag `selectLoop`

A.4 Tags für die Darstellung einzelner Objekte

Im vorigen Abschnitt wurde gezeigt, wie ein Wrapper selektiert werden kann. Im Folgenden sollen nun die Renderer-Tags vorgestellt werden, die das Anzeigen einzelner Werte von Wrappern übernehmen.

Alle RendererTags sind ValueReaderTags, d. h. sie lesen den Wert eines Wrappers. Wenn nicht explizit ein anderer Wrapper angegeben wurde, so wird der Wert vom gerade selektierten Wrapper gelesen.

Das Tag textRenderer

Dieses Tag gibt den gelesenen Wert als einfachen Text aus.

```
<!--
 | Ausgabe des Attributs "Name" von Wrapper "user".
 | Beispiel-Ausgabe: "Anneliese Schmitt"
 +-->
<m:textRenderer attribute="Name" fromKey="user" />
```

Quelltext A.9: Beispiel zum Tag `textRenderer`

Das Tag numberRenderer

Mit Hilfe dieses Tags können Zahlen ausgegeben werden. Mit dem optionalen Parameter `pattern` läßt sich angeben, wie die Zahl formatiert werden soll. Auf diese Weise kann man die Anzahl der Nachkommastellen, das zu verwendende Tausendertrennzeichen, eine Einheit und vieles mehr festlegen.

```
<!--
 | Ausgabe des Attributs "FillLevel" vom gerade selektierten Wrapper.
 | Beispiel-Ausgabe: "85,76 %"
 +-->
<m:numberRenderer attribute="FillLevel" pattern="0,00 %" />
```

Quelltext A.10: Beispiel zum Tag `numberRenderer`

Das Tag csvRenderer

Um ganze Arrays von Werten als komma-separierte Liste (CSV) auszugeben, kann das `csvRenderer`-Tag genutzt werden. Für komplexere Listen, die z. B. aus Wrappern bestehen, kann der `csvBuilder`-Tag genutzt werden (siehe Abschnitt A.5 auf Seite 64).

```
<!--
 | Ausgabe der Adressaten eines E-Mail-Verteilers.
 | Beispiel-Ausgabe: "karl@stz-ida.de, info@murfman.de, otto@stz-ida.de"
 +-->
<m:csvRenderer attribute="ReceiverList" />
```

Quelltext A.11: Beispiel zum Tag `csvRenderer`

Das Tag imageRenderer

Dieses Tag kann je nach Ausprägung eines Wertes ein bestimmtes Bild anzeigen. So kann z. B. ein grüner Haken gezeigt werden, wenn der Wert `true` ist, und ein Hand-Symbol, wenn er `false` ist.

Momentan kann das Tag nur auf Gleichheit prüfen. Es wären noch weitere Ausbaustufen denkbar, die z. B. ein Bild anzeigen, wenn sich ein Zahlenwert innerhalb eines Intervalls bewegt. Es könnten auch reguläre Ausdrücke hinterlegt werden.

```
<!--
 | Zeigt einen grünen Haken, wenn das Attribut "isRunning" den Wert "true"
 | hat, und eine schwarze Hand, bei "false".
 +-->
<m:imageRenderer attribute="isRunning">
  <m:image onValue="true" image="img/green.gif" alt="Läuft"/>
  <m:image onValue="false" image="img/black.gif" alt="Läuft nicht"/>
</m:imageRenderer>
```

Quelltext A.12: Beispiel zum Tag `imageRenderer`

Das Tag linkRenderer

Wie bereits aus dem Tag-Namen unschwer zu erkennen ist, erzeugt das Tag `linkRenderer` einen Link. Die Aktion, die beim Drücken des Links ausgelöst werden soll, wird durch den eingebetteten Action-Tag bestimmt.

Momentan gibt es nur eine Sorte Action-Tags: Das `openPageAction`-Tag. Dieses Tag kann beliebig viele `pageParam`-Tags einbetten. Jedes dieser `pageParam`-Tags fügt der zu öffnenden URL einen MBean-Wert als Parameter hinzu.

In einer späteren Version könnte man noch weitere Action-Tags einführen, um beispielsweise eine Operation bei einer MBean aufzurufen wenn der Benutzer auf den Link klickt.

