Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BSC-2009-07

Bachelorarbeit

im Studiengang "Angewandte Informatik"

Implementierung eines Werkzeugs zur Visualisierung von CCS-Prozessen

Jan-Martin Kirves

Arbeitsgruppe

Datenbanken & Informationssysteme

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

25. August 2009

Georg-August-Universität Göttingen
Zentrum für Informatik

Goldschmidtstraße 7
37077 Göttingen
Germany

Tel. +49 (5 51) 39-1 72 010
Fax +49 (5 51) 39-1 46 93
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 25. August 2009

Bachelorarbeit

Implementierung eines Werkzeugs zur Visualisierung von CCS-Prozessen

Jan-Martin Kirves

25. August 2009

Betreut durch Prof. Dr. Wolfgang May
Arbeitsgruppe Datenbanken & Informationssysteme
Georg-August-Universität Göttingen

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung eines Visualisierungs-Werkzeugs für Prozessbäume als Erweiterung eines Apache Tomcat Web Servers. Es wird im Framework „Modular Active Rules for the Semantic Web“ Verwendung finden und darin die Fehleranalyse erleichtern.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit beschäftigt sich mit der Implementierung einer Visualisierung von Prozessen, die nach dem Kalkül für kommunizierende Systeme, im englischen „Calculus of Communicating Systems“ (CCS), gestaltet sind und im Framework für „Modular Active Rules for the Semantic Web“(MARS) Verwendung finden.

Das MARS-Framework stellt eine eine Infrastruktur für Kooperation im Semantic Web zur Verfügung. Dabei werden unter der Verwendung von aktiven Regeln, die auf bestimmte Ereignisse reagieren, entsprechende Prozesse gestartet, die wiederum Ereignisse auslösen können.

Ein Beispiel für ein solches Ereignis wäre der Eingang einer Anfrage von Person B für einen Termin mit Person A. Das bedeutet, wenn die Anfrage für einen Termin eintrifft, muss der Terminkalender von Person A eingesehen werden, die freien Termine von Person A gefunden und mit dem gewünschten Termin angeglichen werden. Wenn ein passender Termin gefunden wurde, muss dieser in den Kalender von Person A eingetragen werden. Anschließend muss eine E-Mail erstellt werden, die das Datum, die Uhrzeit und die beteiligten Personen (Person B) des Termins enthält. Person A wird mittels der erstellten E-mail darüber informiert, dass sie einen Termin mit Person B hat. Person B wird ebenfalls mit dieser E-mail darüber informiert, dass sie den Termin mit Person A hat. Wenn kein freier Termin für den Vorschlag von Person B in Person A’s Kalender vorliegt, muss eine E-mail an Person B erstellt werden, die eine Terminabsage enthält. Diese muss dann an Person B geschickt werden.

Dieses konkrete Beispiel ist in Abbildung 1.1 veranschaulicht und zeigt alle beschriebenen Schritte in zeitlicher Abfolge.

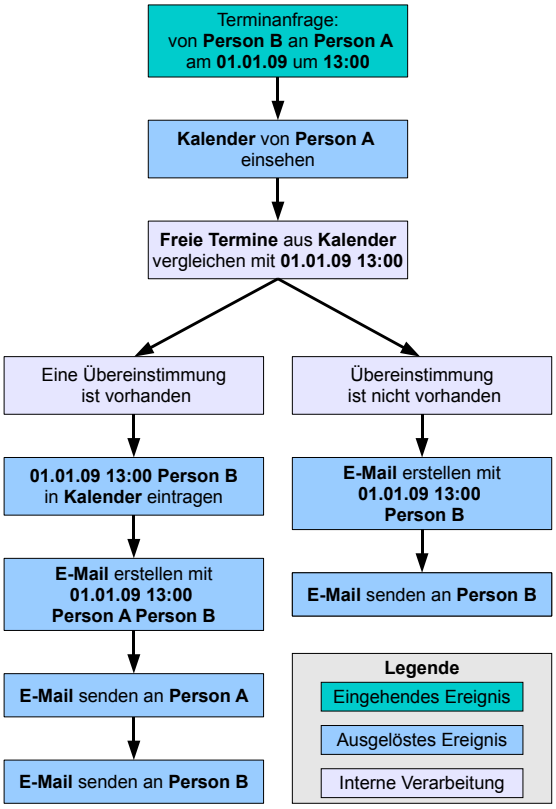

Abbildung 1.1: Beispiel für einen Prozess Ablauf

An diesem Beispiel ist bereits zu erkennen, dass auf ein einfaches Ereignis ein komplexer Prozess als Reaktion erfolgen kann. Um einen solchen Prozess verarbeiten zu können, ist eine Sprache notwendig, wie sie durch CCS gegeben wird.

Mit dieser Sprache kann ein Prozess modelliert werden, der den oben beschriebenen Ablauf darstellt und Schritt für Schritt dessen Abarbeitung ermöglicht. Dabei ändert sich in jedem Schritt der Zustand des Prozesses.

In einigen Schritten entstehen neue Informationen, die gespeichert werden müssen. Im Beispiel: Einträge im Terminkalender von Person A oder die erstellten Nachrichten. In anderen Schritten müssen gespeicherte Informationen evaluiert werden. Somit wird, indem der Wunschtermin mit dem Kalender angeglichen wird, dem Ergebnis entsprechend reagiert.

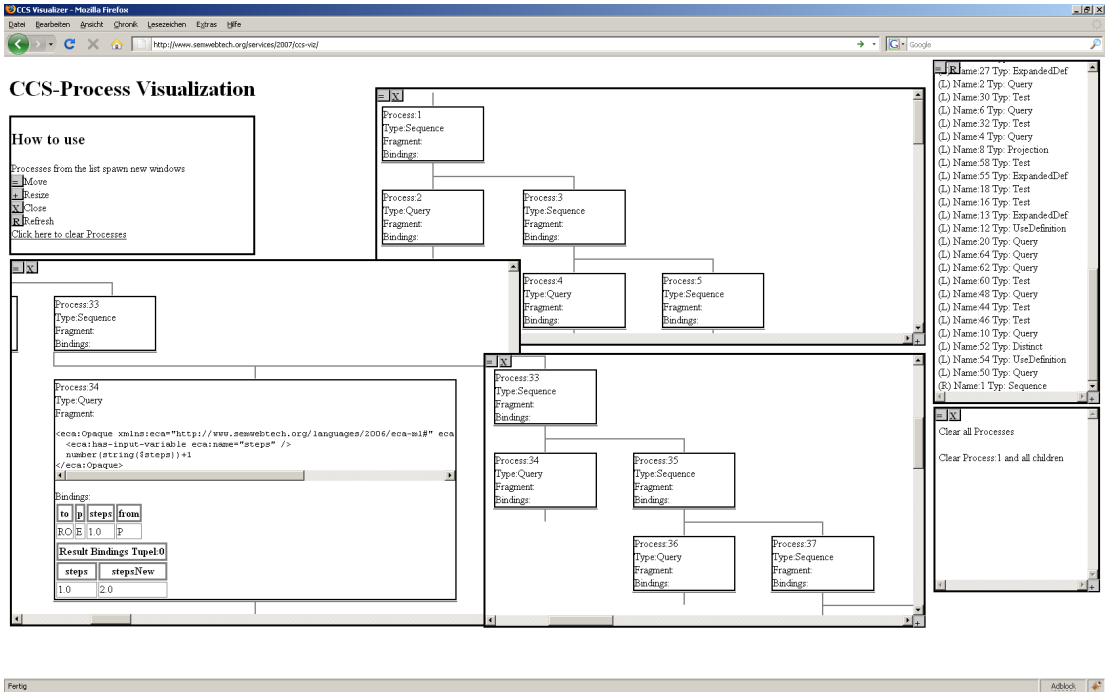

Abbildung 1.2: Darstellung des JavaScript-Interfaces. Zu sehen ist eine Prozessauswahl, mehrere Fenster mit Prozessbäumen, Fenster zum Löschen von Prozessen und Prozessdetails

Bei einer solchen Prozesskette müssen neue Unter-Prozesse, so genannte Kind-Prozesse, gestartet werden. Jeder Kind-Prozess kennt die Daten seines erzeugenden Prozesses (Vater-Prozess) und kann diese Daten verändern und eigene Kind-Prozesse erzeugen. Auf diese Weise entsteht schon bei kleinen Prozessabläufen ein größerer Prozessbaum,

der die Hierarchie zwischen den einzelnen Prozessen darstellt.

Im obigen Beispiel ist das Erstellen der E-Mail ein Kind-Prozess, der gestartet wird. Darüber hinaus können Prozesse parallel ablaufen - zum Beispiel zwei Terminanfragen gleichzeitig eintreffen - so dass zwei Prozessbäume zeitgleich erzeugt werden.

