INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Konfigurierbare Sensoren für das Monitoring in Grid-Umgebungen

Cornelius Dirmeier und Luka Leovac

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Konfigurierbare Sensoren für das Monitoring in Grid-Umgebungen

Cornelius Dirmeier und Luka Leovac

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering
Betreuer: Timo Baur

Abgabetermin: 1. April 2008

Hiermit versichern wir, dass die vorliegende Ausarbeitung des Fortgeschrittenenpraktikums selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet wurden.

München, den 1. April 2008

. . . . . .

(Unterschriften der Kandidaten)

Zusammenfassung Am Leibniz Rechenzentrum wird als Teil des D-Grid-Projekts das Monitoring von Ressourcen in der Grid-Umgebung untersucht. Dabei geht es unter anderem darum bestimmte Informationen aus den Sensoren der D-Grid Ressourcen für den Benutzer sichtbar zu machen. Dies geschieht über das einheitliche WebMDS-Portal. Das Problem ist bisher, dass die Sensoren erst aufwendig konfiguriert werden müssen, bevor die Änderungen im WebMDS-Portal sichtbar sind.

Um die aufwendige Konfiguration für den normalen Benutzer zu erleichtern wurde im Rahmen dieses Projekts eine Benutzerschnittstelle für die Sensoren im Globus Toolkit erstellt. Mit Hilfe eines einfachen Frontends kann man nun die Daten der verfügbaren Sensoren konfigurieren.

Um unsere Funktionalitäten anderen Applikationen des Globus Toolkits zur Verfügung zu stellen wurde auf den Standards des Globus Toolkit ein WebService implementiert, der die Konfiguration der Sensoren zulässt.

Eine wesentliche Neuentwicklung war das An- und Abschalten von Sensoren. Während bisher der Sensor manuell aus der Konfiguration gelöscht werden musste, kann man mit Hilfe dieses Sensor Konfigurators Sensoren einfach aktivieren, deaktivieren bzw. neu erstellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Einleitung

Anforderungen an Rechenleistung und Rechenkapazität wachsen stetig, da immer komplexere Aufgaben zu lösen sind. Zum Beispiel benötigt die Berechnung von Simulationen eines Erdbebens viele Ressourcen, die nicht überall zur Verfügung stehen. Um freiligende Rechenkapazität an anderen Orten optimal auszunutzen, liegt es nahe diese in irgend einer Form zu verteilen. Dies kann durch eine Grid-Infrastruktur erfolgen. Bei einem Grid handelt es sich um eine Infrastruktur, die eine integrierte, gemeinschaftliche Verwendung von meist geographisch auseinander liegenden, autonomen Ressourcen erlaubt.

Der Begriff hat seinen Ursprung in dem Vergleich dieser Technologie zum Stromnetz (engl. power grid). Zum Beispiel kann der Strom, wenn eine Komponente (Kraftwerk) ausfällt, von einem anderen Kraftwerk genommen werden. Analog kann man auch in einem Computer Grid Resourcen ersetzen. Diese Ressourcen werden jedoch nicht von einer zentralen Instanz kontrolliert. Um eine dezentralisierte Struktur zu verwirklichen werden offene, standardisierte Protokolle und Schnittstellen verwendet, um diese nicht trivialen Dienstgüter bereitzustellen [FKTT 98]. Ein Grid ermöglicht zudem die Koordination von Benutzerzugriffen. Dabei werden Sicherheitskonzepte und Ressourcen-Verwaltung vom gesamten System gesteuert. Als Ressourcen sind in diesen Zusammenhang z.B. Datenbanken, Supercomputer, Massenspeicher und ähnliches zu verstehen.

Eine solche Infrastruktur wird in Deutschland im Rahmen des D-Grid-Verbundes für den Bereich des sogenannten e-Science (enhanced science) aufgebaut. In diesem wissenschaftlichen Projekt werden mit staatlicher Hilfe die Bereiche Wissensmanagement (Wissensvernetzung), Open Access, e-Learning und Grid-Computing gefördert. Zur Verwirklichung von e-Science, sind international verschiedenste Forschungs- und Entwicklungstätigkeiten geplant oder bereits angelaufen, die eine neue Infrastruktur für wissenschaftliche Kommunikation und Kollaboration, für Informationsbereitstellung, Datenaustausch und -nutzung sowie die Publikation wissenschaftlicher Ergebnisse zum Ziel haben [esc 07]. Die D-Grid-Infrastruktur soll dabei helfen, Methoden des e-Science zu etablieren, indem sie Projekte aus verschiedenen wissenschaftlichen Gebieten (u.a. Astronomie, Hochenergiephysik, Meteorologie, Medizin, Ingenieurwissenschaften und Geisteswissenschaften) eine Grid-Infrastruktur mit den benötigten Diensten und Benutzer-Support zur Verfügung stellt. So können sich in Zukunft die beteiligten Projekte ganz auf ihre fachliche Problematik konzentrieren und auf eine funktionierende und stabile Infrastruktur zugreifen.

1.2 Motivation für diese Arbeit

Im Rahmen des D-Grid Projekts untersucht das Leibniz Rechenzentrum u.a. Fragen des Dienst- und Ressourcen-Monitorings in Grid Umgebungen. Das LRZ stellt die Ressourcen (z.B. Rechenleistung der Supercomputer) anderen Institutionen im D-Grid Projekt zur Verfügung. Damit andere Institutionen oder Benutzer sich über die statischen und dynamischen Zustände der Ressourcen oder die Qualität der angebotenen Dienste informieren können, werden sogenannte Sensoren (Informationsprovider) in der Grid Umgebung bereitgestellt. Mit Hilfe von Sensoren können die Administratoren den Status (den aktuellen Betriebszustand) eines Rechners den Benutzern zugänglich machen. Die Informationen aus dem Monitoring werden dann veröffentlicht und verschiedenen Nutzergruppen und Grid-Diensten zur Verfügung gestellt.

Die Konfiguration der in Grid-Umgebungen verfügbaren Sensoren erfolgt derzeit per Hand, d.h. die Administration erfolgt lokal an der Konsole. Dazu ist das Einloggen an den jeweiligen Rechner und das Verändern langer XML-Dateien notwendig. Dies ist sehr mühsam und zeitaufwändig, und kann nur von Administratoren durchgeführt werden. Um den Benutzern das Monitoring zu erleichtern, wurde im Rahmen dieses Fortgeschrittenenpraktikums eine Weboberfläche, sowie eine Schnittstelle für das Monitoring und die Konfiguration

der Sensoren erstellt. Dadurch können allgemeine Sensoreinstellungen einfach per Weboberfläche verändert werden. Sollen neue Sensoren installiert werden, muss der Administrator lediglich ein paar wenige Dateien anpassen.

1.3 Vorgehensweise

Der Aufbau dieses Dokuments wurde strukturell nach dem Ablauf des Projekts gegliederd. So beschreibt es im ersten Teil, Kapitel 2, die Analyse des bestehenden Systems. Kapitel 3 erläutert die Ergebnisse der darauf bauenden Anforderungsanalyse. An dieser Stelle werden zunächst die zu bearbeitenden UseCases definiert und darauf hin die technischen Anforderungen beschrieben. Aus diesen ging auch hervor, dass eine Teilung in Frontend und Backend sinnvoll ist.

Im Kapitlel 4 wird die Architektur und das Design des gesammten Systems erläutert. Dieser Teil wird zur übersichtlicheren Gliederung in getrennten Teilen für Frontend und Backend verfasst. Analog dazu wird die Implementierung in Kapitel 5 ebenfalls aufgeteilt. In diesem Teil wird auch auf die Installation und Handhabung der Teilsysteme hingewiesen.

Zum Abschluss dieser Arbeit wird in Kapitel 6 ein Ausblick auf die weitere Verwendung des Systems gegeben.

2 Ausgangssituation

Bevor das System genauer beschrieben wird, soll der Aufbau des bisherigen Systems erläutert werden.

Am LRZ wird der Kern des Grid-Systems durch das Globus Toolkit 4 (GT4) realisiert. Wie in der Grafik 2.1 schematisch dargestellt, werden die in XML-Dateien gespeicherten Sensoren vom RP-Provider gelesen. Dieser speichert die Daten als Resource-Properties und macht sie durch das GT4-System verfügbar. Dadurch werden sie dem WebMDS-Portal zur Verfügung gestellt. Auf diesem Portal kann der Benutzer die Daten einsehen. Das GT4-System und das WebMDS sind Teil der GRID-Domaine am LRZ. Außerdem in der Grafik dargestellt ist, dass der Administrator bisher per `ssh` auf das System zugreifen muss, um die Sensoren zu konfigurieren.

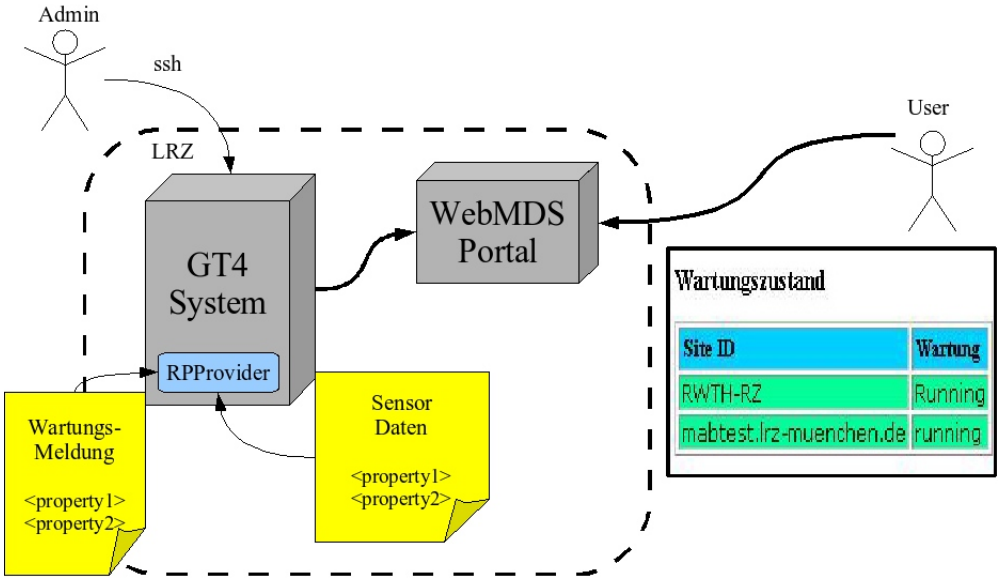

Abbildung 2.1: Systemüberblick

2.1 Bestehende Konzepte

Wie in der Grafik 2.1 erkennbar, werden die Sonsor-Daten in Dateien gespeichert und von einer Komponente namens RP-Provider gelesen.Bisher werden diese Dateien entweder von Hand verändert oder durch das Hilfsprogramm `infoprovider.pl` aus einer Property-Datei erstellt. Die Property-Datei lässt dabei für einige Felder eine Konfiguration zu. Im Listing 2.1 sieht man ein Beispiel dieser Konfiguration. Wie man dabei sehen kann, werden die Einträge für die resultierende Datei als Properties gesetzt. So setzt beispielsweise die Einstellung `site.name=Mabtest` das Element `Name` unter dem Element `Site` auf den Wert “Mabtest”.

Listing 2.1: site.conf

```
########################################################################
# SITE CONFIGURATION
#
# GeoMaint Geo- and Maintenance-Information Provider
#
# parts of this work have been funded by the German Federal Ministry of
# Education and Research under contract 01 AK 800 B (D-Grid)
#
# authors: jrad@RZ.RWTH-AACHEN.DE, timo.baur@lrz-muenchen.de
########################################################################

site.uniqueid=mabtest.lrz-muenchen.de
# should be an URI, can be just the domain name as used in former globus data

site.name=Mabtest
site.description=Monitoring Testsite
site.usercontact=mab@dgrid.de
site.admincontact=mab@dgrid.de
site.securitycontact=mab@dgrid.de

#
# Geolocation
site.location=Garching, Deutschland
site.latitude=48.26166
site.longitude=11.66638
site.web=http://mabtest.lrz-muenchen.de
site.sponsor=LRZ & BMBF
#
# Maintenance
#
# Please use the following syntax for the maintenance-string:
#
# site.maintenance::<state><text>
# state::{0|1|2|3}
# with the state-semantics:     0: ok  1: upcoming maintenance   2: maintenance
 3 or other: somethings broken
# text::{<>|running}
#site.maintenance=3unconfigured site sensor
#site.maintenance=2cluster is down for maintenance until 7pm
#site.maintenance=1maintenance today from 5pm to 7pm
site.maintenance=0running
```

Auf diese Weise ist es zwar möglich eine Wartungsmeldung zu konfigurieren, die Property-Datei ist jedoch nur wenig benutzerfreundlicher als das Ergebnis, die XML-Datei aus Listing 2.2, selbst.

Egal ob von Hand oder durch ein Hilfsprogramm erstellt, müssen Die XML-Dateien auf jeden Fall in einem festgelegten Format vorliegen. Aktuell wird ein kleiner Teil des Schemas `Glue` genutzt. Es ist mit dem Namespace
`http://infnforge.cnaf.infn.it/glueinfomodel/Spec/V12/R2`
referenziert. `Glue` bietet durch eine Definition des Elements `Site` die Möglichkeit eine einfache Wartungsmeldungen zu verpacken. Diese könnte beispielsweise wie in Listing 2.2 aussehen.

Listing 2.2: sensor-rp.xml

```
<Site UniqueID="mabtest.lrz-muenchen.de"
        xmlns="http://infnforge.cnaf.infn.it/glueinfomodel/Spec/V12/R2">
  <Description>Monitoring Testsite</Description>
  <Latitude>48.26166</Latitude>
  <Location>Garching, Deutschland</Location>
  <Longitude>11.66638</Longitude>
  <Name>Mabtest</Name>
  <OtherInfo>0running</OtherInfo>
  <SecurityContact>mab@dgrid.de</SecurityContact>
  <Sponsor>LRZ &amp;amp; BMBF</Sponsor>
  <SysAdminContact>mab@dgrid.de</SysAdminContact>
  <UserSupportContact>mab@dgrid.de</UserSupportContact>
  <Web>http://mabtest.lrz-muenchen.de</Web>
</Site>
```

Neben den komplizierten Dateien besteht außerdem das Problem, dass zum Konfigurieren ein direkter Zugriff auf die Dateien (meist Einloggen per SSH) notwendig ist. Außerdem können Sensoren dadurch lediglich verändert aber nicht abgeschaltet werden.

2.2 Resource-Property-Provider

Die Schnittstelle für die Sensoren, um in das GT4-System zu gelangen, ist in jedem Fall der Resource-Property-Provider (RP-Provider). Um die Aufgabe des RR-Providers zu verstehen, muss zunächst der Begriff Resource-Property geklärt werden.

Das Globus-System stellt eine Reihe von WebServices zur Verfügung. Die Grundlage, auf der diese WebServices aufbauen ist `SOAP`. Die wesentliche Erneuerung gegenüber herkömlichen WebServices von `SOAP` ist, dass die WebServices zustandsbasiert sind. Dazu wurde das Framework der Resource-Properties entwickelt. Wenn man jeden WebService als eine Resource ansieht, sind Resource-Properties die zustands beschreibenden Parameter des Services.

Um dies bildlicher zu verdeutlichen kann man sich die Situation in einer Java-Umgebung vorstellen. Eine Ressource entspricht hierbei einer Java Klasse. Der Zustand der Java-Klasse wird durch deren Variablen beschrieben. Die “public” Variablen einer Klasse sind also analog zu den öffentlichen Resource-Properties einer Ressource anzusehen.

Durch standartisierte Getter- und Setter-Methoden ist es möglich die Resource-Properties eines WebServices zu lesen und zu schreiben. Durch geeignete Policy-Einstellungen kann der Zugriff natürlich beschränkt werden.

Der RP-Provider ist, seit der Version 4.1 des Globus Toolkits, fester Teil des Systems. Er kann XML-Dateien, wie sie im ersten Teil des Kapitels erstellt wurden lesen und als Resource-Property dem System zur Verfügung stellen. Auf diese Weise ist es auf sehr einfache Art und Weise möglich, neue Resource-Properties in das System zu schleusen, ohne dafür einen eigenen WebService schreiben zu müssen.

2.3 WebMDS Portal

Die Sensordaten, die wir mit Hilfe des RP-Providers in das System gebracht haben, müssen nun für den Benutzer sichtbar gemacht werden. Für diese Aufgabe gibt es das WebMDS Portal. Dieses, aus dem internet erreichbare Web-Portal ließt die vorher eingestellten Resource-Properties aus dem GT4-System aus. Wie man ebenfalls in der Grafik 2.1 erkennen kann, sieht der Benutzer eine Internetseite mit den vorher eingestellten Wartungsmeldungen. In diesem Fall stehen keine Wartungen an, der Status steht daher auf `running`.