Umgekehrt wäre es möglich, weitere Tags zu definieren, in die man Action-Tags einbetten kann. Diese Tags würden die Aktion dann bei einem anderen Ereigis auslösen, z. B. beim Drücken eines Knopfes oder bei der Wahl eines Listenelements.

```
<!--
 | Erzeugt einen Link mit dem Namen eines Benutzers als Linktext.
 | Beim Drücken des Links wird die Seite "UserDetail.jsp" aufgerufen, wobei die
 | ID des Benutzers unter dem Parameter "userId" übergeben wird.
 +-->
<m:linkRenderer attribute="Name">
  <m:openPageAction url="UserDetail.jsp">
    <m:pageParam name="userId" attribute="Id" />
  </m:openPageAction>
</m:linkRenderer>
```

Quelltext A.13: Beispiel zum Tag `linkRenderer`

A.5 Tags für die Darstellung ganzer Listen

Im Abschnitt A.3 auf Seite 60 wurde gezeigt, wie mit Hilfe des `selectLoop`-Tag durch eine Liste iteriert werden kann. So konnte für jeden Wrapper in einer Liste eine Ausgabe erzeugt werden.

Es gibt jedoch Widgets, die nicht nur einen Wert, sondern eine gesamte Liste von Objekten darstellen können. Eine Tabelle stellt beispielsweise eine Liste von Objekten in Form von vielen Zeilen dar. Ein Tortendiagramm als eine Menge von Tortenstücken.

Eine solche Darstellung von Listen wird von sog. Builder-Tags übernommen. Ein Builder-Tag ist ein spezialisierter Listen-Selektionstag. Genau wie beim `selectLoop`-Tag werden die Wrapper in der Liste nacheinander selektiert und der Body einmal abgearbeitet. Bei jeder Abarbeitung des Body wird dem Widget ein Element hinzugefügt. Bei einer Tabelle eine Zeile, bei einem Tortendiagramm ein Tortenstück.

Das Tag csvBuilder

Mit Hilfe des `csvBuilder`-Tags können CSVs, also komma-separierte Listen erstellt werden. Für die einfache Ausgabe eines Arrays kann auch das Tag `csvRenderer` genutzt werden (siehe Abschnitt A.4 auf Seite 61).

```
<!--
 | Ausgabe einer Benutzerliste als CSV.
 | Beispiel-Ausgabe: "Hans Maier (182), Karl Schmitt (432), Anna Bauer(185)"
 +-->
<m:csvBuilder fromListKey="userList">
  <m:textRenderer attribute="Name" /> (<m:textRenderer attribute="Id" />)
</m:select>
```

Quelltext A.14: Beispiel zum Tag `csvBuilder`

Das Tag tableBuilder

Dieses Tag kann ganze Tabellen ausgeben. Es enthält eine oder mehrere `column`-Tags, die bei jedem Durchlauf des Body die Zelle der entsprechenden Spalte generieren.

Über den optionalen Parameter `cssClass` kann die Stylesheet-Klasse angegeben werden, der die Tabelle zugeordnet werden soll. Über den ebenfalls optionalen Parameter `hideWhenEmpty` kann man erreichen, dass die Tabelle nicht ausgegeben wird, wenn die Liste leer ist, andernfalls wird nur die Titelzeile ausgegeben.

```
<!-- Ausgabe einer Prozessliste als Tabelle. -->
<m:tableBuilder fromListKey="processList" cssClass="overview">
  <m:column title="Prozessname" attribute="ProcessName" />
  <m:column title="CPU-Last">
    <div align="right">
            <m:numberRenderer attribute="CpuLoad" pattern="0.00 %" />
    </div>
  </m:column>
</m:tableBuilder>
```

Quelltext A.15: Beispiel zum Tag `tableBuilder` (Ausgabe siehe Abbildung A.1 auf der nächsten Seite)

Prozessname	CPU-Last
wait	18,55 %
/usr/bin/tprof	17,12 %
/usr/bin/trcstop	12,99 %
/home/mercator/bin/launcher	9,91 %
/usr/bin/sh	2,72 %
java	0,34 %
IBM.CSMAgentRMd	0,34 %
java	0,10 %
/usr/bin/tprof	0,10 %
java	0,06 %

Abbildung A.1: Ausgabe des `tableBuilder`-Tags

Das Tag pieChartBuilder

Mit Hilfe des `pieChartBuilder`-Tags lassen sich Tortendiagramme erstellen. Im Body wird durch `map`-Tags für jeden Wrapper der Liste ein Tortenstück erzeugt. Ein `map`-Tag gibt die Beschriftung des Stücks an, das andere den Wert.

Manchmal möchte man, dass die Gesamtsumme aller Tortenstücke einen bestimmten Wert, z. B. 100%, bildet. Falls die tatsächliche Summe kleiner als dieser Wert ist, so soll das Diagramm durch ein zusätzliches Tortenstück „aufgefüllt“ werden. Dies kann man durch die Parameter `fillSliceLabel` und `fillSliceValue` erreichen, die die Beschriftung des Füllstücks bzw. die zu erreichende Gesamtsumme angeben.