Dadurch ist es für die Entwicklung von Anwendungen im MARS-Framework wichtig, eine Übersicht der aktuellen Prozesse, ihren Zuständen und ihrer Hierarchie zu haben. Es muss also eine Oberfläche geschaffen werden in der dem Nutzer alle Informationen über die aktuellen Prozesse des MARS-Frameworks präsentiert werden.

Um dies zu ermöglichen, wird dieses Projekt alle aktuellen CCS-Prozesse des Frameworks in Form eines Baums visualisieren und Zugriff auf alle Daten der Prozesse realisieren. Das bedeutet, dass jeder Schritt, den ein Prozess durchläuft, zusammen mit dem Prozesszustand und all seinen Informationen im Prozessbaum dargestellt wird und detailierte Informationen mit einem Klick abgerufen werden können. Eine solche Oberfläche ist in Abbildung 1.2 dargestellt. Auf der rechten Seite ist eine Liste von allen aktuellen Prozessen zu sehen sowie in der Bildmitte mehrere parallele Prozess-Bäume. Prozessdetails sind links unten im Bild zu sehen.

Der Visualisierer wird darüber hinaus als Web Service implementiert, damit ein einfaches Zusammenspiel in einem größeren Netz möglich wird. [6]

2 Grundlagen

Im Folgenden wird auf die in diesem Projekt verwendeten Techniken eingegangen.

2.1 CCS-Prozesse

Es handelt sich bei CCS um ein Kalkül zur Beschreibung von Repräsentanten, die synchron oder asynchron Nachrichten miteinander austauschen. Ein solcher Repräsentant ist in diesem Fall ein Prozess und kann mit Hilfe dieses Kalküls im Vorfeld formal definiert werden.

Die CCS-Algebra über einer Menge A von atomaren Bestandteilen definiert sich als:

$a \in A$

X	ist eine Prozessvariable.
P, Q	sind Prozessausdrücke.
$X := P$	ist eine Prozessdefinition.
a,X, $a.P$	ist eine sequenzielle Ausführung von Bestandteil a des Prozesses P.
$seq(P, Q)$	ist die sequenzielle Abfolge von Prozess P gefolgt von Prozess Q.
$P \mid Q$	ist die gleichzeitige Ausführung der Prozesse P und Q.
$P+Q$	ist die alternative Ausführung der Prozesse P und Q.
0	ist das Ende.

Die exakte Semantik ist in [11] zu finden.

Bei der Ausführung von z.B. *a.P* wird Bestandteil *a* ausgeführt und es bleibt nur noch *P* ohne *a* für die weitere Ausführung über. Ein Prozess ändert sich also während seiner Ausführung mit jedem Schritt. Dabei ist *a.P* ein Sonderfall von *seq(Q, P)* wobei $Q = a$ atomar ist.

Der Zustand eines Prozesses in CCS ist nur durch sein Verhalten, also seine möglichen Aktionen, kodiert. [16]

Die CCS-Algebra wurde von Robin Milner 1980 eingeführt und ermöglicht die formale Beschreibung von Prozessen sowie die qualitative Überbrüfung auf deren Korrektheit.

Für das MARS-Framework wurde CCS erweitert. Es wird die „Extensible Markup Language“(XML), zu deutsch erweiterbare Auszeichnungssprache, genutzt, um einen Prozess zu formulieren. In Abbildung 2.1 sind die grundlegenden Elemente, aus denen eine Prozessdefinition aufgebaut werden kann, als XML-Markup dargestellt.

Jeder Prozess hat eine eigene Definition, die durch ein XML-Fragment dargestellt wird. Dieses Fragment ist zentraler Bestandteil des Prozesses. Abbildung 2.2 zeigt die Definition des in Kapitel 1 beschriebenen Prozesses als XML-Fragment.

Außerdem werden im MARS-Framework atomare Bestandteile eines CCS-Prozesses nicht mehr nur als Aktionen aufgefasst, sie können nun Ereignisbeschreibungen (event-specifications), Anfragen (queries), Tests (tests) und auch atomare Aktionen (atomic actions) sein. [16]

Darüber hinaus kann ein Prozess mit Hilfe des erweiterten CCS im MARS-Framework nun Daten in Variablenbindungen speichern. Dabei handelt es sich um Tupel, die eine Tabelle darstellen, welche sowohl alle Variablen eines Prozesses enthält als auch die möglichen Bindungen von Werten der einzelnen Variablen. [16]

In Abbildung 2.2 sind alle Namen, denen „$“ voran steht, Variablen (“$pa“, „$pb“, „$kalender“, „$datum“, „$zeit“ und „$mail“). Diese bilden die Spaltenüberschriften der Variablenbindungstabelle. Die Werte, die ihnen zugewiesen werden, sind in der

jeweiligen Spalte der Tabelle zu finden. So wird im Beispiel der Variablen „$pa“ der Name von Person A zugewiesen und der Variablen „$kalender“ eine Liste von freien Terminen. Jede Tabellenzeile ist stets komplett gefüllt. Das heißt, es steht in der Spalte unter „$pA“ in jeder Zeile „PersonA“. Unter „$kalender“ stehen hingegen die verschiedenen freien Termine.

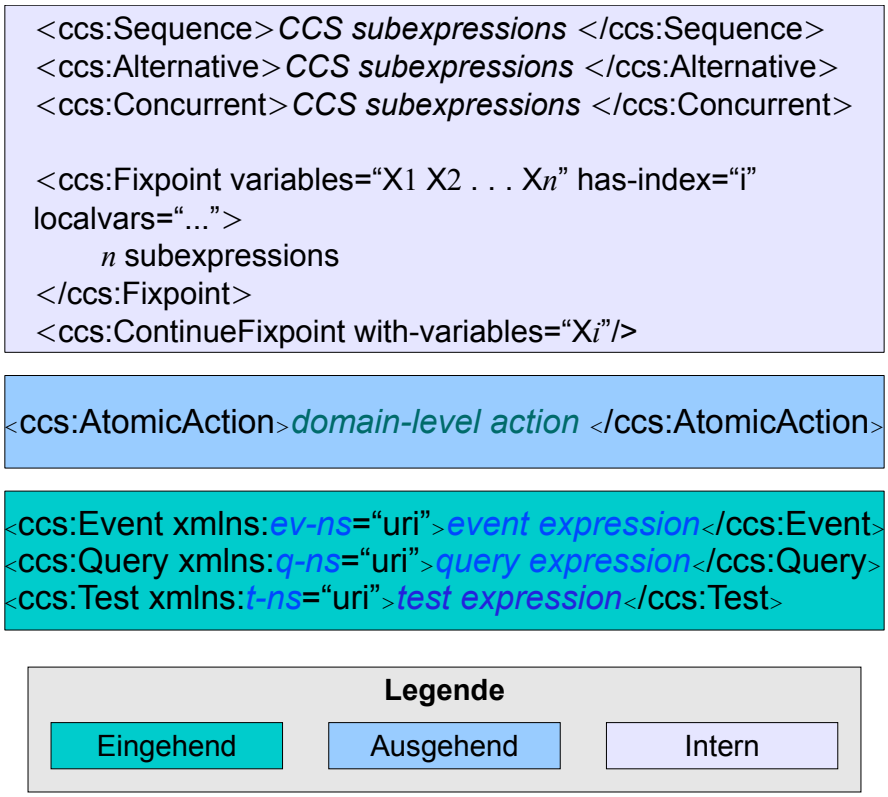

Abbildung 2.1: Die grundlegenden Elemente von CCS als XML-Markup

Der Zustand eines Prozesses ergibt sich aus seiner Definition und seinen Variablenbindungen. Bei der Ausführung eines Prozesses wird für jedes in Abbildung 2.1 aufgelistete Element ein Kind-Prozess gestartet, dessen definierendes XML-Fragment der Inhalt des Elementes ist, von dem aus er aufgerufen wurde.