Auf analoge Art und Weise könnte man das bestehende System so konfigurieren, dass auch andere Daten, z.B. Prozessorauslastung, Kerntemperatur oder Speicherbelegung dargestellt werden. Diese dynamischen Daten

könnten vor beim Einlesen des RP-Providers in die bestehenden XML-Dateien eingepflegt werden. Es bleiben jedoch bestimmte statische Werte, z.B. Name des Betreibers oder Standort, die gepflegt werden müssen. In dieser Arbeit geht es insbesondere um die Konfigurierung der statischen Daten und wie diese vereinfacht werden kann.

3 Notwendige Erweiterungen

Nachdem im letzte Kapitel die Ausgangssituation dargestellt wurde, soll im folgenden Kapitel erläutert werden, wie das System erweitert werden soll. Dazu werden zunächst in 3.1 die Anwendungsfälle für die zu bauende Systemerweiterung definiert. Im zweiten Teil wird auf weitere technische Anforderungen eingegangen.

3.1 Anwendungsfälle

Die Funktionalität des Sensor Konfigurators wird zunächst durch das Anwendungsfall-Diagramm aus Grafik 3.1 skizziert. Hier sieht man neben vier UseCases auch drei Akteure. Als Akteure sind der Administrator und ein normaler Benutzerangegeben. Ein Benutzer kann in diesem Kontext auch ein Globus-Administrator sein, der jedoch keinen direkten Zugriff auf das System hat.

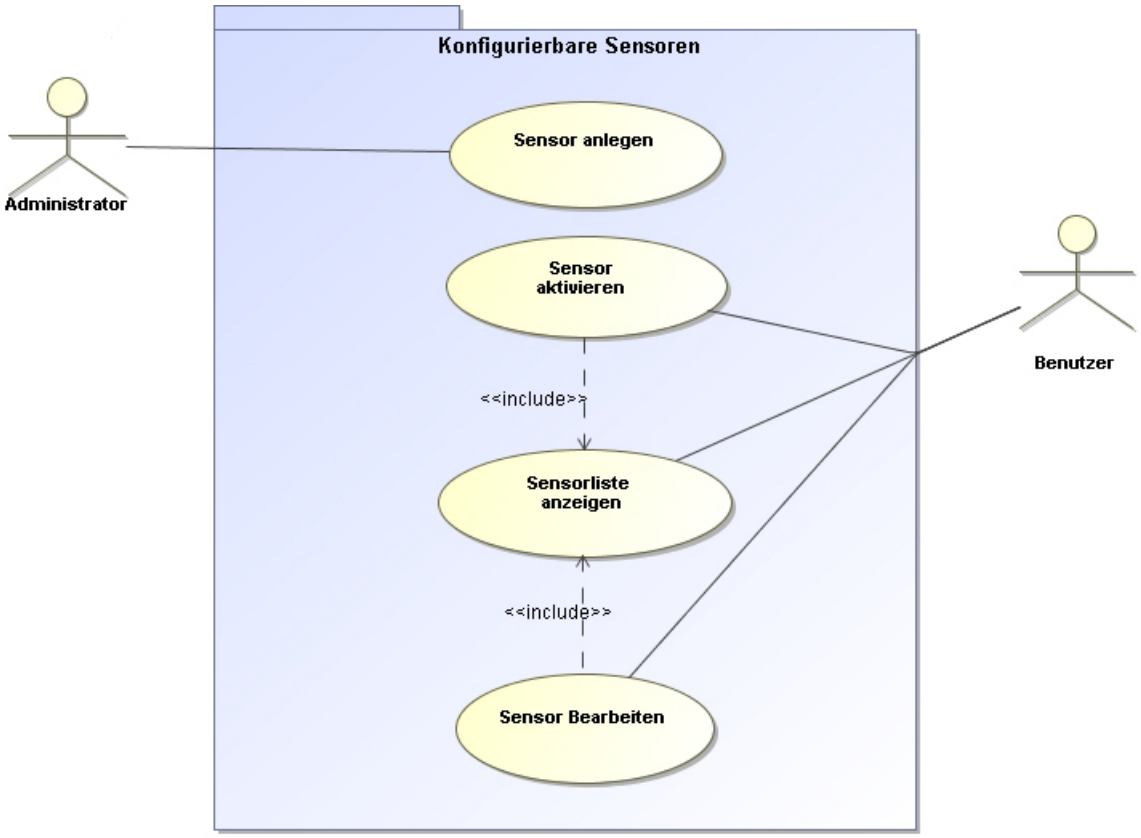

Abbildung 3.1: Use Case Diagramm

Die in der Grafik 3.1 angegebenen UseCases sollen in den folgenden Tabellen 3.1 bis 3.4 genauer beschrieben werden.

Der erste UseCase aus Tabelle 3.1 beschreibt das Anlegen eines neuen Sensors. Dies ist der einzige Anwendungsfall, bei dem weiterhin ein Administrator beteiligt ist. Der Administrator loggt sich am Server ein und legt den Sensor an.

UseCase	Sensor anlegen
Kurzbeschreibung	Ein Administrator legt einen neuen Sensor an, der vom Benutzer konfiguriert werden kann
Auslöser	Es wird ein neuer Sensor benötigt
Vorbedingung	Das System muss betriebsbereit sein
Nachbedingung	Ein neuer Sensor ist bereit zur Konfiguration
Beteiligte Akteure	Administrator
Standardablauf	1. Der Administrator loggt sich per SSH am Server ein 2. Er wechselt in das Verzeichnis für die Sensoren 3. Für den neuen Sensor wird ein neues Unterverzeichnis angelegt 4. In dem neuen Verzeichnis werden die benötigten Dateien angelegt
Varianten	-

Tabelle 3.1: UseCase: Sensor Anlegen

Der zweite UseCase ist in Tabelle 3.2 beschrieben. Er legt fest, wie der Benutzer zu einer Übersicht über alle verfügbaren Sensoren kommt.

UseCase	Sensorenliste anzeigen
Kurzbeschreibung	Der Benutzer erhält am Frontend eine Übersicht über die installierten Sensoren.
Auslöser	Benutzer möchte installierte Sensoren sehen um weitere Aktionen durchzuführen
Vorbedingung	Das System muss betriebsbereit sein
Nachbedingung	Sensoren werden angezeigt
Beteiligte Akteure	Benutzer
Standardablauf	1. Der Benutzer gibt im Frontend die Adresse für das Backend an 2. Das Frontend verbindet sich mit dem Backend 3. Das Frontend fragt das Backend nach den verfübaren Sensoren 4. Das Backend gibt eine Liste mit allen verfügbaren Sensoren zurück 5. Die Liste wird für den Benutzer aufbereitet und im Frontend angezeigt
Varianten	Das Backend ist nicht erreichbar: Der Benutzer erhält eine Fehlermeldung und kann es erneut probieren

Tabelle 3.2: UseCase: Sensorenliste anzeigen

Im dritten Anwendungsfall, Tabelle 3.3 wird beschrieben, wie ein Sensor bearbeitet werden kann. Dies geschieht wie bereits im letzten UseCase durch den Benutzer.

Der vierte und letzte UseCase aus Tabelle 3.4 definiert, wie ein Sensor aktiviert bzw. deaktiviert werden kann. Wie in Grafik 3.1 außerdem ersichtlich, hängen die UseCases aus 3.3 und 3.4 von dem UseCase “Sensorenliste anzeigen” ab.

UseCase	Sensor bearbeiten
Kurzbeschreibung	Ein definierter Sensor soll bearbeitet werden. Dazu sollen bestimmte Felder des Sensors verändert und gespeichert werden.
Auslöser	Benutzer möchte einen Sensor verändern
Vorbedingung	Sensor muss angelegt sein
Nachbedingung	Veränderter Sensor ist gespeichert und wird im WebMDS angezeigt
Beteiligte Akteure	Benutzer
Standardablauf	1. Benutzer öffnet im Frontend die Sensoren-Übersicht (siehe Use-Case “Sensorenliste anzeigen”) 2. Der Benutzer wählt den zu bearbeitenden Sensor aus 3. Die im Sensor definierten Felder werden ihm angezeigt 4. Er kann soweit erlaubt die Felder verändern 5. Durch auswählen von “Speichern” werden die Veränderungen gespeichert 6. Die Ansicht geht zurück auf die Liste der verfügbaren Sensoren
Varianten	A.5 Der Benutzer bricht die Aktion ab A.6 Änderungen werden verworfen, zurück zur Übersicht

Tabelle 3.3: UseCase: Sensor bearbeiten

UseCase	Sensor aktivieren
Kurzbeschreibung	Der Benutzer aktiviert einen Sensor aus der Liste der verfügbaren Sensoren
Auslöser	Benutzter möchte einen Sensor aktivieren
Vorbedingung	1. Sensor ist deaktiviert 2. Sensor wird nicht im WebMDS Portal angezeigt 3. Sensor muss angelegt sein
Nachbedingung	1. Sensor ist aktiviert 2. Sensor wird im WebMDS Portal angezeigt
Beteiligte Akteure	Benutzter
Standardablauf	1. Benutzer öffnet im Frontend die Sensoren-Übersicht (siehe Use-Case “Sensorenliste anzeigen”) 2. Benutzer aktiviert einen Sensor 3. Das Frontend schickt die Änderung an das Backend
Varianten	Sensor ist aktiviert und soll deaktiviert werden: Vorbedingung 1 und 2 werden mit Nachbedingungen getauscht, im Ablauf Punkt 2 wird der Sensor deaktiviert

Tabelle 3.4: UseCase: Sensor aktivieren oder deaktivieren

3.2 Technische Anforderungen

Da die Anwendungsfälle nun deutlich sind, muss noch festgelegt werden, welche technischen Anforderungen gestellt werden.

Im wesentlichen geht es im Backend um die Bearbeitung von zwei Dateien. Dies ist zum einen die XML-Datei mit den, in Kapitel 2 beschriebenen Resource-Properties (siehe auch Listing 2.2) und zum anderen die Konfigurationsdatei des RP-Providers.

Möchte man einen neuen Sensor anlegen, muss also die besagte XML-Datei angelegt werden. Außerdem ist der RP-Provider so zu konfigurieren, dass er den neuen Sensor regelmäßig einließt und aktualisiert. Da der RP-Provider möglichst universell gehalten ist, erweist sich diese Konfiguration als relativ unübersichtlich. Im Listing 3.1 wird exemplarisch der Konfigurationsteil des RP-Providers für einen Sensor dargestellt. Auf die Datei wird später im Kapitel 5.1.1 noch etwas genauer eingegangen.

Listing 3.1: sensor-rp-config.xml

```
<ns1:configArray xsi:type="ns1:resourcePropertyProviderConfig"
    xmlns:ns1="http://mds.globus.org/rpprovider/2005/08"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001XMLSchema">

  <ns1:resourcePropertyName xsi:type="xsd:QName"
      xmlns:glue="http://infnforge.cnaf.infn.it/glueinfomodel/Spec/V12/R2">
    glue:Site
   </ns1:resourcePropertyName>
    <ns1:resourcePropertyImpl xsi:type="xsd:string">
     org.globus.mds.usefulrp.rpprovider.SingleValueResourcePropertyProvider
    </ns1:resourcePropertyImpl>
    <ns1:resourcePropertyElementProducers
     xsi:type="ns1:resourcePropertyElementProducerConfig">
    <ns1:className xsi:type="xsd:string">
     org.globus.mds.usefulrp.rpprovider.producers.ExternalProcessElementProducer
    </ns1:className>
    <ns1:arguments xsi:type="xsd:string">
     cat etc/de_lrz_sensor/sensors/globus.sensor.wartung/sensor-rp.xml
    </ns1:arguments>
    <ns1:period xsi:type="xsd:int">
     300
    </ns1:period>
  </ns1:resourcePropertyElementProducers>
</ns1:configArray>
```

Wie man in der Konfiguration sieht wird als erstes das Top-Element des entstehenden Resource-Properties festgelegt. In diesem Fall `glue:Site`. Im weiteren Verlauf der Konfiguration wird durch den `ExternalProcessElementProducer` festgelegt, dass ein externes Programm den Input liefert. Dieses wird direkt im Anschluss definiert. Letztendlich kann man festlegen, in welchem Intervall das Programm gestartet werden soll.

Möchte man erwirken, dass ein Sensor überhaupt nicht mehr angezeigt wird, muss der entsprechende Teil aus der RP-Provider Konfiguration entfernt werden.

Um etwas an den Daten des Sensors zu verändern muss entsprechend die Datei aus Listing 2.2 verändert werden.

Um diese Schritte zu automatisieren, wird ein Controller benötigt, der die entsprechenden Dateien für jeden Sensor verwaltet und anpasst. Der Controller benötigt außerdem Zugriff auf die Konfigurations-Datei des RP-Providers um Sensoren zu aktivieren bzw. zu deaktivieren.

Eine wichtige technische Anforderung war außerdem, dass Frontend und Backend auf getrennten Systemen laufen können. Dazu wird eine Schnittstelle Benötigt die möglichst gut in das bestehende System integriert

werden kann. Dadurch sollte außerdem ermöglicht werden, dass die neue Funktionalität des Backends auch von anderen System benutzt werden kann.

4 Architektur und Design

Aus den Anforderungen geht hervor, dass dieses Projekt in zwei unterschiedlichen Teile gegliedert werden kann. Zum einen ist dies ein Backend, das die Sensoren auf dem Server konfiguriert und verwaltet. Dieses Backend wird SensorConfigurator genannt. Der zweite Teil ist ein Frontend in einem WebServer, welches auf das bereitgestellte Backend (SensorConfigurator) zugreift und dem Benutzer eine komfortable Oberfläche bietet.

In Abbildung 4.1 sind die verschiedenen Komponenten dargestellt. Auf der einen Seite der WebServer mit dem darin liegenden Frontend und auf der anderen Seite das Backend “SensorConfigurator” im GT4-System. Außerdem auf dem GT4-Server liegt der RP-Provider. Dieser wird vom “SensorConfigurator” für die Sensoren eingestellt. Dadurch kann der RP-Provider die konfigurierten Sensoren lesen.

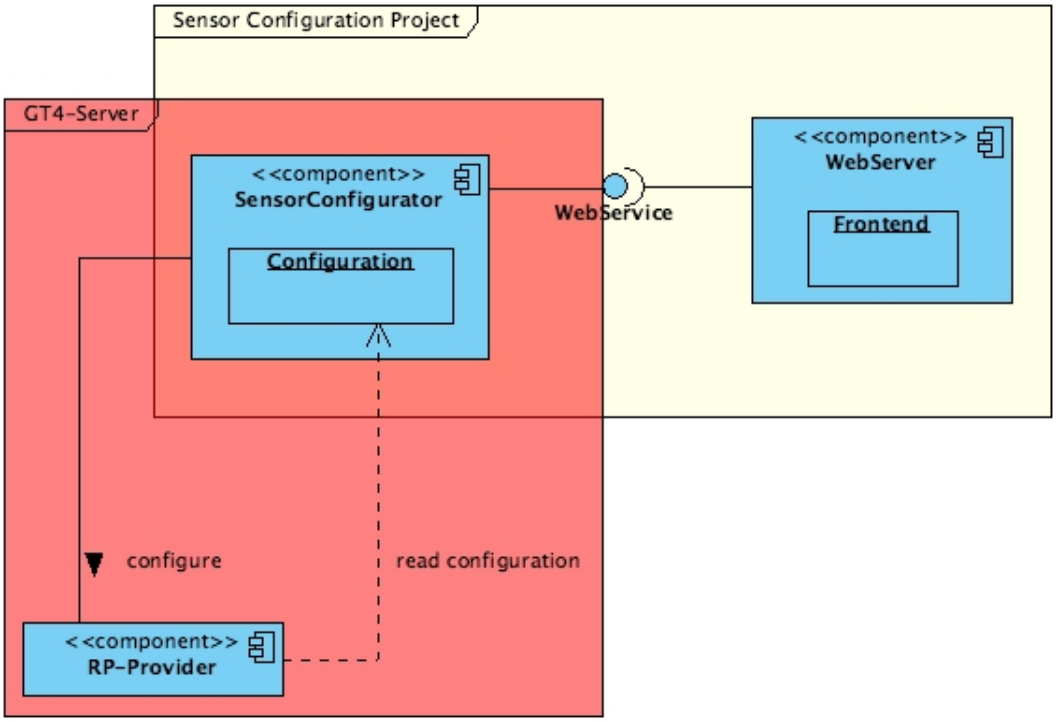

Abbildung 4.1: Komponenten des Systems

Der Benutzer greift lediglich auf das vom WebServer bereitgestellte Frontend zu. Der WebServer hat die Aufgabe die vom SensorConfigurator bereitgestellten Daten in ein anschauliches Format zu übertragen und zurück zu leiten.

Im nun folgenden ersten Abschnitt wird zunächst die genauere Architektur des SensorConfigurators beschrieben. Danach wird im zweiten Abschnitt des Kapitels ein Blick auf das Servlet geworfen.