Mit Hilfe der Parameter `width` und `height` kann die Größe des Diagramms in Pixeln angegeben werden.

Außerdem kann man mit Hilfe des Parameters `valuePattern` ein Zahlenformat angeben, indem die Werte ausgegeben werden sollen.

Wenn man den Parameter `style3D` auf `true` setzt, dann bekommt das Diagramm einen 3-D-Effekt (siehe Abbildung A.3 auf Seite 67).

Bei der Abarbeitung wird als zusätzliche Hilfe für den Benutzer neben dem Diagramm-Bild eine Polygon-Map erzeugt, die ein Polygon um jedes Tortenstück beschreibt. Mit Hilfe dieser Map werden im Browser Tooltip-Texte gezeigt, sobald der Nutzer den Mauszeiger über ein Tortenstück hält (siehe Abbildung A.2 auf der nächsten Seite). Mit Hilfe dieser Map könnte man auch weitergehende Interaktionsmöglichkeiten bieten. Man könnte beispielsweise einen Link hinterlegen, so dass bei einem Klick eine Detailansicht o. ä. gezeigt wird.

Die Diagramm-Bilder werden mit Hilfe des Open-Source-Projekts JFreeChart [Chart] erzeugt. In ähnlicher Weise ließen sich auch andere Diagrammtypen realisieren, wie z. B. ein Balken-, Linien-, Punkt- oder Flächendiagramm.

```
<!-- Ausgabe einer Prozessliste als Tortendiagramm. -->
<m:pieChartBuilder fromListKey="processList" title="CPU-Last"
                   width="500" height="350" valuePattern="0.00%"
                   fillSliceLabel="Sonstige" fillSliceValue="1">
  <m:map to="label" attribute="ProcessName" />
  <m:map to="value" attribute="CpuLoad" />
</m:pieChartBuilder>
```

Quelltext A.16: Beispiel zum Tag `pieChartBuilder` (Ausgabe siehe Abbildung A.2)

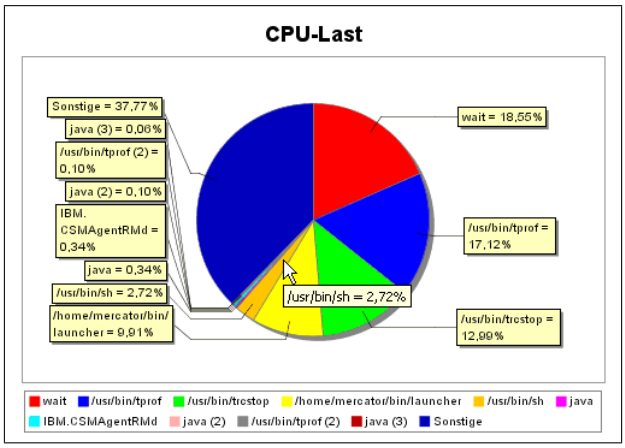

Abbildung A.2: Ausgabe des `pieChartBuilder`-Tags

A.6 Tags für Formulare

Spätestens wenn es darum geht, dem Benutzer eine Möglichkeit zu geben, Werte von ObjectWrappern zu ändern, müssen in einer Web-Applikation Formulare eingesetzt werden.

In den meisten Fällen reichen dazu einfache Felder, wie z. B. Texteingabefelder aus. Diese Tags sind so einfach, dass dazu keine Unterstützung seitens der MLib nötig wäre.

Für kompliziertere Widgets kann die MLib jedoch Tags bereitstellen, die deren Erstellung vereinfachen.

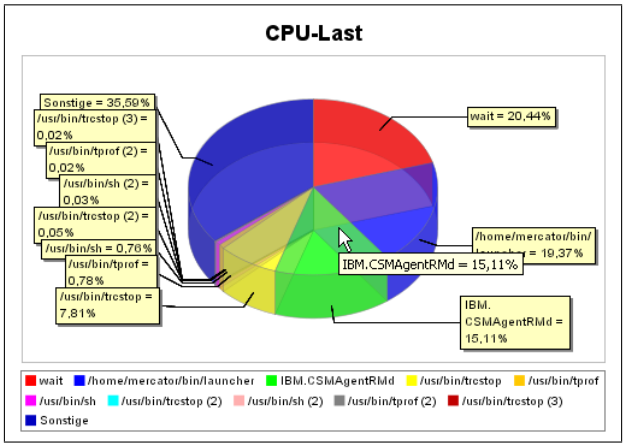

Abbildung A.3: Ausgabe des `pieChartBuilder`-Tags mit `style3D="true"`

Das Tag doubleListInput

Die Doppelliste ist ein komplexeres Widgets für die Auswahl von Untermengen. Eine solche Auswahl kann zwar auch mit nur einer Liste erreicht werden, indem der Benutzer die Taste `Strg` gedrückt hält und jene Elemente markiert, die er auswählen möchte, jedoch kommen viele Benutzer damit nicht zurrecht. Sei es weil sie den „Trick“ mit der `Strg`-Taste nicht kennen oder weil sie sie aus Versehen einmal loslassen und dadurch die gesamte Selektion zunichte machen.