In Abbildung 2.2 beginnt der Prozess bei dem ersten „<ccs:Sequence>“. Dieser ist durch das XML-Fragment definiert, das von dem ersten „<ccs:Sequence>“ und dem letzten „</ccs:Sequence>“ eingerahmt wird. Er hat die Variable „$pa“ mit dem Namen von Person A, also „PersonA“, gebunden. Für „$pb“ ist „PersonB“ gebunden, für „$datum“, das Datum und für „$zeit“, die Zeit des Wunschtermins. Dieser Prozess ist der Wurzel-Prozess und besitzt als einziger Prozess im Prozess-Baum keinen Vater-Prozess.

```
<eca:Rule xmlns:termine="http://www.termine.de">
 <eca:Event> Terminanfrage – bindet $pa $pb $zeit $datum </eca:Event>
 <eca:Action xmlns:ccs="http://www.semwebtech.org/languages/2006/ccs#">
   <ccs:Sequence>
    <ccs:Atomic> Aufruf von Kalender von $pa </ccs:Atomic>
    <ccs:Event> Antwort bindet $kalender</ccs:Event>
    <ccs:Query> Ist $datum und $zeit frei in $kalender </ccs:Query>
    <ccs:Alternative>
     <ccs:Sequence>
      <ccs:Test> ja </ccs:Test>
      <ccs:Atomic> Buche $zeit $datum in $kalender </ccs:Atomic>
      <ccs:Atomic> $mail := „Zusage für $zeit $datum $pa $pb“ </ccs:Atomic>
      <ccs:Atomic> Senden von $mail an $pa </ccs:Atomic>
      <ccs:Atomic> Senden von $mail an $pb </ccs:Atomic>
     </ccs:Sequence>
     <ccs:Sequence>
      <ccs:Test> nein</ccs:Test>
      <ccs:Atomic> $mail := „Absage für $zeit $datum $pa“ </ccs:Atomic>
      <ccs:Atomic> Seden von $mail $pb </ccs:Atomic>
     </ccs:Sequence>
    </ccs:Alternative>
   </ccs:Sequence>
 </eca:Action>
```

Abbildung 2.2: Beispiel in CCS Syntax

Der erste Kind-Prozess wird durch „<ccs:Atomic>“ erzeugt. Dieser Prozess kennt die Variablenbindungen und somit durch die Variable „$pa“ den Namen von Person A. Er startet einen externen Aufruf des Terminkalenders von Person A. Dieser Kind-Prozess ist damit beendet und es wird in den Vater-Prozess, den eben beschriebenen

Wurzel-Prozess, zurück gesprungen.

Im Folgenden wird erneut ein Kind-Prozess mittels „<ccs:Event>“ erzeugt. Dieser Prozess fügt den Variablenbindungen die Variable „$kalender“ hinzu und bindet an diese alle freien Termine des Terminkalenders von Person A, die aus der zuvor gestellten Anfrage ermittelt wurden. Damit endet dieser Kind-Prozess und der Vater-Prozess, noch immer der Wurzel-Prozess, verfügt nun über die freien Termine von Person A.

Nun kann der nächste Kind-Prozess, „<ccs:Query>“, erzeugt werden. Hier wird überprüft, ob sich ein freier Termin im Kalender am Wunschtermin von Person B befindet. Dabei werden die Variablenbindungen reduziert, es bleiben nur die Tabellenzeilen übrig, in denen sich ein freier Termin mit dem Wunschtermin deckt. Dieser Kind-Prozess endet und der Vater-Prozess, wiederum der Wurzel-Prozess, hat nun entweder noch eine Zeile in seinen Variablenbindungen oder keine mehr, je nachdem ob es eine Übereinstimmung gab oder nicht.

Der nächste Kind-Prozess, „<ccs:Alternative>“, wertet dieses aus und erzeugt einen weiteren Kind-Prozess, „<ccs:Sequence>“, je nachdem ob noch eine Variablenbindung für den Termin gesetzt ist oder nicht. Wenn noch eine Bindung vorhanden ist, so wird die erste „<ccs:Sequence>“, in der „<ccs:Test>“ mit „Ja“ bestanden wird, ausgeführt. Ansonsten wird die zweite „<ccs:Sequence>“ ausgeführt, in der der „<ccs:Test>“ mit „Nein“ nicht bestanden wurde.

Der weitere Ablauf in Abbildung 2.2 folgt dem eben beschrieben Schema bis alle Kind-Prozesse beendet sind und lediglich der Wurzel-Prozess beendet werden muss. [6]

2.2 Extensible Markup Language (XML)

Es handelt sich bei XML um eine frei definierbare Metasprache. Die Elemente aus denen eine spezielle Sprache besteht, werden vorher festgelegt. So kann eine Datenstruktur erzeugt werden, die den Eigenschaften dieser Sprache folgt. [17]

XML findet als Übertragungsformat eine große Anwendung im MARS-Framework. Ebenfalls wird XML zur Definition der CCS-Prozesse genutzt.

2.3 Backend

Es folgt nun eine Beschreibung der auf Serverseite verwendeten Techniken.

2.3.1 Java Servlet

Bei Java Servlets handelt es sich um Instanzen von bestimmten Java Klassen, die innerhalb eines Java-Webservers, wie zum Beispiel dem Apache Tomcat, laufen und diesen erweitern. Durch diese Erweiterung wird es möglich, mit Klienten über das „Hyper Text Transfer Protocol“(HTTP), was übersetzt Hyper Text Übertragungsprotokoll bedeutet, zu kommunizieren. Anfragen an den Web-Server können so dynamisch ausgewertet und beantwortet werden. Damit stellen Java Servlets eine Alternative zu „Common Gateway Interface“ (CGI)-Scripten dar.

Servlets werden zu Paketen, sogenannten Web-Archiven zusammengefasst. Diese enthalten kompilierten Java Code, der die für das Servlet nötigen Klassen bereitstellt, sowie eine XML-Datei (web.xml) mit Meta-Informationen über das Servlet. Diese Pakete können plattformunabhängig jedem Java Web-Server zugänglich gemacht werden. [13] [2]

2.3.2 Java Database Connectivity (JDBC)

Die Datenbankschnittstelle unter Java wird „Java Database Connectivity“(JDBC) genannt, zu deutsch Java Datenbank Verbindungsfähigkeit. Es ist eine universelle Schnittstelle zu Datenbanken verschiedener Hersteller. Für verschiedene Datenbanken wird ein Treiber benötigt, die Ansteuerung der Datenbanken ist aber, durch die einheitliche Programmierschnittstelle von JDBC identisch. So ist es sehr einfach möglich das Programm auf eine andere Datenbank zu portieren. [15] [8]

2.4 Frontend

Das Frontend ist der Teil der Applikation, die der Nutzer sieht. Es stellt die Schnittstelle zwischen Nutzer und dem Programm dar. Der Nutzer kann hier alle Informationen entnehmen sowie seine Befehle an das Programm schicken. Es folgt die Beschreibung der Techniken, die im Frontend Verwendung finden.

2.4.1 Hyper Text Markup Language (HTML)

Es handelt sich bei der „Hyper Text Markup Language“(HTML), im Deutschen Hyper Text Auszeichnungssprache, um eine Auszeichnungssprache, die genutzt wird, um Dokumente zu strukturieren. Dabei können verschiedene Inhalte, sowie Meta-Informationen über das Dokument selbst berücksichtigt werden. Die Sprache hält verschiedene Markierungen, genannt „Tags“, bereit, mit denen Inhalte gruppiert und formatiert werden können. Die Formatierung orientiert sich an der Anzeige auf Bildschirmen und nicht an Papierseiten wie bei gedruckten Dokumenten.

Es wird zwischen Struktur mit Inhalt und Layout unterschieden. Die Struktur wird über die verschiedenen zur Verfügung stehenden Tags realisiert, das Layout über eine separate Formatvorlage, sogenannte „Cascading Style Sheets“(CSS), zu deutsch Kas-

kadierende Formatvorlage, gesteuert. Das CSS beinhaltet Informationen darüber, wie verschiedene Einteilungen des Dokuments letztlich dargestellt werden. Dies ermöglicht eine einfache Anpassung des HTML-Dokuments an verschiedenste Anforderungen, wie zum Beispiel für Sehbehinderte. Hier kann einfach ein höherer Kontrast sowie eine größere Schriftart realisiert werden. [3]

2.4.2 Java Server Pages (JSP)

Eine Technik, die von Java zur Verfügung gestellt wird, sind „Java Server Pages“(JSP). Es handelt sich hierbei um HTML-Dokumente, die auf dem Server dynamisch durch Java Code modifiziert werden können.

So wird serverseitig ein HTML-Dokument erzeugt und kann auf eine Anfrage hin entsprechende Änderungen enthalten. [14] [2]

2.4.3 JavaScript

JavaScript is eine vom Web-Browser interpretierte Skriptsprache. Das dieser Sprache zugrundeliegende Datenmodell ist das „Document Object Model“, eine Baumstruktur die alle Daten bzw. Instanzen von Objekten des Scripts und die HTML Daten enthält.