4.1 SensorConfigurator

Der SensorConfigurator ist das Bindeglied zwischen dem Web-Frontend und dem RP-Provider. Er zeigt sich nach außen als ein GT4-WebService und ist somit auch universell von anderen Programmen nutzbar. Durch den Einsatz eines GT4-konformen WebService wird die technische Anforderung, das Frontend vom Backend zu trennen erfüllt. Er ist zudem nahtlos in das bestehende System zu integrieren.

Die Aufgaben des Konfigurators sind das Auslesen und Verändern von Sensorkonfigurationen. Insbesondere müssen die Daten für den RP-Provider aufbereit werden. Außerdem muss der RP-Provider so konfiguriert werden, dass er die aktiven Sensoren ausließt. Auf diese Weise kann man außerdem Sensoren aktivieren und deaktivieren.

In Abbildung 4.2 wird verdeutlicht, dass jeder Sensor aus drei Dateien besteht. Diese werden jeweils in einem eigenen Ordner abgespeichert. Der SensorConfigurator liest bzw. schreibt die nötigen Informationen aus und in die Dateien. Dabei ist zu bemerken, dass die sensor-rp.xml eine Transformation aus der sensor-data.xml ist. Sie muss daher nicht vom Administrator angelegt werden. Die beiden anderen Dateien werden beim Einrichten eines neuen Sensors vom Administrator bereit gestellt. Der RP-Provider ließt letztendendes die sensor-rp.xml.

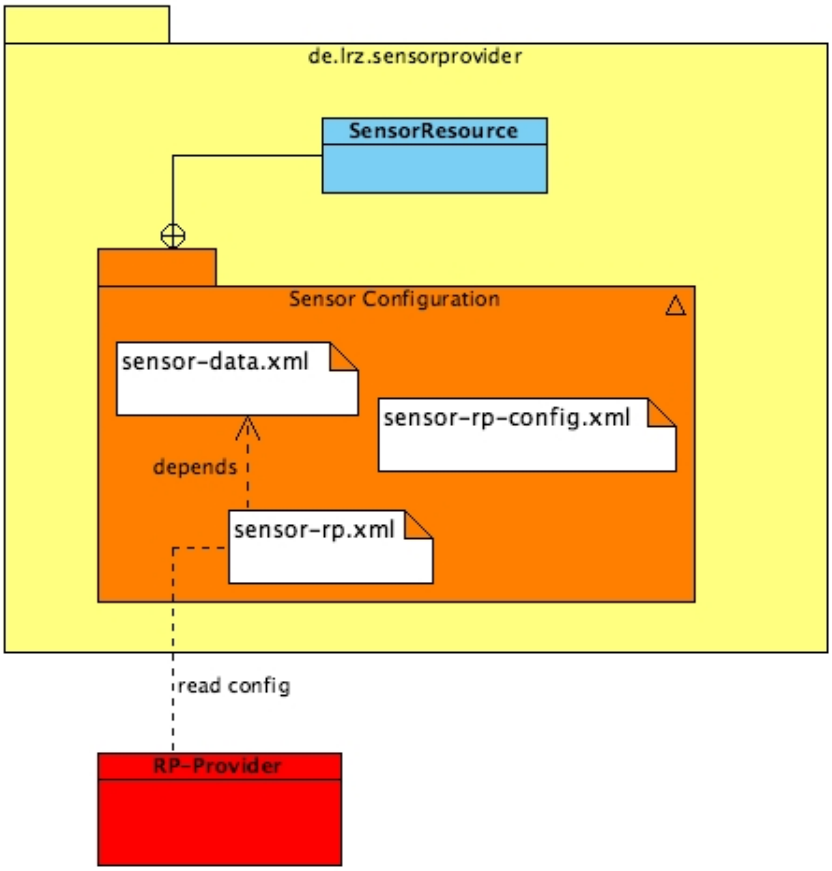

Abbildung 4.2: Dateien eines Sensors

Im folgenden wird auf diese drei Dateien im einzelnen eingegangen.

4.1.1 sensor-data.xml

Die erste Datei heißt “sensor-data.xml” und beinhaltet das Schema der Monitoringdaten eines vorkonfigurierten Sensors. Sie enthält sowohl die Beschreibung als auch die Daten der XML-Elemente, die später dem

RPProvider zur Verfügung gestellt werden. Das Prinzip ähnelt dem eines XML-Schemas. Es wird durch Angabe von Elementen und dazugehörigen Attributen die Struktur der Sensordaten widergespiegelt. Man orientiert sich also an dem Ergebnis, der entstehenden Resource-Property.

Zusätzlich können Einschränkung für das Verändern bestimmter Datensätze festgelegt werden. So können Standardwerte oder Auswahllisten für bestimmte Felder definiert werden. Es ist außerdem möglich, konstante Felder einzusetzen, deren Inhalt vom Benutzer nicht verändert werden darf.

Das XML-Schema für die `sensor-data.xml` wurde rekursiv definiert. In Abbildung 4.3 ist schematisch dargestellt, dass der Element-Typ die Eigenschaften vom Attribut-Typ erbt. In den abgebildeten XML-Typen ist dargestellt, welche Attribute und Unterelemente für dieses Element zulässig sind. Durch die Pfeile ist erkennbar, wie die Typen voneinander abhäig sind.

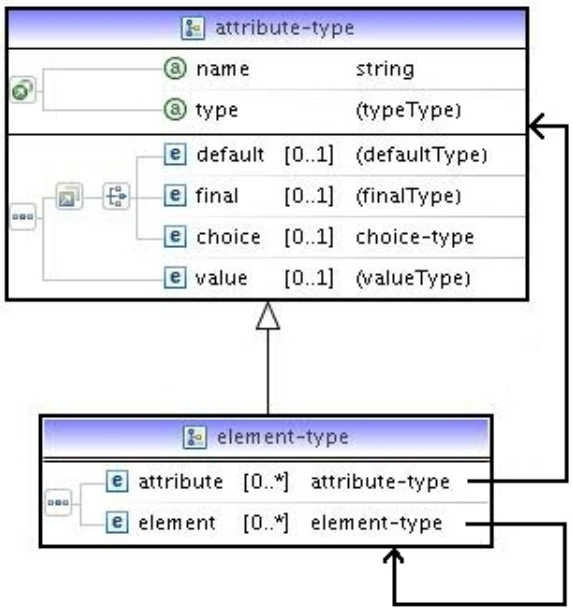

Abbildung 4.3: Schematischer Aufbau der sensor-data.xml

Grundlage für die Definition des Schemas ist der Typ `attribute-type`. Er legt fest, dass jeder Eintrag einen Namen und einen Typ hat. Diese werden durch die Attribute `name` und `type` festgelegt. Als Typ besteht die Möglichkeit zwischen `text`, `number` und `decimal` zu wählen.

Außerdem hält die Definition des attribute-types fest, dass jeder Eintrag eine der folgenden Einschränkungen enthalten darf: `default`, `final` oder `choice`. Die ersten beiden Einschränkungen sind quasi selbsterklärend. Mit ihnen kann man einen Standardwert festlegen oder einen nicht zu veränderten Wert eintragen. Hinter der dritten Möglichkeit verbirgt sich eine Auswahlliste. Durch Angabe von beliebig vielen `option`-Unterelementen kann man eine Liste von möglichen Optionen definieren.

Neben einer Einschränkung kann in dem jeweiligen Eintrag unter dem Element `value` der tatsächlich vom Benutzer eingestellte Wert gespeichert werden. Ist `value` nicht gesetzt soll automatisch der Wert aus dem Element `default` gelesen werden. Ist ein `final` gesetzt wird `value` ignoriert.

Die bisherigen Eigenschaften sind nicht nur für die Beschreibung von Attributen sinnvoll, sondern gleichermaßen für ganze XML-Elemente interessant. Da Elemente jedoch noch mehr können müssen, sieht die Architektur vor, dass der `element-type` von `attribute-type` erbt. Wie in Abbildung 4.3 zu sehen erweitern die Elemente damit die Funktionalität der Attribute. Dies geschieht zum einen dadurch, dass jedes Element eine beliebige Anzahl von Attributen enthalten kann und zum anderen eine beliebige Anzahl von weiteren Unterelementen. Man erhält einen rekursiv definierten `element-type` mit beliebig tiefen Hierarchiestufen.

Damit haben wir die Möglichkeit, jede XML-Datei auf einfache Art und Weise zu definieren. Ferner können wir für die Eingaben des Benutzers bestimmte Einschränkungen vornehmen und Hilfestellungen geben.

Um grundsätzliche Daten zu speichern und ein ein einheitliches Wurzelelement zu definieren, sind die Elemente von dem Root-Element `provider-data` eingefasst. Dieses Element enthält die als Attribute den Namen des Sensors und den Namespace, in dem die definierten Elemente liegen sollen. Die Attribute heißen `sensorname` und `targetNamespace`. Das Element `provider-data` muss genau ein Unterelement vom Typ `element-type` mit dem Namen `element` enthalten.

Damit dieser Aufbau besser verdeutlicht wird, ist in Listing 4.1 ein fiktives Beispiel abgebildet. Es enthält ein Element namens "element_1" mit zwei Attributen (attr_1 und attr_2) und einem Unterelement. Das erste Attribut hat als Standartwert 65, es wurde auch noch kein anderer Wert eingetragen (kein `value` vorhanden). Das zweite Attribut darf nicht geändert werden. Es enthält fest eingestellt den Wert "do not change". Das Unterelement ist vom Typ Text. Dieser Text kann aus einer festgelegten Auswahl (Option 1 bis 3) getroffen werden. Wie der Wert in `value` zeigt, wurde bereits "Option 2" ausgewählt.

Listing 4.1: sensor-data-beispiel.xml

```
<provider-data
         targetNamespace="http://foo.bar"
         sensorname="name_of_sensor">
        <element name="element_1" type="text">
                <attribute name="attr_1" type="number">
                        <default>65</default>
                </attribute>
                <attribute name="attr_2" type="text" >
                        <final>do not change</final>
                </attribute>
                <element name="subelement" type="text">
                        <choice>
                                <option>Option 1</option>
                                <option>Option 2</option>
                                <option>Option 3</option>
                        </choice>
                        <value>
                                Option 2
                        </value>
                </element>
        </element>
</provider-data>
```

Wie eingehend bereits erwähnt ist dabei die hierarchische Struktur der Resource-Property maßgebend. Durch Transformation soll es möglich sein, aus dieser Datei, die für den RP-Provider notwendige XML-Datei zu generieren.

Aus dem gegebenen Beispiel in Listing 4.1 würde nach der Transformation die XML-Datei aus Listing 4.2 entstehen.

Listing 4.2: sensor-rp-beispiel.xml

```
        <element_1 xmlns="http://foo.bar" attr_1="65" attr_2="do␣not␣change">
                <subelement>
                        Option 2
                </subelement>
        </element_1>
```

Weitere Erklärungen und Beispiele werden in Kapitel 5.1.1 aufgeführt.

4.1.2 sensor-rp.xml

Die zweite Datei nennt sich `sensor-rp.xml`. Sie muss nicht vom Administrator erstellt werden sondern wird vom SensorConfigurator erstellt. Die `sensor-rp.xml` enthält ebenfalls die Daten eines Sensors, jedoch in einem für den RPProvider lesbaren Format. Es stellt damit die XML-Interpretation der Resource-Property für den jeweiligen Sensor dar. Da bereits die `sensor-data.xml` die notwendige Struktur vorgibt, kann sie durch eine relativ einfache XSLT-Transformation generiert werden. Nach welchem Schema die Datei aufgebaut sein muss bestimmt das verwendete Resource-Property.

In Listing 4.2 ist ein fiktives Beispiel für eine generierte `sensor-rp.xml` dargestellt.

4.1.3 sensor-rp-config.xml

Die dritte Datei, mit dem Namen `sensor-rp-config`, enthält die Konfiguration für den RPProvider. Im wesentlichen wird das die Information sein, in welchem Abstand die vorherige Datei (sensor-rp.xml) gelesen werden muss. Es können aber an dieser Stelle auch Drittprogramme gestartet werden um dynamische Informationen in die Resource-Properties mit einfliesen zu lassen. Wird ein Sensor aktiviert, kopiert der SensorConfigurator den Inhalt dieser Datei an die richtige Stelle in der Konfiguration des RPProviders. Soll der Sensor dagegen deaktiviert werden, löscht der Konfigurator den entsprechenden Teil wieder raus. Eine weitere Beschreibung über den Aufbau dieser Datei befindet sich im Kapitel 5.1.1.

4.1.4 Arbeitsablauf

Durch die vorgestellten Konfigurationsdateien lassen sich beliebige Sensoren erstellen. Außerdem ist es möglich, Sensoren zu aktivieren bzw. zu deaktivieren.

Der Arbeitsablauf des SensorControllers ist nun relativ einfach zu beschreiben. Jeder Sensor wird in einem eigenen Ordner mit den dazugehörigen XML-Dateien auf der Festplatte gespeichert. Um festzustellen, welche Sensoren verfügbar sind, hat der Controller lediglich jeden dieser Ordner nach der dazugehörigen sensor-data.xml zu durchsuchen. In dieser kann er den Namen und weitere Informationen über den Sensor ablesen.

Muss der Controller dagegen wissen, ob ein bestimmter Sensor aktiv ist, muss er die sensor-rp-config.xml des entsprechenden Sensors laden und überprüfen, ob diese in der rp-provider-config.xml eingetragen ist. Analog muss er zur Aktivierung eines deaktivierten Sensors den Inhalt der sensor-rp-config.xml in die rp-provider-config.xml eintragen bzw. zum Deaktivieren löschen.

4.1.5 Schnittstelle nach außen

Die durch die bisherige Architektur erreichte Funktionalität muss nun in einem weiteren Schritt, mit Hilfe eines WebServices, nach außen angeboten werden.

Der WebService wurde nach dem Factory-Instance-Pattern[1] konzipiert. Dabei wurde jedoch die Factory zumindest von der Namensgebung durch einen “Finder” ersetzt, da die Sensoren nicht erstellt, sondern lediglich im System gefunden werden müssen.

Die Abbildung 4.4 zeigt schematisch die Architektur des Webservices. Hierbei sieht man, dass der Service aus zwei WebServices besteht, dem `FinderService` und dem `SensorService`. Beide WebServices sind dem Client bekannt. Jedoch erfüllt der `SensorService` nur seinen Dienst, wenn er mit einer speziellen Referenz aufgerufen wird.

Der Finder besitzt lediglich eine Methode ohne Parameter: `availableSensors()`. Diese liefert als Ergebnis ein Array von `EndPointReference`. Um diese Referenzen zu bauen, leitet er die Anfrage an die `ResourceHome` weiter. Die `ResourceHome` durchsucht das Sensor-Verzeichnis auf der Festplatte nach

[1] Für eine genaue Beschreibung des Factory-Instance-Patterns, siehe [BS 06]

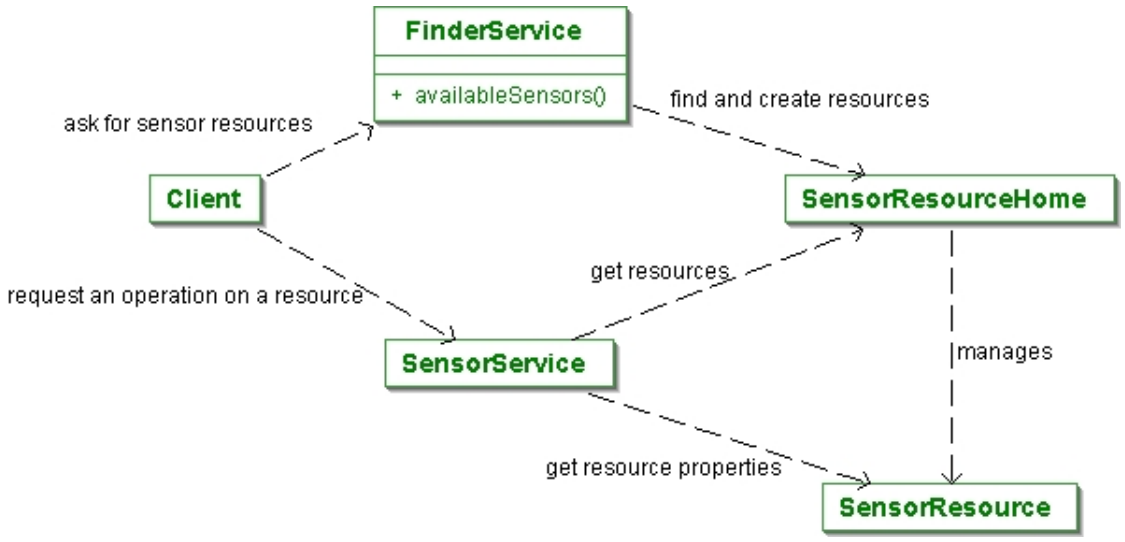

Abbildung 4.4: Aufbau des WebServices

Sensoren. Es wird dabei eine Syntaxüberprüfung durchgeführt, um Unterverzeichnisse mit falschen Dateien auszuschließen. Für jeden gefundenen Sensor legt die `ResourceHome` eine neue Ressource an und speichert diese unter einem Schlüssel. Die Schlüssel zu den verschiedenen Sensoren werden an den Finder zurück gegeben. Dieser baut aus jedem Schlüssel, zusammen mit der Adresse für den `SensorService` eine `EndPointReference`.