Wesentlich komfortabler ist die Auswahl über zwei Listen: Eine Liste (die linke) enthält alle Elemente, die noch wählbar sind, eine zweite Liste (die rechte) enthält alle Elemente, die bereits ausgewählt sind (siehe Abbildung A.4 auf der nächsten Seite). Mit zwei Pfeil-Knöpfen können Elemente zwischen den Listen bewegt werden.

Da für dieses Widget Javascript notwendig ist und da die Beschreibung der Listen meist recht umfangreich ausfällt, bietet die MLib hierfür das Tag `doubleListInput` an, das die Benutzung stark vereinfacht.

In den beiden Untertags `availableList` bzw. `selectedList` wird jeweils die Gesamtmenge bzw. die Menge der vorausgewählten Elemente angegeben. Falls das Seitenattribut, das dem `doubleListInput`-Tag zugeordnet ist, gesetzt ist, wird die Menge der vorausgewählten Elemente daraus bezogen. Dies ist für die Formularprüfung nötig, bei der es oft dazu kommt, dass das selbe Formular nach dem Abschicken noch einmal mit den zuletzt eingestellten Werten gezeigt wird, wenn der Server Eingabefehler festgestellt hat.

Bei der Angabe der Elemente wird dabei zwischen einem angezeigten Wert und einem internen Wert unterschieden. So kann beispielsweise ein für Menschen besser lesbarer

Name oder eine lokalisierte Kurzbeschreibung angezeigt werden, während intern mit IDs gearbeitet wird.

Das Widget hat jedoch den Nachteil, dass der Browser des Benutzers Javascript zulassen muss, was jedoch bei allen modernen Browsern gegeben ist - es sei denn, es wurde vom Benutzer aus Sicherheitsgründen deaktiviert.

```
<!--
 | Erzeugt eine Doppelliste.
 | * Die ausgewählten Elemente werden der vom Formular aufgerufenen Seite mit
 |   dem Seitenparameter "ownerList" übergeben.
 | * Die Menge der wählbaren Elemente wird durch die ObjectWrapper-Liste
 |   "userList" bestimmt, wobei das Attribut "Id" zur Identifizierung des
 |   Elements und das Attribut "Name" für die Anzeige genutzt wird.
 | * Die Menge der vorausgewählten Elemente wird durch die ObjectWrapper-Liste
 |   "selectedUserList" bestimmt.
 +-->
<m:doubleListInput name="ownerList">
  <m:availableList fromListKey="userList">
    <m:map to="value" attribute="Id"/>
    <m:map to="label" attribute="Name"/>
  </m:availableList>
  <m:selectedList fromListKey="selectedUserList">
    <m:map to="value" attribute="Id"/>
  </m:selectedList>
</m:doubleListInput>
```

Quelltext A.17: Beispiel zum Tag `doubleListInput` (Ausgabe siehe Abbildung A.4)