So ist es mit JavaScript möglich, dynamisch und während der Anzeige im Browser Änderungen als Reaktion auf Nutzeranfragen durchzuführen. Da JavaScript vom Browser interpretiert wird, ist das Verhalten von verschiedenen Web-Browsern auf den gleicher JavaScript Code oft unterschiedlich und es muss zwischen den Browsern im Quellcode unterschieden werden, um ein Programm zu entwickeln, das auf allen Browsern die selbe Funktionalität zur Verfügung stellt. [4]

2.4.4 Asynchronous JavaScript and XML (AJAX)

Unter „Asynchronous JavaScript and XML“(AJAX) werden Techniken der Web- Entwicklung zusammengefasst, die es ermöglichen, Daten zwischen Klient und Server auszutauschen. Wichtig ist dabei, dass dies über JavaScript geschieht und kein erneutes Laden der HTML-Seite nach sich ziehen muss. So können Webseiten erstellt werden, die wie herkömmliche offline Anwendungen zu bedienen sind. [9] [12]

3 Entwurf

Dieses Projekt ist als Erweiterung für einen Apache Tomcat Server konzipiert. Dies bedeutet, es ist ein Paket, welches auf jedem Tomcat Server installiert werden kann. Um das Programm mit den entsprechenden Informationen zu versorgen, ist ein Plugin für das MARS-Framework vorgesehen, welches Methoden zur Kommunikation mit dem Visualisierungswerkzeug zur Verfügung stellt.

Wie in Abbildung 3.1 gezeigt, kann auf einen installierten CCS-Visualisierer von mehreren MARS-Frameworks über das HTTP-Protokoll als Web Service zugegriffen werden. Die Daten der einzelnen Prozesse werden in der eigenen Datenbank des entsprechenden Frameworks gehalten und dem Visualisierer über eine direkte Verbindung via „Transmission Control Protocol/Internet Protocol“(TCP/IP), zu deutsch Übertragungskontroll-Protokoll / Internet Protokoll über JDBC zugänglich gemacht.

Der Nutzer kann die Darstellung der aktuellen Prozessbäume mit Hilfe eines Browsers über das Web-Interface abrufen. Dies kann einerseits über das JavaScript-Frontend geschehen. Hier werden die Daten dynamisch per AJAX-Anfragen vom Visualisierer bezogen und im Browser dargestellt. Andererseits kann über eine statische HTML-Seite eine Get-Anfrage direkt geschickt und die aktuelle Anzeige als komplette HTML-Seite zurück zum Browser übertragen werden.

Durch die Verwendung von Java ist eine Plattformunabhängigkeit für die Installation des Visualisierers gegeben und der Web Service stellt als Schnittstelle die Interoperabilität zwischen verschiedenen Betriebssystemen sicher.

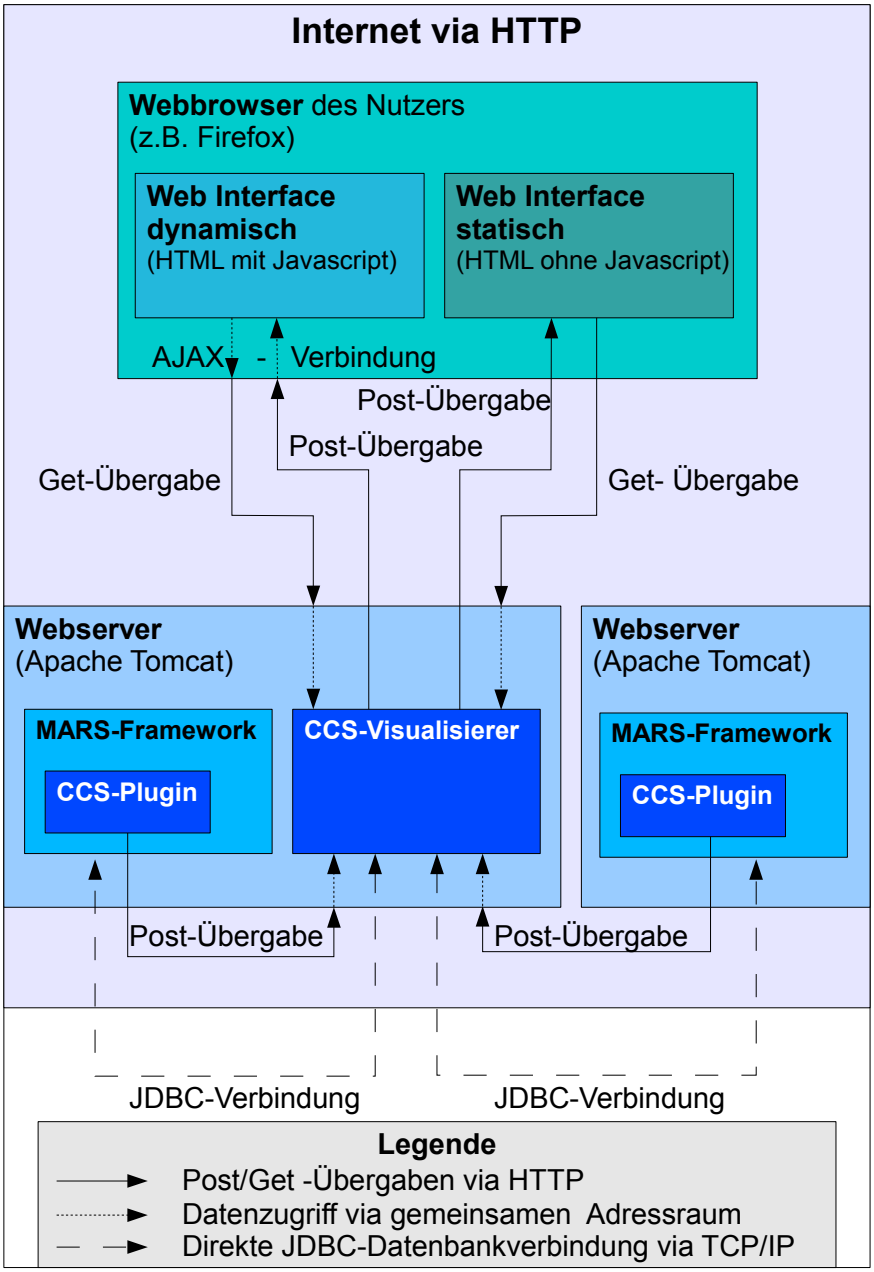

Abbildung 3.1: Konzeptueller Aufbau des CCS-Visualisieres

3.1 Klassenstruktur

Das Paket „ccs-viz.war“, welches auf einem Apache Tomcat Webserver installiert werden kann, enthält die Hauptfunktionalität des Projektes. Es besteht aus folgenden Klassen:

Klasse: *ccs_viz_servlet*

Diese Klasse nimmt Post- und Get-Anfragen entgegen und wertet diese aus, um weitere Funktionalitäten des Visualisierers aufzurufen.

Klasse: *cc_sviz_manager*

Bei dieser Klasse handelt es sich um die Verwaltungseinheit des Projektes. Sie ist statisch implementiert und hält alle CCS-Prozesse. Durch sie wird die Erstellung der angeforderten Prozessbäume durchgeführt.

Klasse: *ccs_viz_process*

Hierbei handelt es sich um die Klasse, deren Instanzen einen expliziten CCS-Prozess darstellen. Jede Instanz kennt ihre Vater- und Kind-Prozesse und verfügt über die Möglichkeit, auf die Daten des Prozesses aus der Datenbank des MARS-Framework via JDBC-Verbindung zuzugreifen.

Klasse: *ccs_viz_dbHelper*

Dies ist eine statische Klasse, die Methoden zum Zugriff auf Datenbanken via JDBC bereit hält.

Klasse: *ccs_viz_task*

Hierbei handelt es sich um eine Klasse, die einen Java *TimerTask* implementiert. Sie sorgt dafür, dass CCS-Prozesse nach einer gewissen Zeit gelöscht und ihre Daten aus der entsprechenden Datenbank entfernt werden.

Klasse: *ccs_viz_plugin*

Um das Einbinden in das MARS-Framework einfacher zu gestalten, gibt es diese statische Klasse, die nicht im Paket *ccs-viz.war* enthalten ist. Sie ist direkt im Framework eingebunden und muss vorort angesprochen werden. Die Klasse *ccs_viz_plugin* hält Methoden bereit, um Daten vom MARS-Frontend in einer bestimmten Formatierung an den CCS-Visualisierer per Post-Übergaben über das HTTP-Protokoll zu schicken.

3.2 Datenspeicherung

Ein CCS-Prozess besteht aus verschiedenen Daten, die es für die Visualisierung zu speichern gilt. Jeder Prozess hat einen Typ, eine Identifikation (Id) , ein XML Fragment welches seine Definition enthält und einen Satz von Variablenbindungen.

Die Variablenbindungen werden generell in einer Datenbank im MARS-Framework hinterlegt, da es sich hierbei um große Mengen handeln kann. Diese Bindungen müssen also nicht noch einmal in Kopie zum Visualisierer gesendet und separat gespeichert werden. Es genügt, die Verbindungsdaten zur Datenbank des MARS-Frameworks zu speichern.