Möchte der Client die Funktionen eines Sensors nutzen, baut der mit der passenden Referenz eine Verbindung zum WebService `SensorService` auf. Anhand des, in der Referenz gespeicherten Schlüssels, kann der SensorService erkennen, welcher Sensor angesprochen wird. Die Eigenschaften des Sensors sind indes selbst als Resource-Properties zu sehen. Durch die standardisieren Getter- und Setter-Methoden der Resource-Properties des Services, soll es dem Client möglich sein die Attribute zu lesen und bestimmte auch zu schreiben.

Die Tabelle 4.1 beschreibt die Resource-Properties, die durch den WebService `SensorService` für jeden Sensor zur Verfügung gestellt werden.

Name	Typ	Beschreibung	Lesen/Schreiben
active	boolean	True, wenn Sensor eingeschaltet	RW
Name	string	Name des Sensors	RO
ResourcePropertyName	string	Name der verwendeten Resource-Property	RO
Location	string	Verzeichnis des Sensors, relativ zum Sensor-Root-Verzeichnis	RO
SensorData	string	Daten des Sensors, `sensor-data.xml`	RW

Tabelle 4.1: Resource-Properties des SensorService

4.2 Webfrontend

Im vorigen Kapiteln wurde beschrieben, dass eine Benutzerschnittstelle gebraucht wird. In diesen Kapitel wird die Architektur der Benutzerschnittstelle erklärt, im folgendem auch Frontend oder Webfrontend genant. Diese sollte die Sensoren leicht und intuitiv konfigurierbar machen. Damit die Benutzer in der Lage sind Informationen ohne großen Aufwand anderen Benutzern des Grids berichten können. Als Meldungen kann man dabei zum Beispiel einfache Wartungsmeldungen einer einzelnen Komponente des Grids verstehen. Um den Zugang den Benutzern zu erleichtern sollte die Benutzerschnittstelle von überall erreichbar sein. Eine Benutzerschnittstelle in der Form eines Webfrontends würde sich am besten dafür eignen, da der Benutzer nur einen

Internetanschluss braucht um eine Änderung machen zu können. Um sicher zu stellen das die Applikation auf allen Platformen läuft wird die Programmiersprache Java verwendet.

4.2.1 Java Server Pages (JSP)

Im folgenden Abschnitt wird die Architektur des Webfrontends beschrieben. Das Webfrontend besteht aus Java Server Pages so genanten Servlets. Sun Microsystems hat die JavaServer Pages (JSP) für die Realisierung der Präsentationsschicht einer Webanwendung entwickelt. Sie integrieren Businesskomponenten und existierende Systeme elegant ins Web. Die JSP-Technik basiert auf dem Java-Servlet-API und dient im Wesentlichen zur einfachen Erzeugung von HTML- und XML-Ausgaben eines Webservers. Wie PHP, ASP oder Perl können Autoren direkt HTML oder XML mit einer Skriptsprache mischen. Natürlich hat sich Sun hier für Java entschieden, um so die Möglichkeiten der vorhandenen robusten und zahlreichen Java-APIs einzubringen. Ein Entwicklerteam kann ausschließlich mit der Programmiersprache Java die Logik implementieren. Das Design kann dagegen weiterhin von einem HTML-Autor und Grafikern durchgeführt werden. Damit das Beste der HTML- und Java-Welt zusammenkommen kann, sind in den JSP einige Tags zur Verfügung gestellt, die es erleichtern, ohne eine einzige Zeile Java kleine und auch große Anwendungen zu schreiben. Mit der JSP-Spezifikation sind die Standard-Tags durch so genannte Taglibs sogar erweiterbar [LR 00].

Abbildung 4.5: Architektur der JavaServer Pages [LR 00]

Eine JSP ist eigentlich ein spezielles Servlet, das durch eine JSP-Engine erzeugt wird. Als Servlets bezeichnet man Java-Klassen, deren Instanzen innerhalb eines J2EE Applicationservers Anfragen von Clients entgegen nehmen. Solche Klassen müssen immer die Schnittstelle “javax.servlet.Servlet” oder eine davon abgeleitete (normalerweise “javax.servlet.http.HttpServlet”) implementieren. Der Inhalt der Antworten kann dabei dynamisch, also im Moment der Anfrage, erstellt werden und muss nicht bereits statisch (etwa in Form einer HTML-Seite) für den Webserver verfügbar sein [Wiki 07]. Eine einfache Anfrage an ein Servlet wird in Bild 4.5 beschrieben. Die JSP generiert die Antwort mit Hilfe von JavaBean, wo die Daten, die der Benutzer angefordert hat, gespeichert sind. Die Antwort wird dann an den Benutzer (Browser) zurückgeschickt.

Servlets sind für unsere Applikation deshalb geeignet, da man mit ihnen leicht dynamische HTML Seiten darstellen kann. Die dargestellten Informationen in den Seiten können dann von Benutzern verändern werden. Die Änderungen werden dann vom Servlet gespeichert. Die genauere Architektur ist nach dem Model-View-Control Prinzip aufgebaut und wird genau im nächsten Abschnitt 4.2.2 beschrieben.

4.2.2 Model-View-Controller

In diesem Abschnitt wird der Aufbau des Webfronends nach dem Model-View-Control (MVC) Prinzip beschrieben. Model-View-Control bezeichnet ein Architekturmuster zur Aufteilung von Softwaresystemen in die drei Einheiten: Datenmodell (engl. Model), Präsentation (engl. View) und Programmsteuerung (engl. Controller). MVC ist ein flexibles Programmdesign, das unter anderem eine spätere Änderung oder Erweiterung erleichtern und eine Wiederverwendbarkeit der einzelnen Komponenten ermöglichen soll.

Das Modell enthält die darzustellenden Daten. Die Daten werden über die Webservice Schnittstelle vom Controller geholt und im Modell gehalten. Das Modell kennt weder die Präsentation noch die Steuerung, weiß also

gar nicht, ob und wie es dargestellt und verändert wird. Die Präsentation ist für die Darstellung der relevanten Daten aus dem Modell zuständig. Sie ist nicht für die Weiterverarbeitung der vom Benutzer übergebenen Daten verantwortlich (siehe Steuerung), sondern lediglich für die Beschaffung der Daten aus dem Modell, deren Darstellung in einer JSP und, bei Änderungen im Modell, der passenden Aktualisierung. Die Steuerung verwaltet die Sichten, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und agiert entsprechend.

Ein allgemeines Szenario könnte wie im Bild 4.6 dargestellt ausschauen. Der Benutzer fordert per HTTP-Request eine bestimmte Webseite an. Diese Anfrage löst eine Aktion im Controller aus. Dieser verifiziert die Anfrage und gegebenenfalls die angegebenen Parameter. Der Controller erstellt das entsprechende Model (JavaBean) für die Anfrage. Die Daten kommen z.B. aus einer externen Datenbank. Der Controller leitet den Request an die entsprechende View (JSP Seite) weiter. Die View bereitet die im Model (JavaBean) enthaltenen Daten zur Darstellung im Browser auf. Diese Seite kommt dann per HTTP-Response an den Browser (Client) zurück.

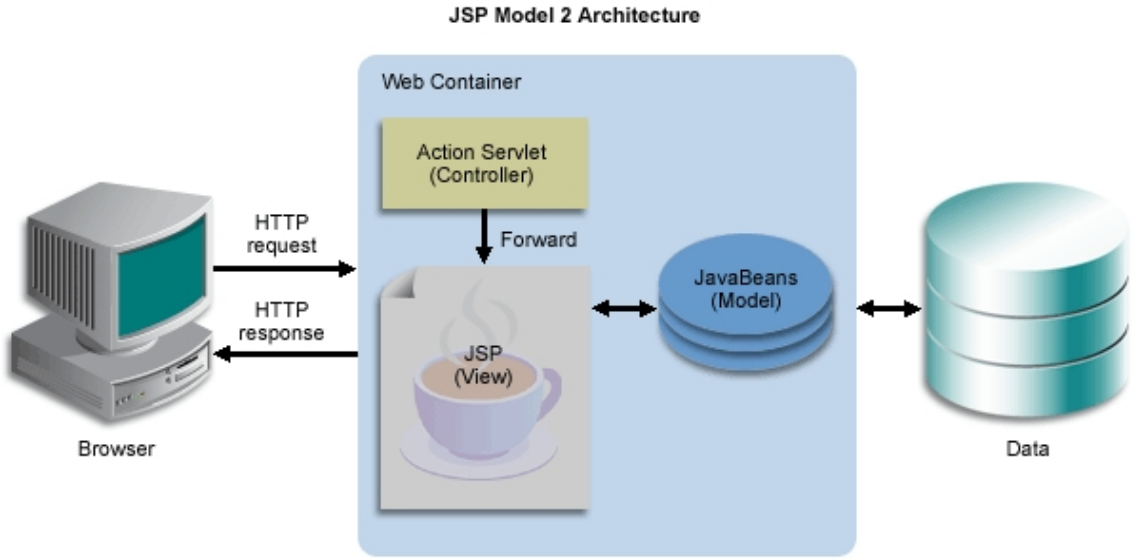

Abbildung 4.6: Model-View-Controller Prinzip bei Struts [Ora 07]

So ähnlich wie im allgemeinen Szenario wird auch in unserem Webfrontend, der Sensor-Konfiguration, ein HTTP-Request empfangen, bearbeitet und beantwortet. Dieses Szenario ist im Bild 4.7 dargestellt. Gehen wir von dem Beispiel aus, das der Benutzer alle Sensoren angezeigt haben will. Dieser fordert mit Hilfe des Browsers die Seite an, auf der alle Sensoren aufgelistet sind. Das Mapping unseres Servlets leitet den Request an den Controller. Um Informationen über alle Sensoren zu haben, stößt er den WebService-Client des Frontends an. Dieser wiederum verbindet sich zu der WebService-Schnittstelle die im Kapitel 4.1.5 beschrieben ist. Mit Hilfe dieser Schnittstelle sind die Sensordaten über das Internet von Überlall erreichbar. Deshalb muss die WebService Schnittstelle nicht auf dem gleichen Rechner laufen wie das Webfrontend. Der Webservice-Client bekommt über die WebService-Schnittstelle die erforderlichen Daten (Zustand, Name, Pfad) jedes Sensors. Diese werden wieder an den Controller weitergeleitet. Der Controller bereitet die Daten für die JSP auf und speichert sie in einem JavaBean. Mit einem Forward auf die JSP, auf der die Sensoren dargestellt werden, wird die JSP mit den Daten aus dem JavaBean erstellt. Die erstellte JSP wird dann als HTTP-Response an den Browser des Benutzers geschickt und angezeigt. Der Benutzer ist in der Lage die Daten zu sehen und zu bearbeiten. Um die Daten auch leicht verändern zu können werden sie in der JSP als Baumstruktur, die in Abschnitt 4.2.3 beschrieben ist, angezeigt.

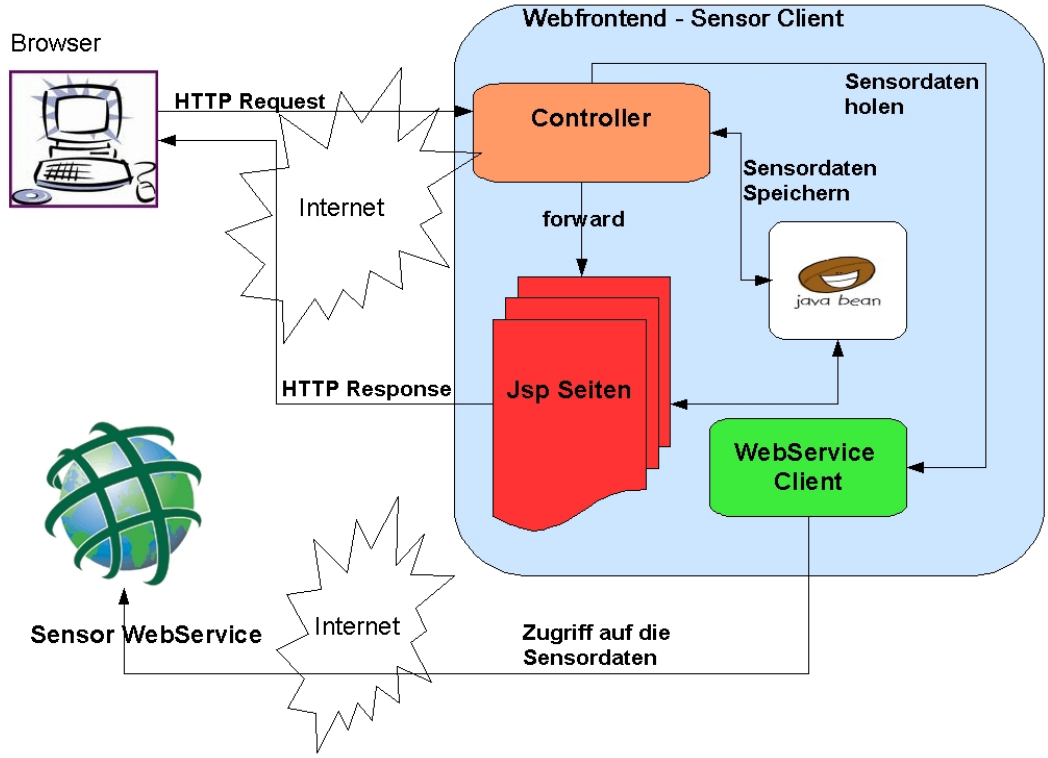

Abbildung 4.7: Architektur des Sensor-Client Webfrontends

4.2.3 Baumdarstellung

Im letzen Abschnitt wurde erklärt wie die Daten vom WebService in die JSP Seiten kommen. In diesen Abschnitt wird die Darstellung der Sensordaten beschrieben. Um den Benutzer das Ändern der Sensordaten zu erleichtern, müssen sie auch übersichtlich dargestellt werden. Der Benutzer sollte ohne größeren Aufwand die für ihn Interessante Datensätze finden und auch Ändern können. Damit die Darstellung die genanten Eigenschaften auch erfüllt werden die Sensordaten in einer Baumstruktur angezeigt.

Das Frontend soll in der Lage sein alle Sensoren, die durch das in Abschnitt 4.1.1 vorgestelltes Schema beschrieben sind, in einer JSP darzustellen. Die Sensordaten die über das Webservice geliefert werden entsprechen diesem Schema. Die Sensordaten können Dabei verschieden verschachtelt sein und beliebig viele Attribute haben. Ein Teil der Sensordaten kann zum Bespiel wie in Listing 4.3 ausschauen. In diesen Beispiel könnte in einem Sensor das Element “Adresse” weitere Unterelemente besitzen (Name, Straße, PLZ, Ort). Diese können wiederum durch Attribute weiter verschachtelt sein (z.B. Attribute Vorname und Nachname im Element Name).

Listing 4.3: Beispiel: Sensor Daten

```
<element type="text" name="Adresse">
    <element type="text" name="Name">
       <attribute type="text" name="Nachname">
           <value>Meier</value>
       </attribute>
    <attribute type="text" name="Vorname">
           <value>Hans</value>
       </attribute>
    </element>
    <element type="text" name="Strasse">
```

```
        <value>Marienstr. 5</value>
    </element>
    <element type="text" name="PLZ">
        <value>80753</value>
    </element>
    <attribute type="text" name="Ort">
        <value>Garching</value>
    </attribute>
</element>
```

Wenn man sich das Beispiel aus Listing 4.3 anschaut versteht man nicht auf Anhieb welche Informationen mit diesen XML beschrieben sind. Und der Benutzter braucht ziemlich lange um die Daten auch zu Finden die er ändern will. Die Informationen aus Listing 4.3 können, wie im Bild 4.8 dargestellt werden. Dort wird die Adresse in Form eines Baumes angezeigt. Die Knoten die als Rechtecke dargestellt sind repräsentieren ein "element-Type" aus Listing 4.3. Rechtecke mit runden Ecken sind "attribute-Types". Knoten die als Sechseck (Hexagon) angezeit sind stellen den "value" Wert aus dem Beispiel da. Die Adresse ist als ROOT Element dargestellt mit den Unterelementen Name, Straße, PLZ und Ort. Das Element Name hat wiederum zwei Unterelemente Nachname und Vorname die die Werte Meier und Hans haben. Die anderen drei Elemente haben keine Unterelemente, sondern haben bestimmte Werte und zwar Straße hat den Wert "Marienstr. 5" ,PLZ den Wert "80753" und Ort hat den Wert "Garching".