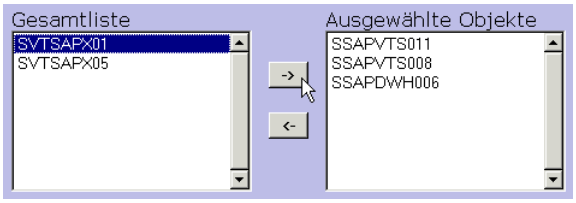

Abbildung A.4: Ausgabe des `doubleList`-Tags

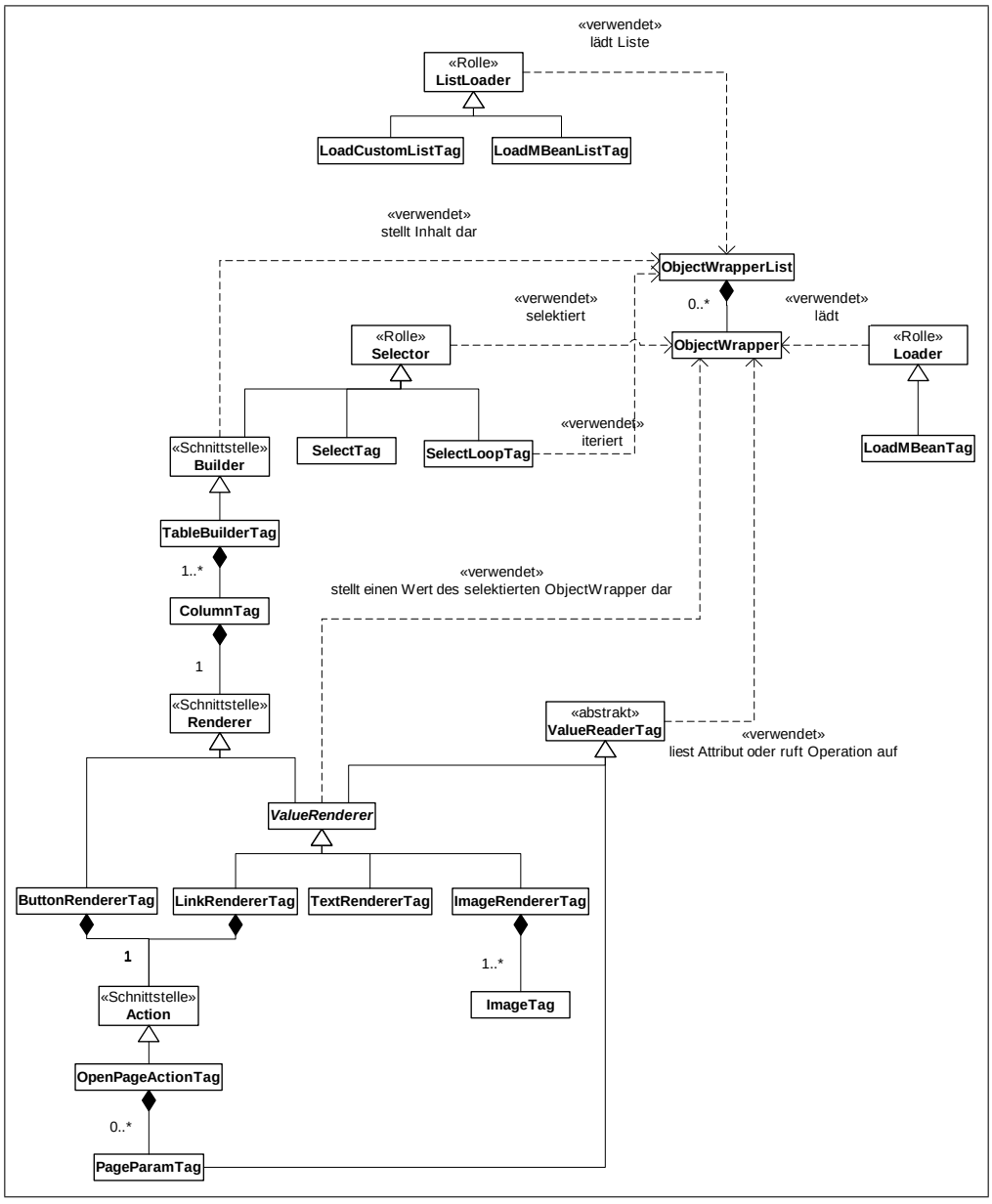

Abbildung A.5: Zusammenarbeit der Tags und Klassen der MLib

Anhang B

XML-Beschreibung des SAP-MBean-Mapping

Wie bereits erwähnt, müssen in der XML-Beschreibung sowohl die Verbindungseinstellungen zu den einzelnen SAP-Systemen als auch die eigentlichen Abbildungen für jeden Ressourcentyp beschrieben werden.

B.1 Das sapConnection-Tag

Die Tags zur Definition einer Verbindung zu einem SAP-System sind relativ trivial. Sie enthalten einen Namen für die Verbindung, die Logindaten und die Daten, die zur Identifizierung des SAP-Systems notwendig sind, wie z. B. Hostname, Systemnummer und Mandant (siehe Quellcode B.1 auf der nächsten Seite).

B.2 Das mapping-Tag

Ein `mapping`-Tag zur Definition eines Ressourcentyps hat ein Attribut für den Typnamen und ein optionales Attribut, in dem die Verbindungen zu den SAP-Systemen angegeben werden können, in denen Ressourcen diesen Typs zu finden sind. In all diesen

```
<sapConnection name="de100" poolSize="5">
  <systemId>DE1</systemId>
  <client>109</client>
  <host>sapde11</host>
  <systemNumber>01</systemNumber>
  <loginUser>karl</loginUser>
  <loginPassword>sehrgeheim</loginPassword>
  <loginLanguage>de</loginLanguage>
</sapConnection>
```

Quelltext B.1: XML-Beschreibung einer Verbindung zu einem SAP-System

Systemen müssen die im Mapping genutzten Funktionsbausteine vorhanden sein. Fehlt das Attribut, so werden alle definierten Verbindungen genutzt.