Die Daten eines Prozesses können sich im Laufe seines Lebens ändern. Alle Kindprozesse nutzen die selben Variablenbindungen und können diese ändern. Daher ist es wichtig, dass die Daten jeder Änderung erhalten bleiben, um eine Prozesskette später im Detail verfolgen zu können. Um dies zu gewährleisten muss eine Kopie der

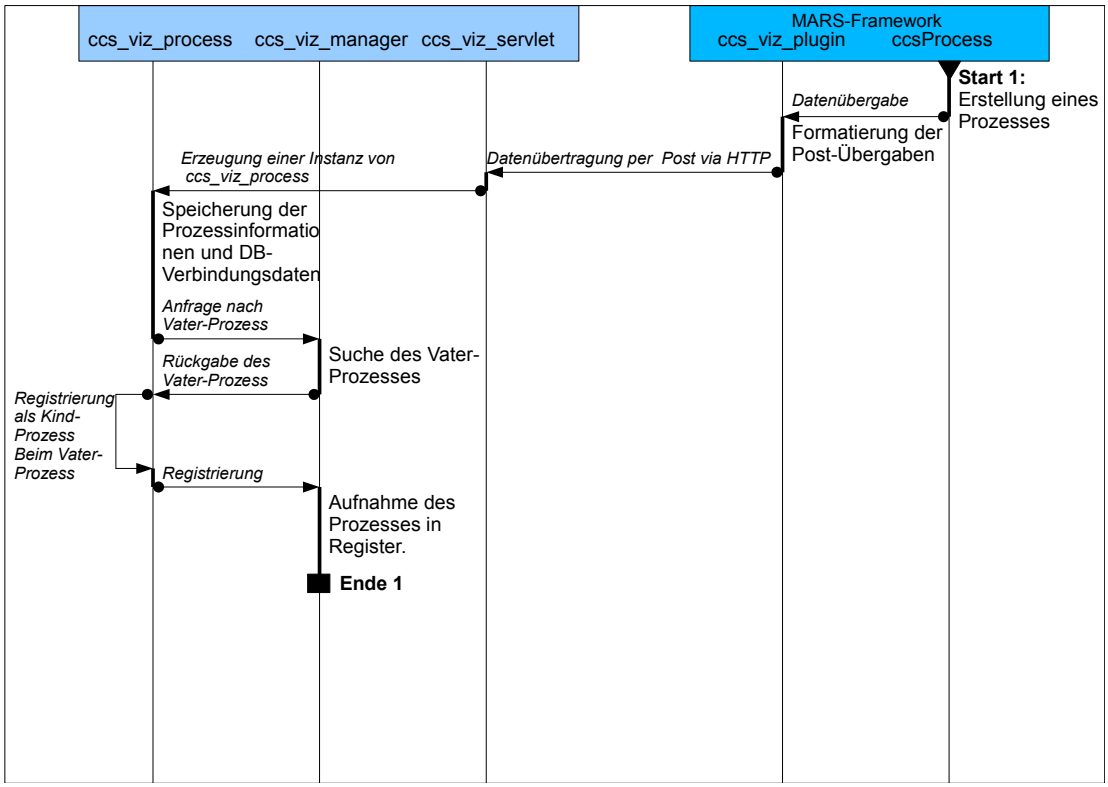

Abbildung 3.2: Ablauf der Erzeugung eines ccs_viz_process

Datenbanktabelle, welche die Variablenbindungen eines Prozesses hält, erstellt und die zugehörigen Zugangsdaten dem visualisierten Abbild zugänglich gemacht werden. Dieser Schritt wird von der Klasse *ccs_viz_plugin* bei jeder Erstellung eines neuen Prozesses durchgeführt, wie in Abbildung 3.2 zu sehen ist. Diese Klasse formatiert die Daten nun so, dass sie über das HTTP-Protokoll in Form einer Post-Übergabe an die Klasse *ccs_viz_servlet* übergeben werden können, zusammen mit dem Befehl, diese Daten als neuen *ccs_viz_process* zu registrieren.

Die Klasse *ccs_viz_servlet* instanziiert nun ein Objekt der Klasse *ccs_viz_process* mit den übermittelten Daten. In diesem Objekt werden alle Daten, also die Id des Prozesses, das XML-Fragment mit seiner Definition und die Datenbank-Verbindungsdaten

für die Kopie der Variablenbindungen gespeichert.

Um nun einen Baum der Prozesse darstellen zu können, ist es wichtig, dass jeder Prozess seinen Vater-Prozess und seine Kind-Prozesse kennt. Dazu stellt, wie in Abbildung 3.2 ersichtlich, der neu erzeugte *ccs_viz_process* eine Anfrage an die Klasse *ccs_viz_manager*, um seinen Vater-Prozess zu ermitteln.

Wenn ein Vater-Prozess vorhanden ist, registriert sich der Prozess bei diesem als Kind. Wenn nicht, muss es sich um einen Wurzel-Prozess handeln, welcher dann im *ccs_viz_manager* gesondert gespeichert wird.

Im letzten Schritt registriert sich der neuen *ccs_viz_process* beim *ccs_viz_manager*, um bei späteren Anfragen gefunden werden zu können.

3.3 JDBC Zugriffe

Zugriffe per JDBC auf die Datenbank des MARS-Frameworks sind vom Visualisierer nur lesend nötig. Die Kopie der Datenbanktabelle eines CCS-Prozesses wird erstellt, indem der Prozess durch das *ccs_viz_plugin* im MARS-Framework gecloned wird.

Wenn vom Nutzer eine Anfrage über das Web-Frontend gestellt wird, um Detailinformationen eines bestimmten Prozesses zu erhalten, wie zum Beispiel die Variablenbindungen, geschieht dies über das HTTP-Protokoll und wird von der Klasse *ccs_viz_servlet* entgegengenommen wie in Abb. 3.3 dargestellt. Diese Klasse stellt nun eine Anfrage zur Ermittlung des Prozesses an die Klasse *ccs_viz_manager*, diese gibt den entsprechenden Prozess zurück.

Die Instanz der Klasse *ccs_viz_process* wird nun nach ihren Variablenbindungen gefragt.

Wenn diese schon vorhanden sein sollten, werden sie als HTML-Fragment zurückgegeben und dann von der Klasse *ccs_viz_servlet* über das HTTP-Protokoll an das Web-Frontend gesandt, welches nun die Daten anzeigen kann (Alternative 1a in Abb. 3.3).

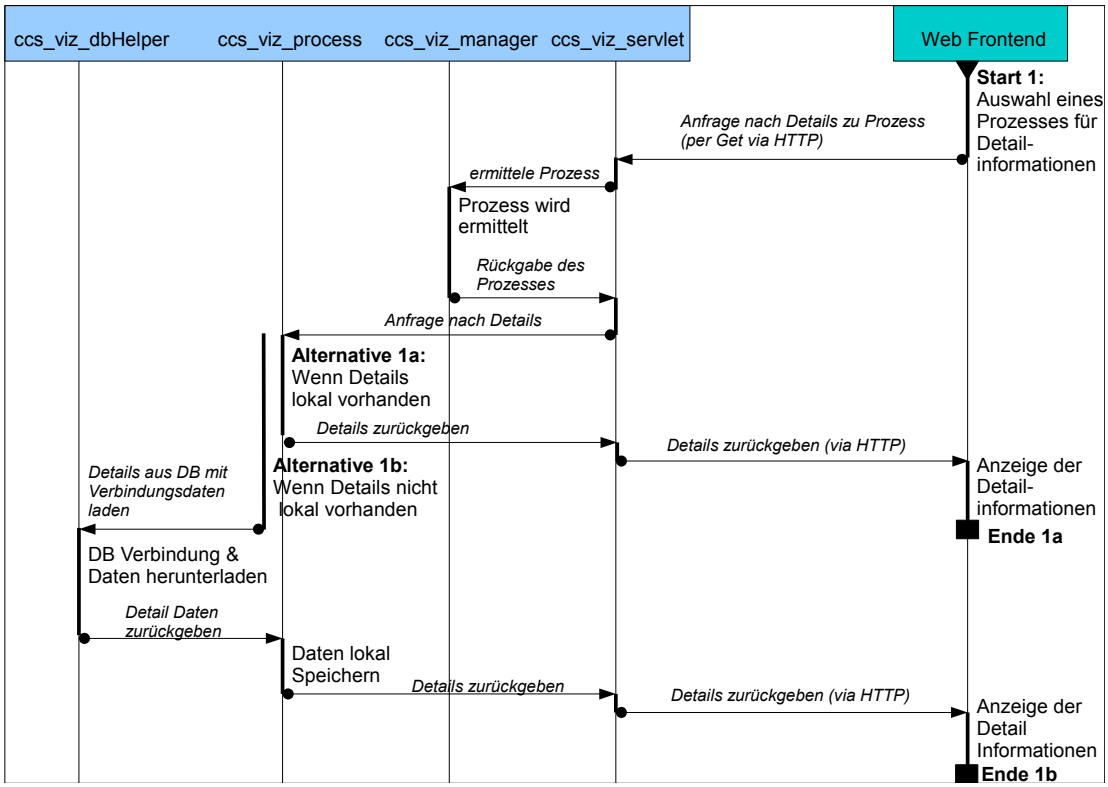

Abbildung 3.3: Zugriff auf Prozessdetails

Sind die Variablenbindungen noch nicht direkt vorhanden, so wird nach Alternative 1b aus Abbildung 3.3 verfahren. Das Prozessobjekt stellt mit Hilfe der Klasse *ccs_viz_dbHelper* eine Anfrage an die Datenbank des MARS-Frameworks. Die benötigten Datenbankinformationen, bestehend aus dem Datenbanknamen, dem Nutzernamen, dem Zugangspasswort und dem Tabellennamen sind im Prozessobjekt gespeicherten.