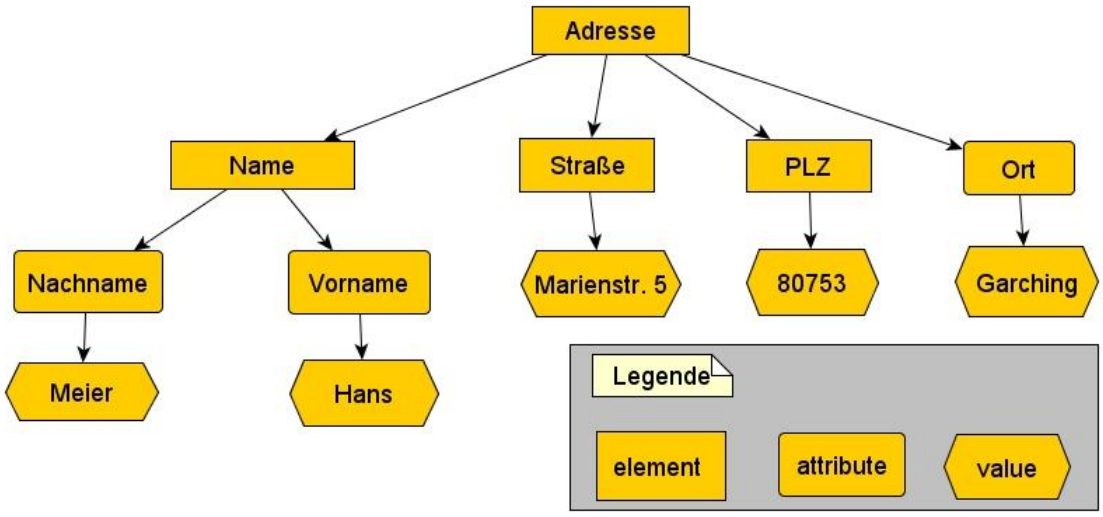

Abbildung 4.8: Beispiel aus Listing 4.3 als Baum dargestellt

Der Benutzer soll eine Art der Baumstruktur im Frontend angezeigt bekommen. Sie ist übersichtlich, auch wenn die Informationen in mehrere Ebenen geschachtelt sind. Man ist so in der Lage schnell die Daten zu finden die man sucht, obwohl sie auch in der untersten Ebene liegen können.

Wie das genau dann im Frontend ausschaut, sowie weitere Details zur Implementierung werden im Kapitel 5.2 beschrieben.

5 Implementierung

Die im vorherigen Kapitel erstellte Architektur wird im nächsten Schritt implementiert. Da auch heir die beiden Teile, Frontend und Backend getrennt voneinander betrachtet werden, wird im ersten Teil der Blick auf das Backend und im zweiten Teil auf das Frontend gelegt. Jeweils am Ende der beiden Abschnitte befinden sich Unterkapitel zur Installation, Handhabung und aufgetretenen Problemen. In einem dritten Abschnitt werden die Anforderungen aus dem vorherigen Kapitel noch mal aufgegriffen und überprüft, in wie weit Sie erfüllt werden konnten.

5.1 SensorConfigurator

Der Abschnitt zum SensorConfigurator gliedert sich in vier Teile. Zunächst wird die Ausarbeitung des Controllers und die Implementierung der Schnittstelle betrachtet. Im folgenden Abschnitt wird eine Installationsanleitung und Hilfestellungen zur Handhabung gegeben. Schließlich wird auf Probleme und Grenzen hingewiesen, die sich während der Implementierung ergaben.

5.1.1 Controller des SensorConfigurators

Wie bereits im Kapitel 4 beschrieben, arbeitet der SensorConfigurator im Wesentlichen durch die Verwaltung verschiedener XML-Dateien.

Um XML-Dateien in Java zu bearbeiten gibt es verschiedene Ansätze. Eine sehr komfortable Variante ist es, die Dateien in Objekte umzuwandeln. Das Framework JAXB wurde für diesen Zweck entwickelt und ist seit JAVA6[1] teil der Standard-API. Durch das mitgelieferte Programm “xjc” kann man sich aus einem bestehenden XML-Schema einen Klassenbaum generieren lassen. Durch “marshalling” und “unmarshalling” kann man die XML-Daten in die Klassen einlesen und zurück schreiben.

Das Problem an dieser Stelle sind verschiedene Feinheiten der XML-Dateien im GT4-System. So erwarten die Dateien des RP-Providers die Angabe des Typs durch “xsi:type” Attribute in jedem XML-Element. Diese müssen selbstverständlich beim Speichern der Objekte mit geschrieben werden. Außerdem ist in einem erhöhten Maße auf die syntaktische und semantische Korrektheit der XML-Dateien zu achten, da sonst der Marshalling-Prozess sofort einen Fehler meldet. Da die Fehlermeldungen des JAXB nicht immer zielführend sind, musste an dieser Stelle viel zusätzliche Arbeitszeit investiert werden.

sensor-data.xml Eine besondere Rolle spielte auch hier wieder die sensor-data.xml. Wie bereits im Kapitel 4.1.1 beschrieben, ist der Aufbau dieser Datei nicht beliebig zu gestalten. Die hierarchische Struktur, die durch die Elemente und Attribute festgelegt wird, bildet die Struktur des daraus entstehenden Resource-Property. Um also eine sensor-data.xml zu erstellen schaut man sich zu erst an, wie die zu erwartende Resource-Property-Datei aussehen muss, damit sie vom RP-Provider und dem restlichen GT4-System richtig verstanden wird. Anhand dieses Aufbaus kann man die sensor-data.xml konzipieren und mit den Werten für den Benutzer füllen. Wenn man sich an diese Vorgaben gehalten hat, kann der XSLT-Prozessor aus der sensor-data.xml die Resource-Property-Datei sensor-rp.xml erstellen. Die angegebenen Namen der Elemente und Attribute werden dabei zu XML-Elementen. Als Inhalt wird jeweils der $value$-Eintrag genommen. Ist kein $value$ angegeben, wird statt dessen der Wert aus dem $default$-Element benutzt. Eine Ausnahme stellt dabei ein $final$-Eintrag dar. In diesem Fall wird in jedem Fall der Wert des $final$-Elements genommen. Beispielhaft kann man in

[1] Um das Programm mit JAVA5 laufen zu lassen, sind die externen JAXB-Biblioteken zu laden. Die Änderungen am Quellcode sind minimal

```
<provider-data
      targetNamespace="http://foo.bar"
      sensorname="wartungs-sensor" ...>
    <element type="text" name="Site">
       <attribute type="text" name="UniqueID">
            <default>mabtest.lrz-muenchen.de</default>
            <value>mabtest.lrz-muenchen.de</value>
        </attribute>
        <element type="text" name="Name">
            <final>MDS Testsite FG2.6</final>
        </element>
        <element type="text" name="email">
           <choice>
                    <option>mailme@you.de</option>
                    <option>mailyou@me.com</option>
           </choice>
       </element>
    </element>
</provider-data>
```

Abbildung 5.1: Beispiel einer sensor-data.xml

Abbildung 5.2 sehen, wie eine solche Umwandlung aussieht. Ein Beispiel für eine sensor-data.xml kann in der Abbildung 5.1 eingesehen werden.

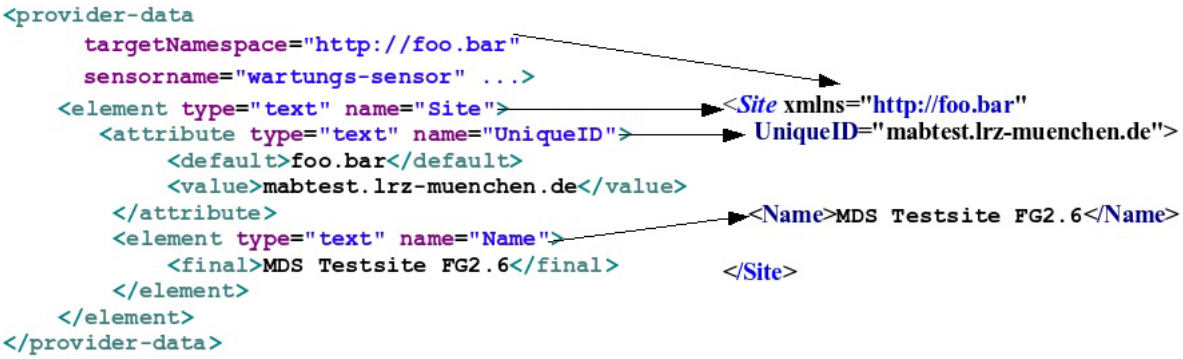

Abbildung 5.2: XSL-Transformation von sensor-data.xml zu sensor-rp.xml

sensor-rp-config.xml Die Konfiguration des RP-Providers sieht vor, dass vordefinierte Resource-Properties eingelesen werden können. Es gibt dabei verschiedene Möglichkeiten, wie die XML-Daten gelesen werden. Eine sehr praktische Möglichkeit ist das Starten eines externen Programms. Dabei wird die Standard-Ausgabe dieses Programms gelesen und als Input für den RP-Provider benutzt. In Abbildung 5.3 sieht man ein Beispiel für solch eine Konfiguration.

Diese Methode bietet interessante Vorteile: Statt des reinen einlesens durch das Programm `cat` kann man beispielsweise einen eigenen XSLT-Prozess starten, der neben den statischen Daten dynamische Daten einarbeitet. Eine genauere Beschreibung der Einstellungsmöglichkeiten des RP-Providers erhält man in [COMM 07].

rp-provider-config.xml Die rp-provider-config.xml stellt lediglich eine Zusammenfassung mehrerer sensor-rp-config-Dateien dar. Sie werden unter dem XML-Element “ResourcePropertyProviderConfigArray”

```
<ResourcePropertyProviderConfigArray ...>
        <configArray xsi:type="resourcePropertyProviderConfig">
                <resourcePropertyName xsi:type="xs:QName" ...>
                        glue:Site
                </resourcePropertyName>
                <resourcePropertyImpl xsi:type="xs:string">
        org.globus.mds.usefulrp.rpprovider.SingleValueResourcePropertyProvider
                </resourcePropertyImpl>
                <resourcePropertyElementProducerser ...>
                        <className xsi:type="xs:string">
        org.globus.mds.usefulrp.rpprovider.producers.ExternalProcessElementProducer
                        </className>
                        <arguments xsi:type="xs:string">
                                /bin/cat .../sensor-rp.xml
                        </arguments>
                        <period xsi:type="xs:int">
                                300
                        </period>
                </resourcePropertyElementProducers>
        </configArray>
</ResourcePropertyProviderConfigArray>
```

Abbildung 5.3: Beispiel für eine sensor-rp-config.xml

zusammengefasst. Der Administrator muss auf diese Datei nicht zugreifen. Änderungen nimmt lediglich der SensorConfigurator selbst vor.

5.1.2 GT4 WebService

Bei der Implementierung des WebServices war die Herausforderung, die angestrebte Architektur in ein möglichst einfaches WSDL zu verpacken. Außerdem musste entschieden werden, wie mit den neuen XML-Typen umgegangen wird. Auf der einen Seite ist es sinnvoll, die XML-Daten aus der sensor-data.xml als Objekte zu übermitteln, auf der anderen Seite macht es eine mögliche neue Client-Implementierung komplizierter. Die Alternative war, die XML-Datei als Text zu übermitteln. Einem Client wurde es somit selbst überlassen, ob die Daten in Objekte geparst werden oder die Daten direkt als Text weiter verarbeitet werden. Außerdem war es durch diese Variante möglich, das inzwischen gut funktionierende JAXB-Framework auf der Serverseite weiter zu verwenden.

Sensor-Finder Der Sensor-Finder beinhaltet nach der vorgestellten Architektur nur die Methode `availableSensors()`. Diese Methode sucht auf der Festplatte nach möglichen Sensor Konfigurationen. Dies geschieht durch sequentielles Lesen der Unterverzeichnisse eines konfigurierten Sensor-Verzeichnisses. Dabei werden die Dateien auf ihre syntaktische Korrektheit überprüft. Jeder gefundene Sensor wird sogleich an die ResourceHome weiter geleitetet. Die ResourceHome hält die Instanzen der gefundenen Sensoren in einer Hash-Tabelle. An den Finder übergibt sie jeweils den Key, unter dem der Sensor aufzufinden ist. Der Finder baut aus dem Hash-Schlüssel und der Adresse des SensorServices für jeden Sensor eine Endpoint-Reference. Baut ein Client mit solch einer Endpoint-Reference eine Verbindung zum SensorService auf, erhält er Zugriff auf den ausgewählten Sensor.

Sensor-Service Mit Hilfe des Sensor-Service kann ein Client weitere Informationen (Name, Ort, Aktivität und Inhalt) über einen ausgewählten Sensor erhalten und bestimmte Werte (Inhalt und Aktivitätsstatus) verändern. Die Daten der Sensoren liegen dabei selbst als Resource-Properties vor. Dies ist sinnvoll, damit auch andere Client-Applikationen durch die standardisierten Getter-Methoden auf den Namen, den Ort und den Inhalt des Sensors zugreifen können. Die standardisierten Setter-Methoden der Resource-Properties sind jedoch mit Ausnahme über den Inhalt nicht verfügbar. Somit ist es möglich mit Hilfe des WebServices den Inhalt zu verändern. Name und Speicherort des Sensors sind dagegen konstant.

Ein zusätzliches Resource-Property für den Aktivitäts-Zustand des Sensors ermöglicht auch hier den standartisierten Zugriff. Es handelt sich dabei um einen Boolean-Wert, der ebenfalls nicht nur gelesen, sondern auch gesetzt werden kann. Änderungen am Aktivitäts-Zustand und dem Inhalt eines Sensors wirken sich direkt auf das zugrundeliegende Gesamtsystem aus. In der Praxis bedeutet dies, dass der SensorConfigurator beim Aufruf einer Set-Methode die entsprechenden Speicher-Operationen auf dem System vornimmt. Wurde der Inhalt angepasst, wird die veränderte XML-Datei gespeichert und der XSLT-Prozess angestoßen um die neue sensor-rp.xml zu generieren. Beim Ändern des Aktivitäts-Status wird die rp-provider-config.xml Datei überprüft und gegebenenfalls Einträge ergänzt bzw. gelöscht.

5.1.3 Installation und Handhabung

Installation

Vorraussetzung für eine Installation des SensorConfigurators ist eine möglichst aktuelle Version des Globus Toolkits 4. Ab Version 4.1 wird der RP-Provider mit ausgeliefert. Möchte man eine ältere Version benutzen muss man den RP-Provider von Hand nachinstallieren.

Da der SensorConfigurator in Java 6 programmiert wurde, ist es notwendig, den GT4-Server mit dieser Java-Version laufen zu lassen. Für die Installation mit Java 6 steht auf der Globus-Homepage [COMM 07] ein Patch zur Verfügung. Möchte man von einem Java-Versionswechsel absehen, muss das JAXB-Framework von Hand nachinstalliert werden. Außerdem sind vor dem Compilieren des SensorConfigurators in eine ältere Version kleine Änderungen am Quellcode notwendig (z.B. wurde die Methode `String.isEmpty()` erst in Java 6 eingeführt). Ausserdem muss das Compiler-Hilfsprogramm `ant` von Apache in einer aktuellen Form vorliegen (getestet mit ant version 1.7.0).

Um den SensorConfigurator zu installieren, muss man das Paket `deploy_provider.tar.gz` mit Hilfe von `tar -xzf deploy_provider.tar.gz` entpacken. Durch das Ausführen des Compilers mittels `ant` werden die WSDL-Dateien angepasst, die nötigen Stubs und die Services compiliert. Als Ergebniss erhält man ein GAR-File. Dieses muss man mit Root-Rechten installieren: `ant deploy`. Dabei ist darauf zu achten, dass in der Umgebungsvariable `$GLOBUS_LOCATION` der Pfad zur Globus-Installation gesetzt wurde. Im Auslieferungsprozess wird auserdem das XSLT-Skript zum transformieren von sensor-data.xml zu sensor-rp.xml und eine Einstellungsdatei mit Standardwerten erstellt. Beide Dateien werden in `$GLOBUS_LOCATION/etc/de_lrz_sensor/` abgelegt.

Während das XSLT-Skript im Normalfall nicht angepasst werden muss, sollte man auf jeden Fall einen Blick auf die Einstellungsdatei `provider.properties` werfen. Diese Datei enhält drei Einstellungsparameter:

provider.sensorRootDir Hier wird der Ordner angegeben, in dem der SensorConfigurator die erstellten Sensoren suchen soll.

provider.rp-provider-config Diese Einstellung legt fest, in welcher Datei die Einstellungen für den RP-Provider zu finden sind. In der Standardinstallation sollte der eingestellte Wert auf die richtige Datei verweisen.

provider.make-provider-xslt Die dritte Einstellung verweist auf das XSLT-Skript, welches bei der Installation kopiert wurde. Soll ein anderes Skript verwendet werden, kann man dieses hier eintragen.

Nach einer fehlerfreien Installation muss der Globus-Container neu gestartet werden. Dazu steht das Start-Stop-Skript `globus-start-stop` in der Globus-Installation zur Verfügung. In der Log-Datei sollten nun die neuen Services `sensorprovider/SensorService` und `sensorprovider/FinderService` auftauchen. Dabei ist jedoch zu beachten, dass es im Rahmen des Praktikums nicht möglich war, ein Sicherheits-Konzept für den Service zu implementieren. Der Service ist daher bislang nur ohne die Securityoptions des Globus-Containers getestet (Option `-nosec` muss gesetzt sein!). Sie hierzu auch Kapitel 5.1.4.