Innerhalb des `mapping`-Tags können ein Initialisierer und beliebig viele Attribut-Getter, -Setter oder Operationen definiert werden. Wenn der Initialisierer fehlt, wird eine MBean pro SAP-System angelegt.

```
<mapping type="MercatorMapping" useConnections="de100, dk100">
  <initializer ... />

  <attributeGetter ... />
  <attributeGetter ... />
  ...

  <attributeSetter ... />
  <attributeSetter ... />
  ...

  <operation ... />
  <operation ... />
  ...
</mapping>
```

Quelltext B.2: XML-Beschreibung eines SAP-MBean-Mappings

B.3 Funktions-Mapping-Tags

Parameter sind bei SAP-Funktionsbausteinen in drei Kategorien aufgeteilt: Import, Export und Table. Die Import-Parameterliste enthält Skalare und flache Strukturen, die der Funktion übergeben werden. Die Export-Parameterliste besteht ebenfalls aus Skalaren

und flachen Strukturen, jedoch werden diese von der Funktion zurückgegeben. Die Tabellenliste enthält tabellarische Daten. Diese können der Funktion sowohl übergeben werden, als auch von der Funktion zurückgegeben werden.

Die XML-Beschreibung eines Funktions-Mapping nimmt diese Aufteilung auf, unterscheidet jedoch noch zusätzlich zwischen Import- und Export-Tabellen.

Die Aufgabenbeschreibung der Funktions-Mappings werden in Abschnitt 5.3.2 auf Seite 42 erläutert.

Das Tag initializer

Neben dem Parameter `function`, das den Namen des zu verwendenden SAP-Funktionsbausteins angibt, kennt das `initializer`-Tag einen `maxAge`-Parameter. Er gibt an, in welchem Intervall der Initialisierer aufgerufen werden soll. Ein solcher Aufruf hat zur Folge, dass, falls sich die Menge der Instanzen im SAP-System geändert hat, auch die entsprechenden MBeans gelöscht bzw. erzeugt werden.

Eingabeparameter kennt das `initializer`-Tag keine. Als Ausgabeparameter muss ein einziges `exportTable`-Tag angegeben werden, in dem über `attributeMapping`-Tags alle Attribute angegeben werden, die vom Initialisierer gesetzt werden sollen. Schlüsselattribute werden dabei mit `isKey="true"` markiert.

Die `attributeMapping`-Tags erlauben jeweils die Angabe eines `sapName`- und eines `jmxName`-Parameters. Dadurch wird angegeben wie das entsprechende Feld des SAP-Funktionsbausteins heißt und auf welchen Attributnamen es in der SAP-MBean abgebildet werden soll. Wenn der `jmxName`-Parameter fehlt, wird für die MBean die selbe Bezeichnung wie im SAP-System verwendet.

```
<initializer function="Y_GET_KEYS" maxAge="7200"> <!-- 7200sec = 2h -->
  <exportTable name="PET_EXPORT">
    <attributeMapping isKey="true" sapName="MAPPING_ID" jmxName="Id"/>
    <attributeMapping sapName="MAPPING_NAME" jmxName="Name"/>
  </exportTable>
</initializer>
```

Quelltext B.3: XML-Beschreibung eines Initialisierers

Das Tag attributeGetter

Auch das Tag `attributeGetter` kennt einen `maxAge`-Parameter. Er gibt an, wie lange bei Anfragen der Wert aus dem Attribut-Cache zurückgegeben werden soll, bevor der Getter erneut aufgerufen wird. Da ein Getter meistens Werte für mehrere Attribute liefert, sollte hier selbst bei Werten, die sich ständig ändern, wenigstens ein kleiner Wert von z. B. 10 Sekunden angegeben werden. So wird verhindert, dass der Getter bei einer zeitnahen Anfrage vieler (verschiedener) Attributwerte mehrmals aufgerufen wird.

Als Eingabeparameter können beliebig viele `import`-Tags angegeben werden. Damit können dem SAP-Funktionsbaustein beliebig viele Attributwerte übergeben werden. Um zyklische Aufrufe von Attribut-Gettern zu vermeiden, werden hierzu jedoch nur Werte aus dem Cache genutzt.

Als Ausgabeparameter sind beliebig viele `export`-Tags möglich. Jedes angegebene Attribut wird dann im Cache neu gesetzt.

```
<attributeGetter function="Y_GET_META" maxAge="300"> <!-- 300sec = 5min -->
  <import name="PI_IMPORT">
    <attributeMapping sapName="MAPPING_ID" jmxName="Id"/>
  </import>
  <export name="PE_EXPORT">
    <attributeMapping sapName="OWNER"/>
    <attributeMapping sapName="MAPPING_TYPE"/>
  </export>
</attributeGetter>
```

Quelltext B.4: XML-Beschreibung eines Attribut-Getters

Das Tag attributeSetter

Dem `attributeSetter`-Tag können als Eingabeparameter wiederum beliebig viele `import`-Tags mitgegeben werden. Diese können beliebig viele `attributeMapping`-Tags beinhalten, jedoch darf insgesamt nur ein `paramMapping`-Tag angegeben werden. Dieses Mapping gibt das Attribut an, dessen Wert mit diesem Setter gesetzt wird.