Die Klasse *ccs_viz_dbHelper* gibt dann die ausgelesenen Daten in Form eines HTML-Fragmentes an das *ccs_viz_process*-Objekt zurück. Dieses speichert die Daten für spätere Zugriffe, damit die Anzahl der Datenbankanfragen möglichst gering gehalten werden kann, und gibt das HTML-Fragment an die Klasse *ccs_viz_servlet* zurück.

Von hier werden nun die Variablenbindungen als HTML-Fragment über das HTTP-Protokoll zurück an das Web-Frontend gesendet, um sie dem Nutzer anzuzeigen.

3.4 Ausgabe

Wenn das Web-Frontend aufgerufen wird, wird dem Nutzer eine Liste von allen im *ccs_viz_manager* registrierten *ccs_viz_process*-Prozessen angezeigt.

Wie in Abbildung 3.4 ersichtlich, wird mit dem Initialisieren des Web-Interfaces eine Get-Anfrage an die Klasse *ccs_viz_servlet* über das HTTP Protokoll geschickt, um die Liste aller Prozesse abzurufen.

Von der Klasse *ccs_viz_servlet* wird nun die statische Klasse *ccs_viz_manager* angewiesen, aus ihren Daten, die sie wie in Kapitel 3.2 beschreiben erhalten hat, eine Prozessliste zu generieren. Diese wird dann in Form eines HTML-Fragments an das Servlet zurückgegeben, welches dieses Fragment wiederum an das Web-Interface über das HTTP-Protokoll überträgt.

Hier kann nun der Nutzer, wie in Abbildung 3.4 zu sehen ist, einen Prozess als Wurzel für seinen Prozessbaum auswählen.

Wenn nun ein Prozess vom Nutzer als Wurzel des zu betrachtenden Prozessbaums gewählt wurde, wird eine Anfrage vom Web-Interface über das HTTP-Protokoll an die Klasse *ccs_viz_servlet* geschickt, um den gewünschten Baum zu erzeugen.

Die Klasse *ccs_viz_manager* wird von der Klasse *ccs_viz_servlet* angewiesen, den als Wurzel gewünschten *ccs_viz_process* zu ermitteln und zurückzugeben. Wenn das Servlet diesen Wurzel-Prozess kennt, weist es diesen an, rekursiv den Prozessbaum seiner Kind-Prozesse zu generieren.

Nachdem der Prozessbaum zurückgegeben wurde, werden die einzelnen Prozesse vom *ccs_viz_servlet* angewiesen, eine entsprechende Ausgabe zu generieren. Dabei wird unterschieden, ob die Ausgabe für das JavaScript-Frontend oder das einfache

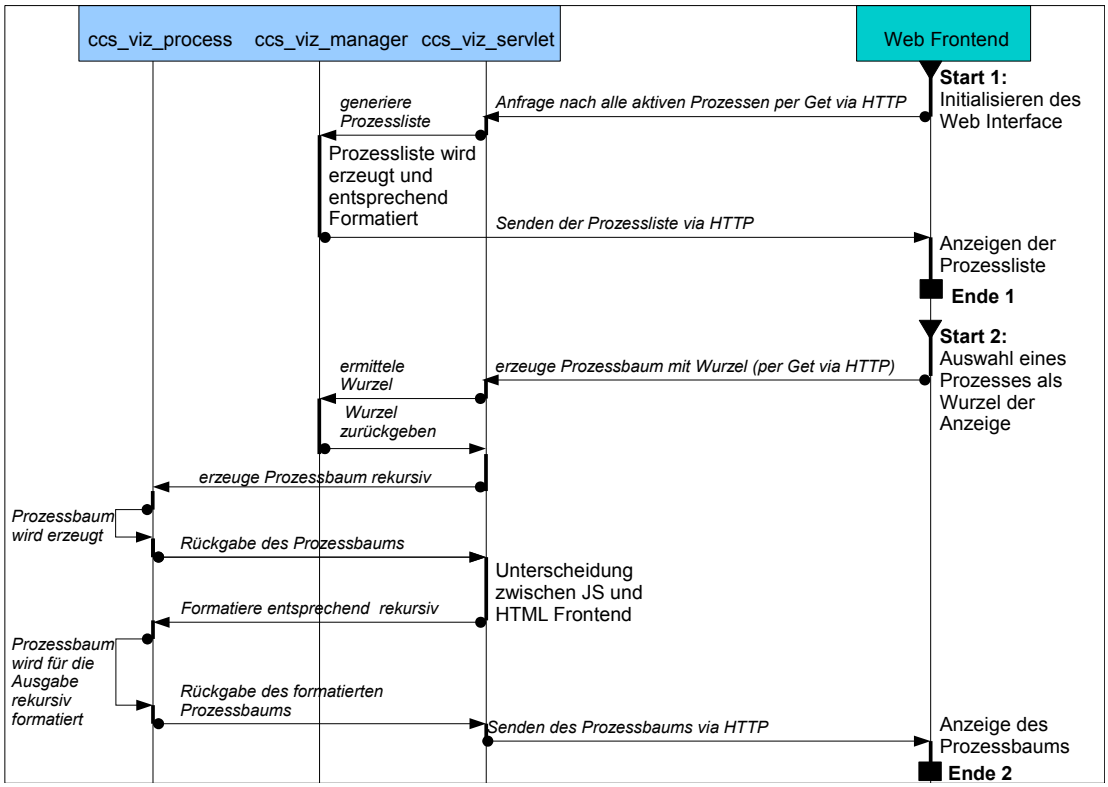

Abbildung 3.4: Auswahl von Prozessen

HTML Frontend benötigt wird. Wenn es sich um das JavaScript-Frontend handelt, wird eine durch Kommata getrennte Liste erstellt; für das HTML Frontend wird ein HTML Dokument erstellt, welches den Prozessbaum als Tabelle enthält.

Diese generierten Ausgaben werden über das HTTP Protokoll an das Web Interface übertragen.

3.5 Aufräumroutine

Da die Daten der einzelnen *ccs_viz_process*-Objekte in Tabellen auf einer entfernten Datenbank gespeichert werden, ist es notwendig, dafür zu sorgen, dass diese Tabellen

gelöscht werden, sobald ein Prozess nicht mehr benötigt wird.

In Abbildung 3.5 ist im ersten Ablauf zu sehen, dass, wenn ein neuer *ccs_viz_process* instanziiert und diesem kein Vater-Prozess zugewiesen wird, ein *ccs_viz_task* Objekt erzeugt wird, welches die Referenz auf den neu erzeugten Prozess speichert.

Dieses Objekt bekommt bei seiner Erstellung die Referenz auf das neue *ccs_viz_process*-Objekt übergeben und kennt so einen Wurzel-Prozess. Diesen und alle seine Kind-Prozesse wird das *ccs_viz_task*-Objekt später löschen können.

Nachdem das *ccs_viz_task*-Objekt gespeichert wurde, wird das *ccs_viz_process*-Objekt, wie schon in Abbildung 3.2 gezeigt, bei der Klasse *ccs_viz_manager* registriert. Da in diesem Fall aber ein *ccs_viz_task*-Objekt im *ccs_viz_process*-Objekt registriert ist, wird dem Timer der Klasse *ccs_viz_manager* dieses *ccs_viz_task*-Objekt übergeben. Dieser Timer wird gestartet, um das *ccs_viz_task*-Objekt nach einer gewissen Zeit auszuführen, wie in Abbildung 3.5 zu sehen ist.