Erstellen eines Sensors

Wurde der SensorConfigurator korrekt installiert kann er nun mit Daten gefüttert werden. Dazu erstellt man im Sensor-Ordner (wie im letzten Abschnitt konfiguriert) einen neuen Unterordner. Der Name des Ordners entspricht idealerweise dem Namen des Sensors. In diesem Ordner erzeugen wir die beiden XML-Dateien, die bereits eingehend im Abschnitt 5.1.1. beschrieben wurden. Sollte der neue Sensor nicht automatisch angezeigt werden, kann man ein erneutes Einlesen durch Neustart des Containers erzwingen. Anschließend kann der Sensor über das WebFrontend beliebig konfiguriert und aktiviert werden.

5.1.4 Bekannte Probleme und Grenzen

An dieser Stelle soll kurz auf Probleme bzw. Grenzen der Implementierung eingegangen werden.

Prefix xsd not bound

Diese Fehlermeldung trat bereits bei den ersten Versuchen in Zusammenhang mit der rp-provider-config.xml und dem JAXB-Framework auf. Es stellte sich heraus, dass in der vorliegenden Version der rp-provider-config.xml das Namespaceprefix “xsd” nicht gebunden ist. Verändert man die Konfigurationsdatei indem man im root-Element den Namensraum definiert, ist das Problem gelöst.

Sicherheits-Konzept

Das Globus-Toolkit enthält ein sehr umfangreiches und ausgeklügeltes Sicherheitskonzept. Dieses wird vor allem durch das Authentifizieren mittels Zertifikaten realisiert. Die Implementierung der Sicherheitsmechanismen hätte jedoch den Rahmen dieses Praktikums gesprengt.

Es ist sehr gut möglich in einer weiterführenden Arbeit ein geeignetes Konzept für die beiden Services zu erstellen und dieses nachträglich in die bestehende Implementierung zu integrieren. Um das Projekt für den tatsächlichen Gebrauch im D-GRID nutzbar zu machen, wäre dieser Schritt unbedingt zu empfehlen.

Auf Grund der fehlenden Zertifikate war es im Rahmen des Praktikums auch nicht möglich zu testen, in wie weit sich die Services im bestehenden Sicherheitskonzept des Containers eingliedern. Somit wurde das Programm bislang nur ohne die Security-Optionen des Globus-Containers getestet. Dies geschieht durch das setzen der Option `-nosec` im Startskript.

Aktivstatus wird nicht aktualisiert

Es stellte sich heraus, dass der vorliegende RP-Provider die Konfigurationsdatei `rp-provider-config.xml` nur einmal während des Startens einließt. Damit sind die änderungen, die durch den Konfigurator getätigt werden im laufenden Betrieb nicht sichtbar. Eine einfache Lösung den RP-Provider zu zwingen die Datei erneut einzulesen wurde bisher nicht gefunden. Bislang gab auch die Nachfrage auf der Developer-Mailinglist des GT4 kein Ergebnis. Es besteht daher nach wie vor Bedarf eine geeignete Lösung zu finden.

5.2 Webfrontend

Wie schon im Kapitel 4.2 erläutert ist, ist unser Frontend ein Servlet. In diesen Abschnitt wird die Implementierung des Servlets beschrieben. Zuerst wird erläutert wie die Model-View-Controller (MVC) Architektur des Servlets implementiert ist. Dannach wird im Abschnitt 5.2.2 genauer auf den WebserviceClient eingegangen und erklärt wie die Kommunikation zwischen den Webservice und WebserviceClient funktioniert. Im Abschnitt 5.2.3 wird die Umsetzung der Baumstruktur beschrieben. Danach im vorletzten Abschnitt 5.2.4 wird

die Benutzungsanleitung näher erläutert. Im letzten Abschnitt 5.2.4 befindet sich die Installationsanweisungen für das Frontend.

Um die Model-View-Controller Architektur des Servlets auch zu verwirklichen wird das Apache Struts Framework eingesetzt. Das Framework wird im nächsten Abschnitt 5.2.1 erklärt.

5.2.1 Apache Struts

In diesen Abschnitt wird das Apache Struts Framework erläutert, nach dem das Frontend implementiert ist. Struts ist ein Open-Source-Framework für die Präsentations- und Steuerungsschicht von Java-Webanwendungen. Struts beschleunigt die Entwicklung von Webanwendungen wesentlich, indem es HTTP-Anfragen in einem standardisierten Prozess verarbeitet. Dabei bedient es sich standardisierter Technologien wie JavaServlets, Java Beans, Ressource Bundles und XML sowie verschiedener Jakarta-Commons-Pakete [Apac 08]. Es bietet viele Funktionen die das Darstellen sowie Verändern der dynamischen Daten im Servlet erleichtert. Da Struts auf dem Model-View-Controller Prinzip (Model 2) basiert, bietet es eine schöne Trennung der Darstellung, Daten und Steuerung. Dies erleichtert das Programmieren der Internetseiten. Die zentralen Features für Struts sind:

- Automatische Überprüfung von Benutzereingaben
- Fehlerbehandlung
- Internationalisierung
- Verhindern eines “mehrfachem Abschickens“ einer Seite (Tokens)

Die praktische Umsetzung der Struts im Servlet wird im folgendem beschrieben. Ziel der des Struts Framenworks ist die Trennung von Präsentation, Datenhaltung und Anwendungslogik.Die MVC Architektur erhöht die Übersicht und die Wartbarkeit. Deshalb definiert man in Struts drei Hauptkomponenten:

- JSP: Präsentation. Wird zur Laufzeit in ein Servlet (Java-Code) umgewandelt und kompiliert.
- Action: Controller. Schnittstelle zwischen View und Anwendungslogik (Holen und Aktualisieren der Daten).
- FormBean: Datenhaltung und Validierung (Inhalte von HTML-Formularfeldern, Kollektionen für Listen usw. in Java Beans).

Alle drei Komponenten werden in der zentralen Konfigurationsdatei von Struts (`struts-config.xml`) miteinander verknüpft und können somit miteinander kommunizieren. Für unser Frontend sieht die `struts-config.xml` wie in Listing 5.1 aus. Als erstes stehen die die Namen der FormBeans verknüpft mit der entsprechenden Java Klasse. Da sind auch die sogenannten Globalen forwards beschrieben, die in dieser Anwendung die Startseite (`index.jsp`) ist. Globale Forwards repräsentieren die Seiten, an die man aus jeder JSP oder Action weitergeleitet werden kann. In den action-mappings werden die Action Klassen mit den entsprechenden FormBeans verknüpft. Dort wird auch ein Name den Actions zugewiesen (path), über den werden die Actions aus den JSP Seiten angesteuert. Die `forward` zeigen an wohin alles der User, von dieser Seite aus kommen kann, abhängig von seinen Angaben.

Listing 5.1: struts-config.xml - Zentrale Konfigurationsdate der Struts

```
<struts-config>
<!-- FormBean Definitionen -->
<form-beans>
 <form-bean name="SensorInfoCollectionForm"
            type="de.lrz.sensorprovider.gui.SensorInfoCollectionForm"/>
 <form-bean name="SensorInfoTreeForm"
            type="de.lrz.sensorprovider.gui.SensorInfoTreeForm"/>
</form-beans>
<!-- Forwards, hier zur startseite -->
<global-forwards>
 <forward name="start" path="/index.jsp"/>
</global-forwards>
```

```
 <!-- Action Definitionen -->
 <action-mappings>
  <action name="SensorInfoCollectionForm"
          path="/getSensors"
          scope="session"
          type="de.lrz.sensorprovider.gui.GetSensorInfoAction">
   <forward name="showSensorInfos" path="/pages/showSensorInfos.jsp"/>
   <forward name="emptyUrl" path="/pages/emptyUrl.jsp"/>
   <forward name="errorActivation" path="/pages/errorActivation.jsp"/>
  </action>


  <action input="/pages/showSensorInfos.jsp"
          name="SensorInfoTreeForm"
          parameter="do"
          path="/sensorEdit"
          scope="session"
          type="de.lrz.sensorprovider.gui.EditSensorInfoAction">
   <forward name="editSensorInfoAction"
            path="/pages/editSensorInfo.jsp"/>
   <forward name="showSensorInfos"
            path="/pages/showSensorInfos.jsp" redirect="true"/>
   <forward name="errorSavingSensorData"
            path="/pages/errorSavingSensorData.jsp"/>
  </action>
  </action-mappings>
</struts-config>
```

JSP View

Bei der Entwicklung ist eins der Ziele in der JSP so wenig Java-Code (JavaScript) wie möglich zu verwenden. Um dies zu ermöglichen, sollte man die von Struts mitgelieferten Tag-Libs einsetzen, die einem beim Auslesen der Form-Bean helfen. Geschrieben und gelesen wird hier jeweils nur aus der im Request liegenden Form-Bean. Im Listing 5.2 ist ein Teil der `index.jsp` aufgelistet. Auf dieser Seite wird der Benutzter aufgefordert Host und Port des Rechners anzugeben wo der Webservice aus Abschnitt 5.2.4 läuft. In der `index.jsp` wird zuerstes die Tag-Lib von Struts eingebunden. Mit Hilfe diser Taglib ist es möglich eine Form mit Hilfe der Action `/getSensors.do` zu benutzten. Wenn der Benutzer `submit` tätigt wird die Action `/getSensors.do` ausgeführt und in der Form `SensorInfoCollectionForm` werden die Werte `hostname` und `port` gesetzt. Damit die JSP die richtige Form-Bean findet wird durch `struts-config.xml` aus Listing 5.1 sichergestellt.

Listing 5.2: index.jsp

```
<%@ taglib uri="/WEB-INF/struts-html" prefix="html" %>
......
    <html:form action="/getSensors.do" >
       <td>Hostname:</td>
       <td>
           <html:text size="90" property="hostname"
                      value="mabtest.lrz-muenchen.de"/>
       </td>
    <tr>
       <td>Port</td>
       <td>
          <html:text size="10" property="port"
                     value="9009"></html:text>
       </td>
    </tr>
    <tr>
```

```
        <td>
            <html:submit value="Get all Sensors"></html:submit>
        </td>
    </tr>
    </html:form>
.....
```

Form Beans

Die Form-Bean ist eine normale Java Bean, die alle benötigten Daten für die JSP und die Action hält. Die Form-Bean fungiert dabei als Schnittstelle zwischen diesen beiden Komponenten und wird über die `struts-config.xml` mit einem Formular in der JSP verknüpft. Wenn das Formular ausgefüllt und abgeschickt wird, wird die Bean durch das ActionServlet (noch bevor die Action ausgeführt wird) über die setter-Methoden mit den entsprechenden Eingabewerten gefüllt.

In der `index.jsp` aus Listing 5.2 werden in der Form-Bean `SensorInfoCollectionForm` die Werte `hostname` und `port` gesetzt. Um dies zu ermöglichen müssen in dieser Form-Bean die Methoden `setHostname` und `setPort` implementiert sein. Die Action Klasse wird die Werte auch auslesen und deswegen müssen die entsprechenden "getter" `getHostname` und `getPort` in der Form-Bean auch implementiert werden.

Die FormBean enthält außerdem eine Validierungs-Methode, welche (wenn in der Konfiguration aktiviert) die Daten der Form-Bean prüft, bevor sie zur Action geschickt werden. Außerdem enthält sie eine reset-Methode um den Inhalt zurückzusetzen, damit sie wiederverwendet werden kann.

Action

Action ist die die Controller Komponente im Struts Framwork. Sie kommuniziert mit dem WebService Client, der in Abschnitt 5.2.2 beschrieben ist. Über den die Sensordaten holt und sie auch wieder mit Hilfe dieser Klasse speichert. Sie enthält weitere Prüf- und Auswertungsmechanismen, wie z.B. validieren der Eingabe des Hostnames wo der WebService läuft.

Die Action bereitet die Sensordaten auf und speichert sie in der Form-Bean ab. Die Daten werden, nachdem die Action eine Forward auf die JSP abgeschickt hat, wieder von der JSP herausgelesen. Daher ist es uberflüssig, die Daten direkt in den Request zu speichern dazu ist die Form-Bean da.

Die Action ist ebenfalls für die Navigation durch die JSPs verantwortlich, da sie nach getaner Arbeit ein ActionForward zurückgeben muss. Dieser Forward hat einen bestimmten Namen, im Fall des Beispiels aus Listing 5.2 könnte ein ActionForward `showSensorInfos` lauten. Je nachdem, welchen Forward die Action zurückgibt, entscheidet das Struts-Framework anhand der `struts-config.xml`, zu welcher Seite weitergeleitet werden soll.

5.2.2 WebService Client

Um die Sensordaten im Frontend anzuzeigen und bearbeiten müssen sie zuerst mit Hilfe dieses WebService Clients geholt werden. In diesen Abschnitt wird der Aufbau der Verbindung vom WebService Client zum Webservice aus Kapitel 4.1.5 beschrieben.

Die Action Klassen aus dem Frontend stoßen die Klasse `Sensorcontroller.java`, die die Verbindung zum WebService verwaltet. Die Klassen übergeben dabei die genaue Addresse (Hostname:Port) wo der WebService läuft an die `Sensorcontroller.java`. Diese baut die Verbidung zum WebService auf, der in Kapitel 4.1.5 näher beschrieben ist. Da der Service GT-4 spezifisch ist, muss der Aufbau der Verbindung folgendermaßen ablaufen:

Zuerst wird ein `EndpointReferenceType` erstellt. Diesem wird die genaue Adresse des Webservice (`serviceURI`) zugewiesen.

Listing 5.3: SensorController.java

```
EndpointReferenceType endpoint = new EndpointReferenceType();
endpoint.setAddress(new Address(serviceURI));
```

Danach verläuft der Aufbau der Verbindung wie im Sequenzdiagramm aus Abbildung 5.4. Mit Hilfe von `FinderServiceAddressingLocator` und des erstellten `EndpointReferenceType` bekommt man den `FinderPortType`. Über diesen bekommt man alle Sensoren in Form von `EndpointReferenceTypes`. Um die eigentlichen Daten zu bekommen muss man über den `SensorServiceAddressingLocator` die einzelnen `SensorPortTypes` holen. Dort liegen die die einzelnen Informationen der Sensoren, die mit `getResourceProperty(QName)` erreichbar sind. Der QName kann dabei die Werte aus Tabelle 5.1 annehmen.

QName	Beschreibung
RP_NAME	der Name des Sensors
RP_LOCATION	der Ordnername
RP_ACTIVE	Status des Sensors (aktiv, inaktiv)
RP_SENSOR_DATA	Daten des Sensors

Tabelle 5.1: Mögliche QNames

Die Informationen aus der Tabelle 5.1 werden vom Controller, den Action Klassen aus dem Frontend, aufbereitet. In den dynamischen Webseiten (JSPs), die in Abschnitt 5.2.4 beschrieben sind, dem Benutzter zur Verfügung gestellt. Dieser kann bestimmte Änderungen vornehmen und sie anschließend abspeichern. Die Daten werden letzlich über den gleichen Webservice im Server gespeichert. Da aber muss der WebService Client die `SensorPortTypes` erst erstellen und dann zum Webservice schicken der sie speichert.

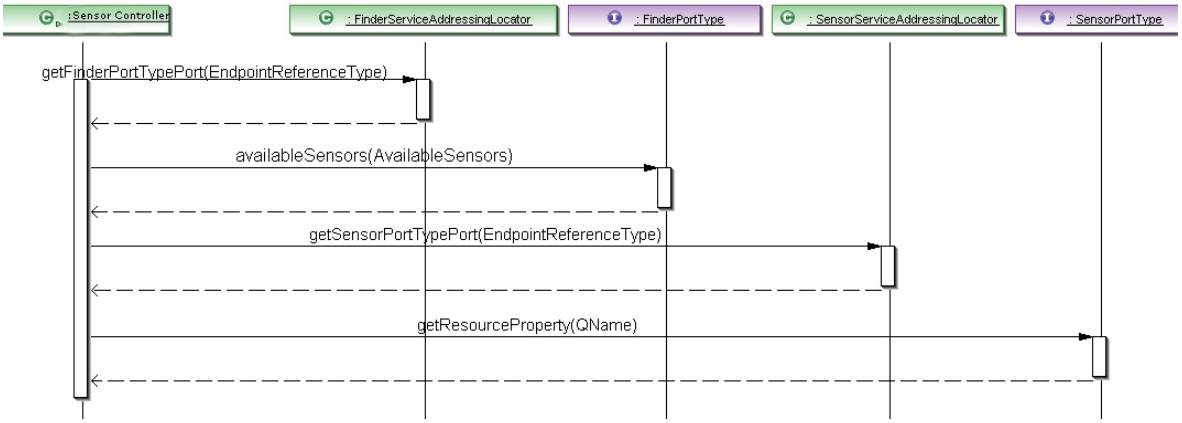

Abbildung 5.4: Webservice Abfrage

5.2.3 Baumstruktur mit Hilfe von Tree-Tags

Im letzten Abschnitt wurde beschrieben wie die Sensordaten zum Controller, den Action Klassen des Servlets, kommen. Die Sensordaten müssen auch entsprechend aufbereitet und im Frontend angezeigt werden, damit der User auch Änderungen vornehmen kann. Im diesen Abschnitt wird die Umsetzung der Aufbereitung der Daten in eine Baumstruktur, die im Kapitel 4.2.3 beschrieben ist.