Ausgabeparameter kennt das `attributeSetter`-Tag keine.

```
<attributeSetter function="Y_SET_NAME">
  <import name="PI_IMPORT">
    <attributeMapping sapName="MAPPING_ID" jmxName="Id"/>
    <paramMapping sapName="NEW_NAME" jmxName="Name"/>
  </import>
</attributeSetter>
```

Quelltext B.5: XML-Beschreibung eines Attribut-Setters

Das Tag operation

In einem `operation`-Tag können als Eingabeparameter beliebig viele `import`-Tags angegeben werden. Dabei können einerseits `attributeMapping`-Tags angegeben werden, um dem SAP-Funktionsbaustein Attributwerte zu übergeben, andererseits können `paramMapping`-Tags genutzt werden, um die MBean-Operation mit Parametern zu bestücken. Diese müssen beim Aufruf der MBean-Operation von der Management-Applikation übergeben werden.

Für die Ausgabe sind drei Szenarien denkbar:

- Durch die Angabe eines einzelnen, leeren `exportTable`-Tags, wird eine MBean-Operation erzeugt, die eine Tabelle als `TabularData`-Objekt zurückgibt (siehe Abbildung B.6 auf der nächsten Seite, Zeile 6).
- Wenn ein einzelnes, leeres `export`-Tag angegeben wird, dann wird die resultierende MBean-Operation eine Struktur zurückgeben (als `CompositeData`-Objekt).
- Wurde ein `export`-Tag mit einem einzelnen `returnMapping`-Tag angegeben, so gibt die MBean-Operation einen Skalar zurück, der den Wert des angegebenen Feldes hat (siehe Abbildung B.6 auf der nächsten Seite, Zeilen 14 bis 16).

```
<operation function="Y_GET_IND_LIST" jmxName="getIndicators">
  <import name="PI_IMPORT">
    <attributeMapping sapName="MAPPING_ID" jmxName="Id"/>
    <paramMapping sapName="BET_ZEIT"/>
  </import>
  <exportTable name="PET_EXPORT" />
</operation>

<operation function="Y_GET_SOME_INT" jmxName="getSomeInt">
  <import name="PI_IMPORT">
    <attributeMapping sapName="MAPPING_ID" jmxName="Id"/>
    <paramMapping sapName="BET_ZEIT"/>
  </import>
  <export name="PE_EXPORT">
    <returnMapping sapName="THE_INT"/>
  </export>
</operation>
```

Quelltext B.6: XML-Beschreibung von Operationen

Literaturverzeichnis

[JMX] Java Management Extensions Instrumentation and Agent Specification, v1.2, Sun Microsystems, Inc, Oktober 2002,
http://java.sun.com/products/JavaManagement/reference/docs

[DevNet] SAP Developer Network,
https://www.sdn.sap.com

[Serv] Java Servlet 2.4 Specification, Sun Microsystems, Inc, November 2003,
http://jcp.org/aboutJava/communityprocess/final/jsr154

[JSP] JavaServer Pages 2.0 Specification, Sun Microsystems, Inc, November 2003,
http://jcp.org/aboutJava/communityprocess/final/jsr152

[Struts] Homepage von Struts, einem Framework für die einfache Erstellung von JSP-basierten Web-Anwendungen,
http://jakarta.apache.org/struts

[MX4J] Projektseite von MX4J, einer alternativen JMX-Implementierung,
http://mx4j.sourceforge.net

[AppMan] Das Management von Applikationen und Web-Services, H. Trautmann, S. Buchberger, Januar 2004,
http://www.sigs.de/publications/os/2004/01/buchberger_OS_01_04.pdf

[WebDyn] Mehr Freiheit für Entwickler, SAP AG, Abruf 15.07.04,
http://www.sap.info/public/en/index.php4/article/Article-275503d9ad5672b242/de/articleStatistic

[Chart] Homepage von JFreeChart, einem Projekt zur Erstellung von Graphen,
http://www.jfree.org/jfreechart

[JUnit] Homepage von JUnit, einem Projekt für Unit-Tests,
http://junit.org

[Ant] Homepage von Ant, einer Skriptsprache für die Automatisierung von Erzeugungsvorgängen,
http://ant.apache.org

[Log4J] Homepage von Log4J, einem Projekt für Logging,
http://logging.apache.org/log4j

Glossar

API ist die Abkürzung für Application Programming Interface. Eine API ist die Schnittstelle, die eine Applikation nutzen soll, um auf eine Bibliothek zuzugreifen.