Der Timer für die *ccs_viz_task*-Objekte der Wurzel-Prozesse muss neu gesetzt werden, solange mit Unterprozessen des Wurzel-Prozesses gearbeitet wird, damit diese arbeitenden Unterprozesse nicht gelöscht werden. Dies ist im zweiten Ablauf von Abbildung 3.5 zu erkennen. Sobald ein *ccs_viz_process*-Objekt angefragt wird um Informationen auszugeben oder aufzunehmen, wird dieses rekursiv alle Vater-Prozess Objekte des Baumes durchgehen, bis der Wurzel-Prozess erreicht ist. Dieser Prozess wird angewiesen den Timer der Klasse *ccs_viz_manager* für sein *ccs_viz_timer*-Objekt neu zu starten, was dann von der Klasse *ccs_viz_manager* ausgeführt wird.

Im dritten Ablauf der Abbildung 3.5 wird aufgezeigt, was geschieht, wenn der Timer für ein *ccs_viz_task*-Objekt abgelaufen ist. Nach dem Ablaufen des Timers eines *ccs_viz_task*-Objekts wird dieses ausgeführt. Es weist nun den Wurzel-Prozess, zu dem es eine Referenz gespeichert hat, an, rekursiv eine Liste aller Datenbank-Tabellen seiner Kind-Prozesse zu erstellen und sich und die Kind-Prozess zu löschen.

Mit der zurückgegebenen Liste der Tabellen und der Datenbank- Zugangsinforma-

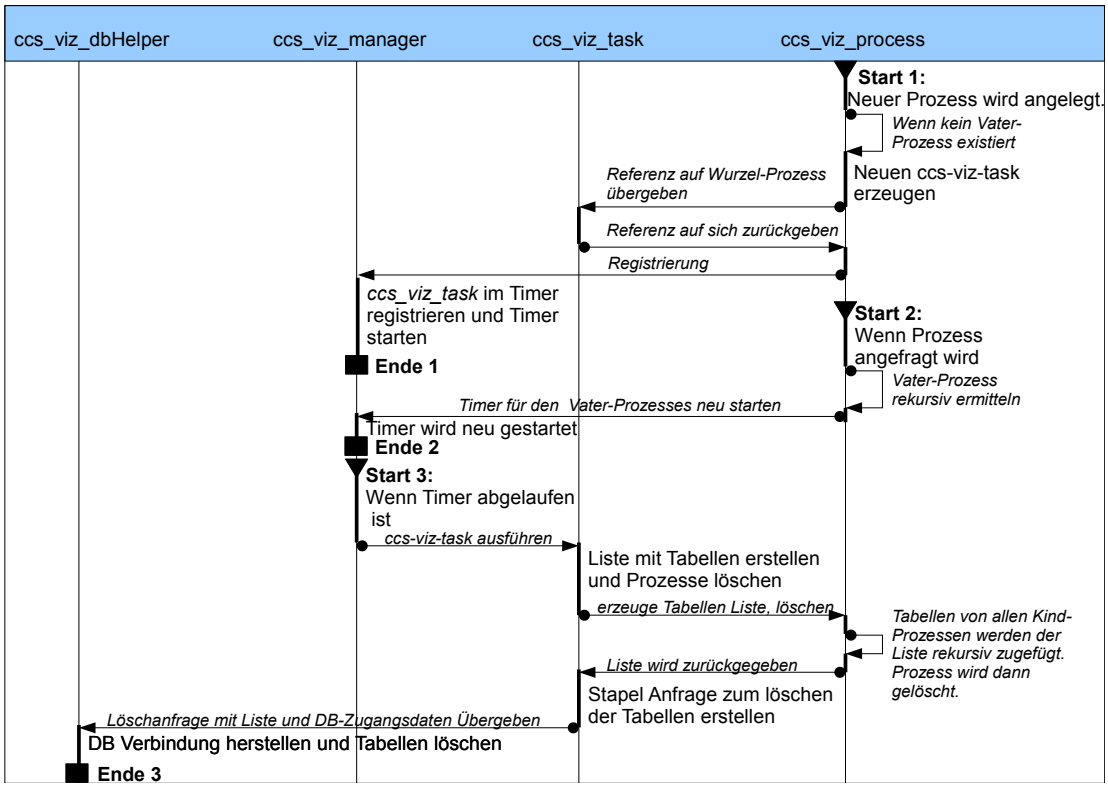

Abbildung 3.5: Ablauf der Aufräumroutine

tionen des Wurzel-Prozesses wird nun die Klasse *ccs_viz_dbHelper* angewiesen, eine Stapel-Anweisung zu generieren, die alle Tabellen der erhaltenen Liste löschen soll.

Mit Hilfe der Datenbank-Zugangsinformationen stellt die Klasse *ccs_viz_dbHelper* eine Verbindung zur entsprechenden Datenbank her und führt die Stapel-Anweisung über diese einzige Verbindung aus. Damit sind nun alle Prozesse, die sich unterhalb des Wurzel-Prozesses befanden gelöscht und auch aus der Datenbank entfernt.

3.6 Frontend

Das Frontend, also die dem Nutzer zugängliche Seite des Projektes, ist das Web-Interface. Hier werden dem Nutzer im Web-Browser alle Daten präsentiert und die Möglichkeit der Kommunikation zur eigentlichen Anwendung auf dem Server bereitgestellt.

Die Arbeit mit der Applikation soll für den Nutzer möglichst komfortabel und instinktiv sein, weswegen sich für das Web-Interface JavaScript und Ajax anbieten. So wird es dem Nutzer ermöglicht mit dem Visualisierungs-Interface ähnlich zu arbeiten, wie er es von Desktop-Anwendungen gewohnt ist. Das Interface reagiert direkt auf seine Eingaben und Anforderungen, ohne dass im Browser die HTML-Seite neu geladen werden muss.

Um diesen Komfort zu ermöglichen, muss der Web-Browser JavaScript unterstützen und vor allem aktiviert haben. Da dies aber nicht immer der Fall ist, wird neben dem benutzerfreundlichen Interface auf JavaScript Basis auch ein einfaches Interface zur Verfügung gestellt, welches ausschließlich HTML nutzt. Damit ist für alle Nutzer ein Zugang zu den visualisierten Prozessdaten sichergestellt.

Im Folgenden werden das JavaScript-, sowie das HTML-Interface genauer beschrieben.

3.6.1 JavaScript-Interface

Das Web-Interface auf JavaScript Basis ist in Abbildung 1.2 dargestellt. Es ist zu erkennen, dass dem Nutzer mehrere Fenster innerhalb des Browsers zur Verfügung stehen. Diese werden im Folgenden genauer beschrieben.

Prozessauswahl

Zum Einstieg wird nur ein Fenster zur Prozessauswahl angezeigt, in Abbildung 3.6 dargestellt.

Hier werden alle im Visualisierungs-Werkzeug registrierten Prozesse mit ihrer Identifikationsnummer und ihrem Typ angezeigt, wobei sie nach ihrer Position im Prozessbaum gruppiert sind.

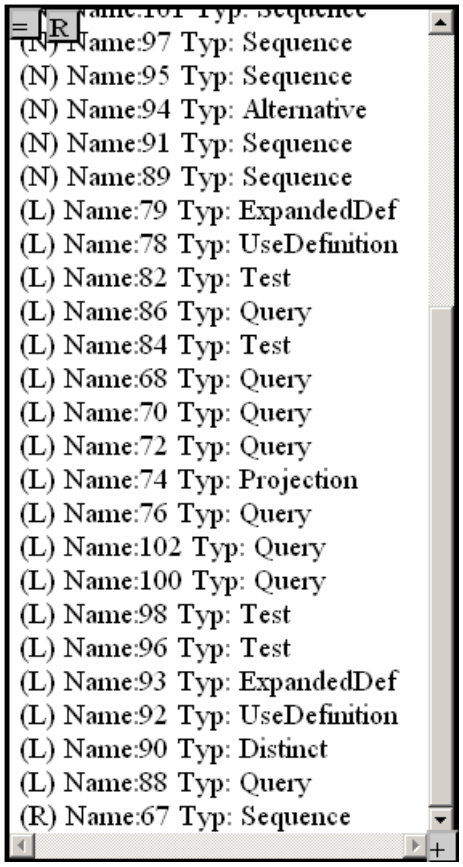

Abbildung 3.6: Darstellung des Prozessauswahl Fensters des JavaScript-Interfaces

Dabei sind Prozesse denen „(R)“ vorangestellt ist, Wurzel-Prozesse (im Englischen „Root-Processes“), also Prozesse die keinen Vater-Prozess besitzen. Prozesse, die mit „(N)“ als Präfix versehen sind, sind Knoten (englisch „Nodes“) im Prozessbaum. Sie haben sowohl einen Vater-Prozess, als auch mindestens einen Kind-Prozess. Bei Prozessen, die mit „(L)“ eingeleitet werden, handelt es sich um Blätter (im Englischen

„Leaves“) im Prozessbaum, also Prozesse, die einen Vater Prozess aber keine Kind-Prozesse besitzen.