Für die Darstellung in einer JSP werden die Sensordaten `RP_SENSOR_DATA` aus dem letzen Abschnitt 5.2.2 in eine Baumstruktur umgewandelt. Diese wird mittels der Open-Source erhältlichen so genannten “Tree-Tags” der Firma Jenkov Developement aus Dänemark umgesetzt. Die Tags ermöglichen eine vom Windows-Explorer her bekannte Darstellung von Hierarchien. Die “Tree-Tags” wurden speziell für JavaServerPages (JSP) entwickelt. Mit Hilfe dieser kann auch die Baumstruktur aus Abschnitt 4.2.3 in Java Server Pages visualiesiert werden.

Um Tree Tags zu nutzen, muss erst ein Treemodel aus den Sensordaten erstellt werden in dem die einzelnen Knoten und ihre Position in der Hierarchie des Baumes dargestellt werden. Ein Teil der Sensordaten kann wie im Listing 5.4 auschauen. Diese stellt einen Namen da, der sich aus einem Nachnamen und einen Vornamen zusammensetzt. Mittels der Tree-Tags kann in der JSP-Seite der entsprechende Knoten (Name, Vorname oder Nachname) aufgerufen werden. Das Modell wird in der Session gespeichert und steht dem Benutzer dynamisch zur Verfügung.

Listing 5.4: Beispiel: Sensor Daten

```
<element type="text" name="Name">

    <attribute type="text" name="Nachname">
        <value>Meier</value>
    </attribute>

    <attribute type="text" name="Vorname">
        <value>Hans</value>
    </attribute>

</element>
```

Um ein Tree-Model aus dem Listing 5.4 zu erstellen wird zuerst eine Tree-Instanz `ITree` mit dem Namen “Name“ erstellt. Im Listing 5.5 werden die Knoten `ITreeNode` ”Nachname” und “Vorname” erstellt. Den Knoten kann man Objekte zuweisen, die dann in der JSP mit dem Knoten verlinkt werden. Die Werte der zugewiesenen Objekte wird man verändern können und somit die Sensordaten ändern.

Listing 5.5:

```
ITree tree = new Tree();
ITreeNode rootNode = new TreeNode("ID1" , "Name" , "Element");

ITreeNode node1 = new TreeNode("ID11" , "Nachname" , "Attribute");
node1.setObject("Meier");

ITreeNode node2 = new TreeNode("ID12" , "Vorname" , "Attribute");
node1.setObject("Hans");
```

Dem Knoten wird neben der eindeutigen Knoten-ID und dem Knotennamen ein Knotentyp zugeordnet, welcher die Knoten in einer Hierarchieebene widerspiegelt. Im Listing 5.6 wird über die Methode `addChild` die Hierarchie im Baum erzeugt. Die Knoten `node1` und `node2` werden zum Knoten `rootNode` hinzugefügt und stehen dann eine Ebene unter ihm.

Listing 5.6:

```
rootNode.addChild(node1);
rootNode.addChild(node1);
```

Schlussendlich wird im Listing 5.7 die Wurzel des Baumes angegeben und der Baum der Session übergeben. Aus der die JSP auf den Baum zugreifen kann.

Listing 5.7:

```
tree.setRoot(rootNode);
session.setAttribute("tree", tree);
```

Ist das Modell angelegt kann der Baum über die in der Web-INF des Servlets befindliche treetag.tld-Datei (Tag Library Descriptor) in der JSP dargestellt werden. Die taglib wird furch den Eintrag im Listing 5.8 der JSP zugewiesen.

Listing 5.8:

```
    <%@ taglib uri="/WEB-INF/treetag.tld" prefix="tree" %>
```

Da auch mehrere Modelle in einer JSP enthalten sein können, müssen die Tags einem Modell zugeordnet werden. Dies geschieht durch den Eintrag im Listing 5.9

Listing 5.9:

```
<tree:tree tree="tree.model" node="tree.node">
</tree:tree>
```

Mittels `node=` werden den Tags die Knoten zugewiesen. Als Grundgerüst für die Darstellung des Baumes dienen mehrere verschachtelte HTML-Tabellen. In den Tabellen können dann abfragen gemacht werden ob Elemente Unterelemente haben und dementsprechend die Knoten auch darstellen. Wie die Baumstruktur genau in der JSP auschaut wird in nächsten Abschnitt 5.2.4 gezeigt Für Eine genauere Beschreibung zu den Jenkov Tree-Tags möchte ich auf die Webseite http://www.jenkov.dk [Deve 04] verweisen.

5.2.4 Installation und Handhabung

In diesen Abschnit wird zuerst auf die Handhabung des WebFrontends eingegangen, dannach wird die Installation genau beschrieben.

Handhabung

In diesen Abschnitt werden die einzelnen Webseiten vorgestellt und die zusammenhänge erklärt, wie man die Sensoren angezeigt bekommt oder was muss man tun um sie zu ändern.

Die Startseite 5.5 fordert den Benutzter auf die IP-Adresse oder den Rechnernamen sowie denn Port anzugeben unter dem der Webservice läuft. Als Vorgabe wird der Rechner `mabtest.lrz-muenchen.de` mit dem Port `9009` angegeben.

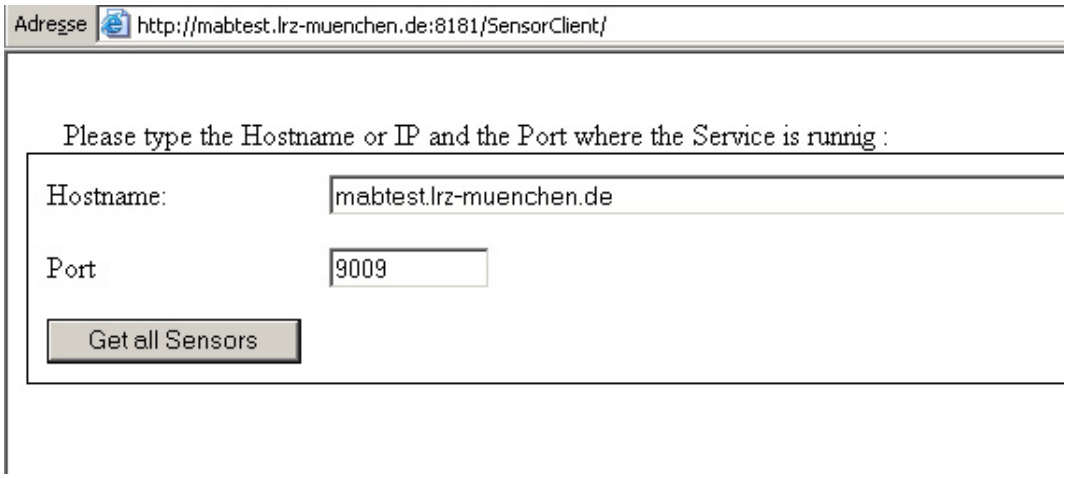

Abbildung 5.5: Startseite des Servelts

Wenn man auf den Button `Get all Sensors` drückt, wird die genaue URI zusammengebaut und zwar nach dem folgenden Prinzip:

Listing 5.10:

```
http://[Rechnername]:[Port]/wsrf/services/sensorprovider/FinderService
```

Mit dieser zusammengebauten URI wird der Webservice Client aus Abschnitt 5.2.2 angestoßen. Dieser versucht alle Sensoren die in dem angesprochenen WebService vorhanden sind zu finden.

Falls bei dem Versuch die Sensoren zu finden ein Fehler auftritt, wird der User auf die Startseite verwiesen, erweitert durch eine bestimmte Fehlermeldung, siehe Abbildung 5.6. Als Fehlermeldungen können auftreten:

- Rechnernamen nicht angeben,
- Port nicht angeben,
- Rechner nicht erreichbar
- Auf diesem Rechner unter diesen Port läuft gar kein Webservice
- Webservice nicht ansprechbar
- Andere Fehler

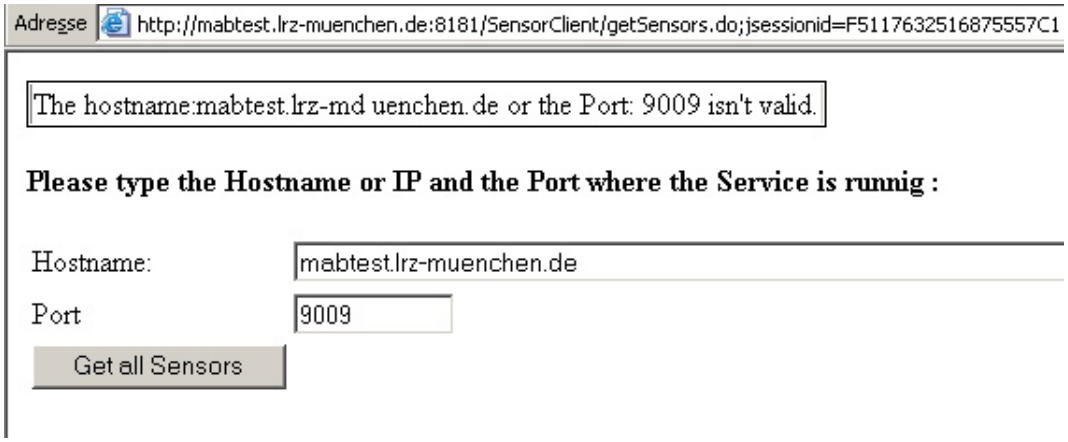

Abbildung 5.6: Beim verbinden zum Webservice ist ein Fehler aufgetreten

Wenn sich der WebServiceClient erfolgreich die Sensordaten über das Webservice holt, werden alle Sensoren auf der Seite wie in Abbildung 5.7 aufgelistet. Der Status jedes Sensors wird in der ersten Spalte angezeigt. Wenn der Status geändert wird, überträgt der WebServiceClient (Kapitel 5.2.2) die Änderung an den Server (vgl. Kapitel 4.1.4).

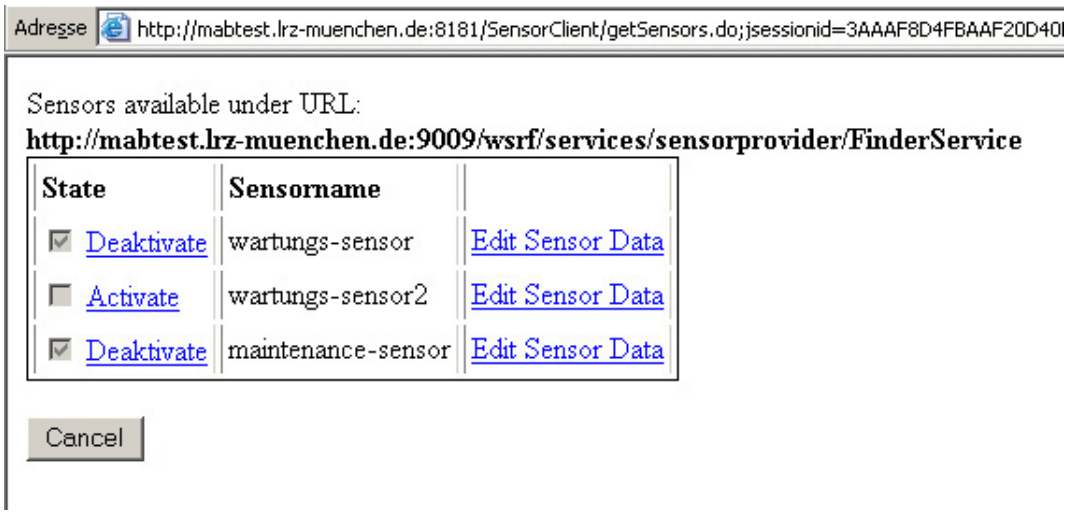

Abbildung 5.7: Alle verfügbaren Sensoren

Falls der Benutzer einen Sensor editieren will drückt er den Link `Edit sensor Data` im Bild 5.7 und wird dann auf die nächste Seite (Abbildung 5.8) verwiesen. Zuerst erscheint nur der Name des Sensors mit einen Ordner Symbol davor. Den Ordner kann man aufklappen. Dann schaut die Seite wie in der Abbildung 5.9 aus.

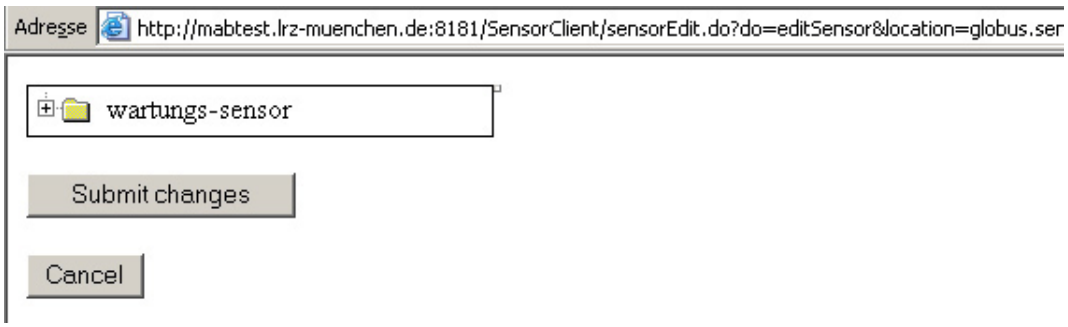

Abbildung 5.8: Wartungssensor zugeklappt

In der Abbildung 5.9 wird die `sensor-data.xml` (siehe Abschnitt 4.1.1) als eine Baumstruktur dargestellt. Diese Darstellung ist mit dem Datei-/Ordner-Browser in gängigen Betriebsystemen zu vergleichen. Eine genaue Beschreibung wie die Datei `sensor-data.xml` in die Baumstruktur umgewandelt wird, ist im Abschnitt 5.2.3 beschrieben.

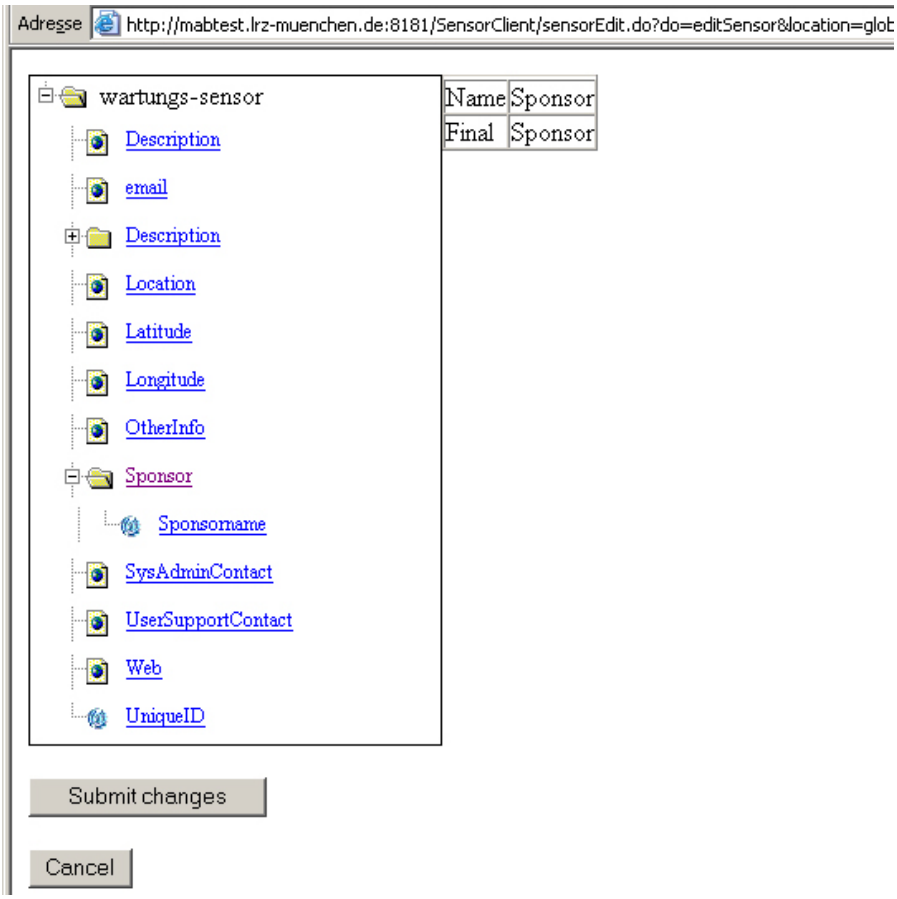

Abbildung 5.9: Wartungssensor als Baumstruktur

Diese Darstellung wurde gewählt, damit sich alle Benutzer dieses Webfrontends schnell zurechtfinden. In der Baumstruktur ist das Wurzel-Element der Name des Sensors, der beliebig viele Unterelemente besitzen darf. Ein Unterelement kann ein Attribut oder Element sein. Ist es ein Attribut, wird es wie in Abbildung 5.9 `UniqueID` dargestellt. Attribute können nur einen Bestimmten Wert haben wie in Kapitel 4.1.1 beschrieben. Unterelemente dagegen können wieder ein Element sein. Hat es jedoch keine Unterelemente, ist es von der Funktionalität ähnlich der eines Attributs, siehe Abbildung 5.9 `Location`. Wenn ein Element weitere Unte-

relemente oder Attribute hat, wird es mit dem Ordner-Icon markiert und hat selbst keinen Wert, siehe in der Abbildung `Sponsor`. Alle Elemente die mit einem Ordner markiert sind kann man auf und zu klappen. Wenn der Benutzter auf ein Element oder Attribut mit der Maus klickt, erscheint rechts vom Baum ein Rahmen mit den bestimmten Werten des Attributes bzw. des Elementes. In der Abbildung 5.9 wurde auf `Sponsor` (rot markiert) gecklickt. Im Bereich rechts wird der Name und Wert des angeklickten Elementes angezeigt. Da dieser Wert wieder Unterlelemente hat er nur einen "Finalen" Wert den man nicht ändern kann. Da nach der Definition von der Sensordaten beschrieben in Kapitel 4.1.1 kann ein Element nur Finale Werte haben.