Bootstrap Als Bootstrapping wird der Startvorgang eines Systems bezeichnet. Bootstrap ist das englische Wort für Schnürsenkel. Bildlich gesehen werden also zuerst die Stiefel geschnürt, bevor das System laufen kann. Bootstrapping wird häufig auch mit „booten“ abgekürzt.

Get-Methode ist eine Methode, bei einer HTTP-Anfrage Parameter mitzugeben. Im Gegensatz zur Post-Methode werden die Parameter bei Get an die URL angefügt, so dass sie beim Setzen eines Lesezeichens (engl.: Bookmark) erhalten bleiben.

Getter nennt man eine Methode, die den Wert eines Objektattributs nach außen zugänglich macht. In der Java-Bean-Konvention beginnen solche Methoden mit „get“. Beispiel: `public int getSize()` (siehe auch Setter).

HTTP bedeutet Hypertext Transfer Protocol und ist ein vom W3C spezifiziertes Protokoll zur Übertragung von HTML-Seiten. Mittlerweile wird es jedoch auch zur Übertragung von anderen Inhalten eingesetzt.

JCP Der Java Community Process wurde 1998 von Sun gegründet. Über den JCP gibt Sun anderen Firmen und Organisationen die Möglichkeit, and der Weiterentwicklung von Java mitzuwirken.

JDBC steht für Java Database Connectivity und ist eine von Sun entwickelte API zum Zugriff auf relationale Datenbanken.

JSP steht für Java Server Page. Ein JSP ist eine dynamische Web-Seite, in die Java-Code integriert ist (siehe Absatz 2.3 auf Seite 4).

JMX bedeutet Java Management Extension. JMX bietet Management für Java (siehe Absatz 2.6 auf Seite 6).

Post-Methode ist eine Methode, bei einer HTTP-Anfrage Parameter mitzugeben. Im Gegensatz zur Get-Methode werden die Parameter bei Post im HTTP-Header übertragen, so dass sie für den Benutzer verborgen bleiben.

Proxy Siehe Wrapper.

Regulärer Ausdruck ist eine Beschreibung für ein Textmuster. Die Beschreibung eines Textmusters erfolgt durch die Benutzung von sehr vielseitigen Platzhaltern.

Seitenkontext Der Seitenkontext (engl.: „page context“) ist eine vom Servlet-Container bereitgestellte Datenstruktur, die Schlüssel-Wert-Paare aufnehmen kann. Im Seitenkontext können Daten abgelegt werden, die temporär zum Aufbau einer Antwortseite benötigt werden. Der Seitenkontext ist von allen anderen Tags der selben Seite aus erreichbar.

Neben dem Seitenkontext stellt der Servlet-Container auch noch einen Session- und einen Applikationskontext zur Verfügung. All die Kontexte unterscheiden sich in der Sichtbarkeit und der Lebensdauer. Während der Seitenkontext nur innerhalb einer Seite sichtbar ist und nach dem Seitenaufbau gelöscht wird, ist der Sessionkontext innerhalb einer Session und der Applikationskontext innerhalb einer ganzen Applikation sichtbar.

Servlet ist eine Java-Klasse zur Bearbeitung von Anfragen vom Client an den Server (siehe Absatz 2.2 auf Seite 3).

Setter nennt man eine Methode, die den Wert eines Objektattributs von außen veränderbar macht. In der Java-Bean-Konvention beginnen solche Methoden mit „set“. Beispiel: `public void setSize(int newSize)` (siehe auch Getter).

Tag-Library ist eine Sammlung spezieller Tags für JSP-Seiten (siehe Absatz 2.4 auf Seite 4).

URL bedeutet Uniform Ressource Locator und beschreibt den physikalischen Ort einer Ressource eindeutig. Ein Beispiel für eine URL wäre: `http://www.murfman.de`.

W3C steht für World Wide Web Consortium und ist eine Vereinigung zur Entwicklung freier und interoperabler Internettechnologien. Vom W3C stammt das HTTP-Protokoll und viele wichtige Internetformate, wie z. B. HTML, CSS, XML, XSL, PNG und SVG um nur die wichtigsten zu nennen (siehe `http://www.w3.org`).

Wrapper ist der Name für ein Entwurfsmuster. Ein Wrapper kontrolliert den Zugriff auf ein Objekt mit Hilfe eines vorgelagerten Stellvertreterobjektes. Auch als „Proxy“ bekannt.

Index