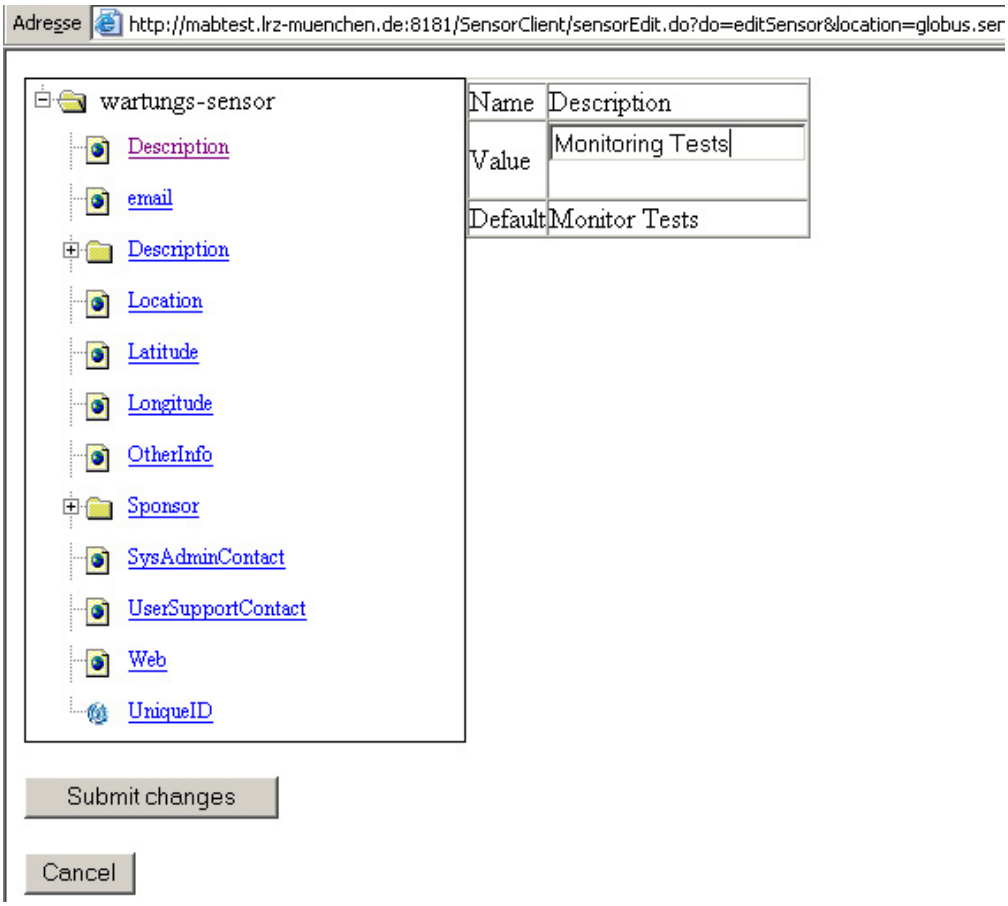

Abbildung 5.10: Wartungssensor Daten ändern

In der Abbildung 5.10 wurde auf `Description` (rot markiert) geklickt. Im Bereich rechts davon sieht man den Typ und den Namen des Elementes. In diesem Beispiel ist der Wert von `Description` ein einfacher Text, den man verändern kann. Dieser Text kann beliebig lang sein.

Wenn ein Attribut oder Element ohne Unterelemente nur ein bestimmte Anzahl von vordefinierten Werten annehmen kann, wird dies durch einen `Choice Type` umgesetzt. In der Abbildung 5.11 ist das Element Email ein Choice Type bei dem man den Wert mit Hilfe von einer Auswahlliste ändern kann.

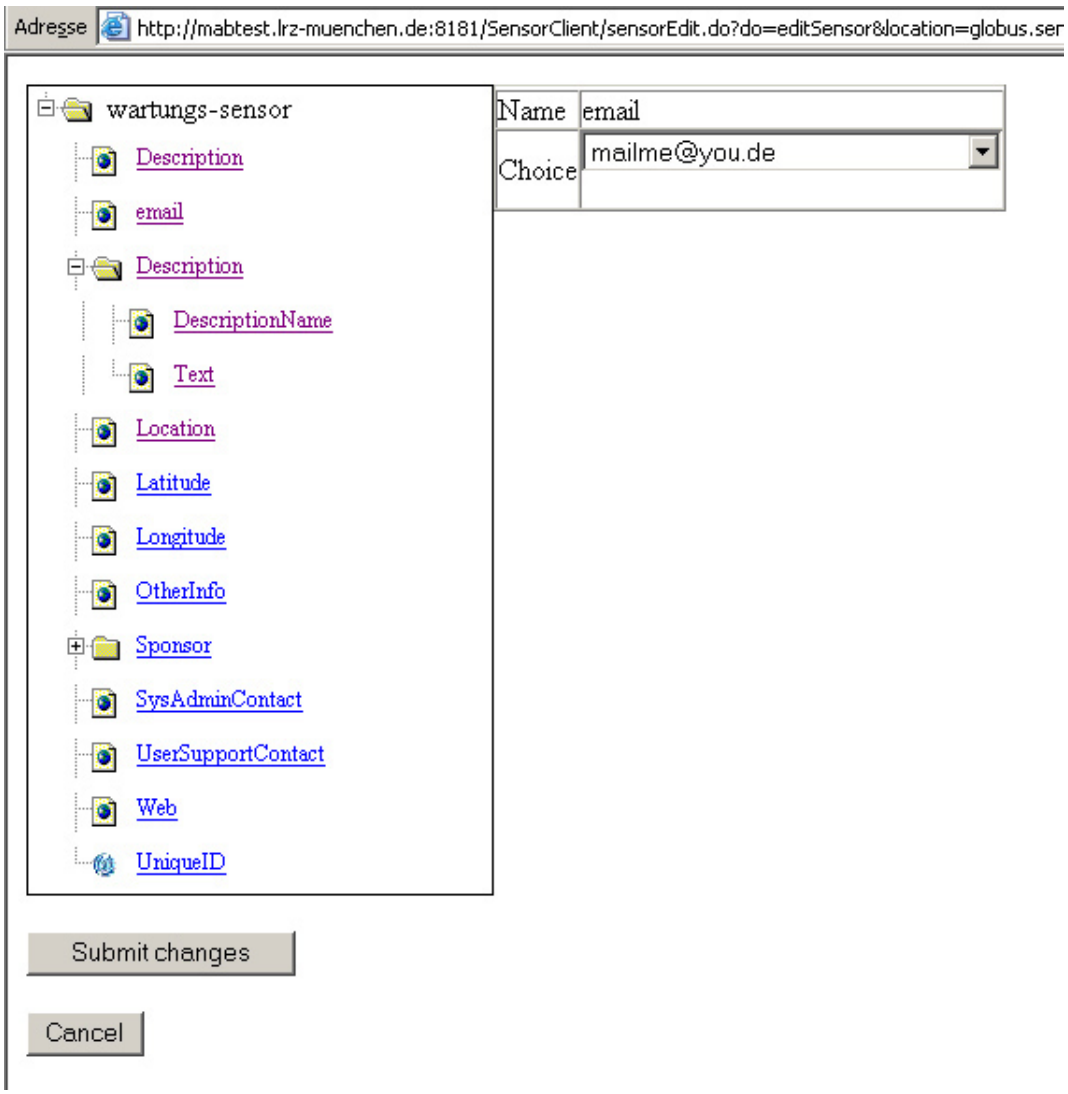

Abbildung 5.11: Wartungssensor Choice Type Beispiel

Installation

Das Servlet wird als ein War-File (SensorClient.war) ausgleiefert, das beudeutet es muss in einen Tomcat (Version 6) deployed werden. Dazu muss die Datei SensorClient.war in das ”webapps“ Verzeichnis vom Tomcat kopiert werden, oder mit Hilfe der Tomcat-Weboberfläche hoh geldane werden. Wenn Tomcat bereits läuft entpackt er die ”.war“ Datei automatisch. Die Anwendung wird auf den Webservice vom Globus-Server zugreifen. Der Client braucht daher eine Konfigurationsdatei mit dem Namen `client-config.wsdd`. Die Datei befindet sich im Verzeichnis `$CATALINA_HOME/webapps/SensorClient/conf`. Sie muss beim starten vom Tomcat mit folgender Java -D Option geladen werden:

Listing 5.11:

```
-Daxis.ClientConfigFile="Pfad_zur_Datei"
```

Dies kann im Start-Skript vom Tomcat, `catalina.sh` eingefügt werden. Dazu muss die Zeile für die JAVA-Optionen (Standartkonfiguration Zeile 90) editiert werden.

Listing 5.12:

```
JAVA_OPTS=-Daxis.ClientConfigFile="$CATALINA_HOME"
          /webapps/SensorClient/conf/client-config.wsdd
```

Anschließend braucht man nur den Tomcat neu zu starten, damit der Pfad zur client-config.wsdd dem SensorClient bekannt ist. Die Webanwendung sollte, nach dem Neustart, nun laufen und ist unter `Rechnername:Port/SensorClient` erreichbar. Wobei Rechnername und Port des Rechners wo der Tomcat läuft zu ersetzen ist. Wenn man die Seite aufruft wird die Startseite des Servlets angezeigt, wie die Abbildung 5.5.

Log Eigenschaften Als Log-System wird das Apache Log4j [Apa 07] benutzt. Die Konfigurationsdatei `log4j.properties` für das Log4j liegt im "$CATALIA_HOME/weppaps/SensorClient/WEB-INF/classes" Verzeichnis. Dort können u.a. Einstellungen bezüglich des Loglevels und der Logdateien gesetzt werden. Bei Änderungen der Einstellung ist ein reload des Servlets oder ein Neustart des Tomcats nötig damit die Änderung auch angewendet werden. Die log4j.properties für den SensorClient schaut folgendermaßen aus:

Listing 5.13:

```
log4j.rootLogger=DEBUG, R
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=${catalina.home}/logs/sensorClient.log
log4j.appender.R.MaxFileSize=10MB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%
```

5.3 Evaluation

Zum Abschluss dieses Kapitels soll überprüft werden, ob die gestellten Anforderungen erfüllt wurden. Dazu sind die Anforderungen in der Tabelle 5.2 nochmal übersichtlich aufgelistet. In der dritten Spalte ist vermerkt, in wie weit die Anforderung erfüllt wurde.

Anforderung	Referenz	Status
Sensoren anlegen	Tabelle 3.1	Erfüllt
Sensoreniste anzeigen	Tabelle 3.2	Erfüllt
Sensor bearbeiten	Tabelle 3.3	Erfüllt
Sensor aktivieren	Tabelle 3.4	Teilweise erfüllt
Technische Anforderungen	Kapitel 3.2	Erfüllt

Tabelle 5.2: Übersicht der UseCases und ihrem Status

6 Zusammenfassung und Ausblick

Abschließend zu diesem Projekt lässt sich zusammenfassen, dass die gesetzten Ziele erreicht wurden. Der WebService legt den Grundstein für eine einfache Konfiguration der Sensoren. Eine wesentliche Erneuerung stellt dabei die Möglichkeit des Aktivierens und Deaktivierens eines Sensors dar. Bisher musste man dazu immer direkt an der Konfiguration des RPProviders arbeiten.

Durch das Web-Frontends wurde eine benutzerfreundliche Schicht aufgesetzt, die es ermöglicht, plattformunabhängig die Konfiguration der Sensoren zu verändern.

Die in Kapitel 3 erläuterten Anwendungsfälle wurden implementiert. Verfügbare Sensoren werden vom SensorConfigurator gesucht und als Resourcen bereitgestellt. Das Frontend fragt über einen standardisierten GT4-WebService nach der Liste und stellt sie übersichtlich auf dem Frontend dar. Durch einen einfachen klick in der Liste lassen sich Sensoren aktivieren und deaktivieren. Die notwendigen Änderungen werden jeweils an den WebService übermittelt und dort gespeichert.

Auch der dritte UseCase, Sensoren bearbeiten, ist zufriedenstellend gelöst. Durch auswählen eines Sensors in der Übersicht werden die Daten des Sensors in einer Baumstruktur dargestellt. Die Einführung eines eigenen XML-Formats ermöglicht es, für verschiedene Datenfelder unterschiedliche Einschränkungen festzulegen. Damit ist es zum einen möglich bestimmte, falsche Informationen direkt auszuschließen, auf der anderen Seite wird dem Benutzer die Konfiguration erleichtert.

Das Erstellen eines neuen Sensors ist von einem Administrator relativ einfach durchzuführen. Er loggt sich per SSH in das System ein und erstellt dort die beiden notwendigen Dateien. Die Konfiguration des RP-Providers ähnelt sich dabei im allgemeinen. Es genügt daher meist, eine bestehende Konfiguration zu kopieren. Lediglich die Pfadangaben müssen angepasst werden. Für die Erstellung der sensor-data.xml ist im allgemeinen eine Vorlage vorhanden, wie der fertige Sensor auszusehen hat. Damit lässt sich auf relativ einfache Art und Weise der Sensor erstellen.

Alle geforderten Anwendungsfälle werden somit vom Frontend bereit gestellt und auf der anderen Seite von einem WebService ausgeführt. Lediglich das zuletzt auf der Backend-Seite aufgetretene Problem mit dem RP-Provider ließ sich nicht lösen.

Das Ergebnis dieses Projekts ist eine einfache Fernkonfiguration für die Konfiguration von Sensoren im Grid. In dem weiteren Prozess kann zusammen mit der GT4-Community eine Lösung für den RP-Provider gefunden werden. Darüber hinaus wird es kein großes Problem darstellen ein geegnetes minimales Sicherheits-Konzept zu integrieren. Damit ist dieses System, auch für weitere Untersuchungen des Ressourcen-Monitorings am LRZ, eine wichtige Grundlage.

Literaturverzeichnis

[Apa 07] APACHE: *Apache Log4j*, 2007, http://logging.apache.org/log4j/ .

[Apac 08] APACHE: *Appache Struts*. Februar 2008, http://struts.apache.org .

[BS 06] BORJA SOTOMAYOR, LISA KAUFMAN: *Globus Toolkit 4 - Programming Java Services*. Morgan Kaufmann, 2006.

[COMM 07] COMMUNITY, GLOBUS: *GT 4.1.1 Component Guide to Public Interfaces: WS MDS UsefulRP*, 11 2007, http://www.globus.org/toolkit/docs/development/4.1.1/info/usefulrp/usefulrp-public-interfaces.html .

[Deve 04] DEVELOPMENT, JENKOV: *Prize Tags: Tree Tag*. www.jenkov.com, Oktober 2004.

[esc 07] *eScience*. Technischer Bericht, "Bundesministerium für Bildung und Forschung"BMBF, November 2007, http://www.bmbf.de/de/298.php .

[FKTT 98] FOSTER, I., C. KESSELMAN, G. TSUDIK und S. TUECKE: *A security architecture for computational grids*. ACM Conference on Computer and Communications Security, Seite 83 to 92, 1998.

[LR 00] LARS RÖWEKAMP, PETER ROSSBACH: *JSP-Tutorial*. 'ix', Seite 152, Juli 2000.

[Ora 07] ORACLE: *Oracle JDeveloper 10g (10.1.3) Documentation*, 10 2007, http://www.oracle.com/webapps/onlinehelp/jdeveloper/10.1.3?topic=adf_aboutmvc2_html .

[Wiki 07] WIKIPEDIA: *Servlet*. Wikipedia, Oktober 2007, http://de.wikipedia.org/wiki/Servlet .