Prozessbaum

Wenn ein Prozess aus der Auswahlliste durch anklicken ausgewählt wird, so wird ein neues Fenster erzeugt, in dem ein Prozessbaum dargestellt wird (siehe Abbildung 3.7). Dieser Baum hat den ausgewählten Prozess, in Abbildung 3.7 Prozess 341, als Wurzel und zeigt alle seine Kinder bis in eine bestimmte Tiefe an. In der Darstellung wird oberhalb der Wurzel noch der Vater-Prozess, Prozess 339 aus Abbildung 3.7, angezeigt, falls dieser vorhanden ist.

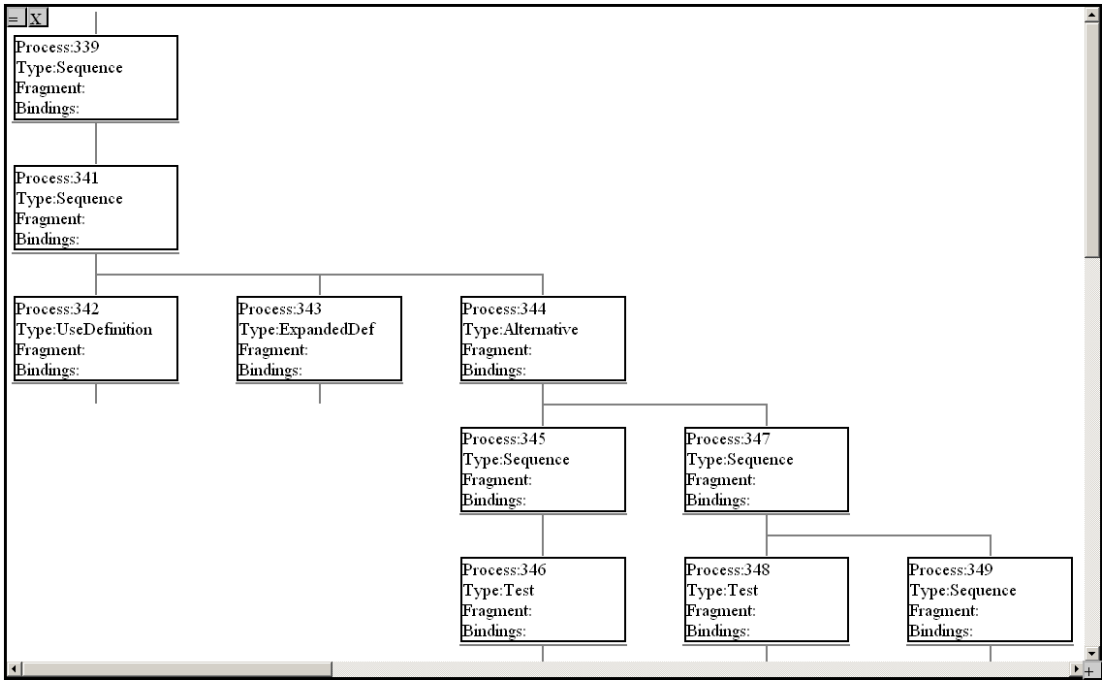

Abbildung 3.7: Darstellung eines Prozessbaums im JavaScript-Interface.

Dies ist wichtig, damit sich der Nutzer durch den Baum bewegen kann und zwar

sowohl auf- als auch absteigend. Indem er auf die Identifikation eines angezeigten Prozesses klickt, zum Beispiel auf „Process:345“,wird dieser als neue Wurzel ausgewählt und ein neuer Prozessbaum im selben Fenster angezeigt.

Um also aufsteigend zu navigieren, kann auf den vor der Wurzel angezeigten Prozess geklickt werden, also auf „Process:339“, so dass der Baum eine Ebene höher angezeigt wird. Um sich absteigend zu bewegen, muss nur auf einen Prozess unterhalb der Wurzel geklickt werden, zum Beispiel auf „Process:349“, und der Baum wird entsprechend viele Ebenen tiefer dargestellt.

Um ein weiteres Fenster mit einem Prozessbaum zu öffnen, muss lediglich ein Prozess aus der Auswahlliste in Abbildung 3.6 angeklickt werden. Die Fenster lassen sich unabhängig voneinander navigieren.

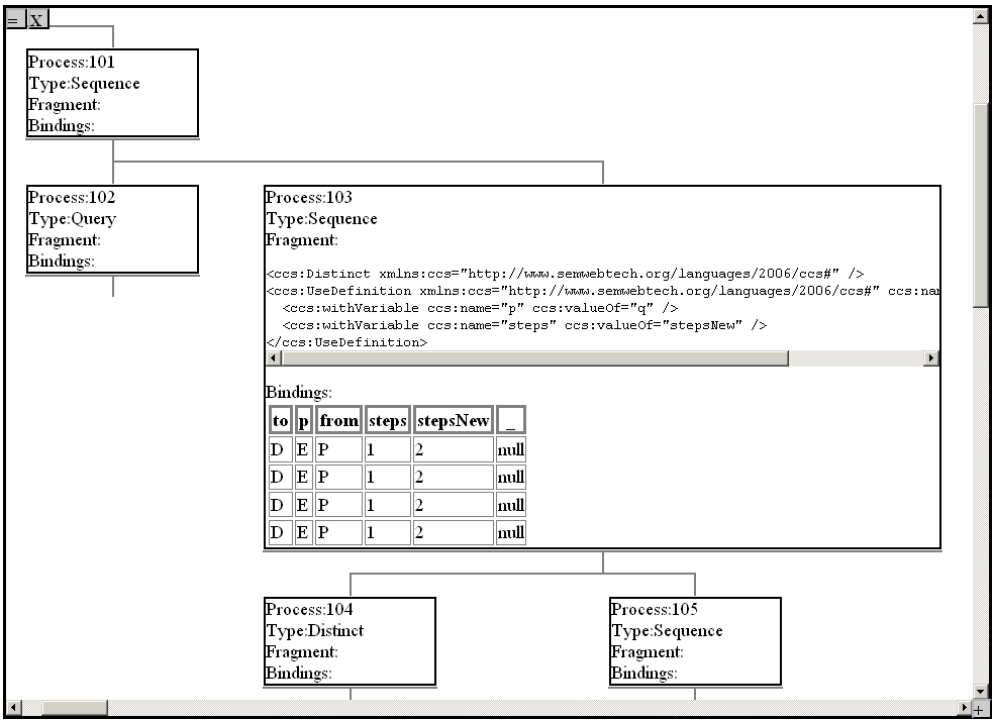

Abbildung 3.8: Darstellung der Detailansicht eines Prozesses im JavaScript-Interface.

Prozessdetails

Wenn der Nutzer an Details eines bestimmten Prozesses interessiert ist, kann er bei diesem „Fragment“ oder „Bindings“ anklicken, um den dargestellten Prozess zu vergrößern und die Informationen, die er wünscht, eingeblendet zu bekommen. In Abbildung 3.8 wird die Detailansicht von Prozess „Process:103“ gezeigt; dabei sind sowohl sein XML-Fragment, als auch die Variablenbindungen zu sehen.

Löschen von Prozessbäumen

Die Abbildung 3.9 zeigt eine Liste mit allen Wurzel-Prozessen. Hier kann durch Auswahl eines Prozesses, dieser und alle seine Kind-Prozess gelöscht werden. Durch Auswahl des Eintrags „Clear all processes“ können alle aktuell registrierten Prozesse gelöscht werden.

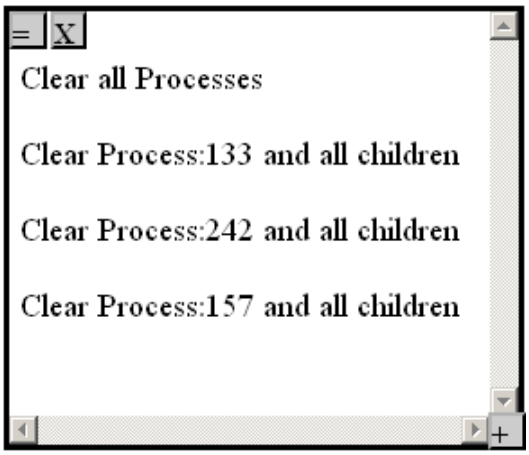

Abbildung 3.9: Darstellung der Liste zum Löschen von Prozessen im JavaScript-Interface.

Bedienung

Alle Fenster des JavaScript Web-Interfaces sind mit Hilfe des „=“-Symbols frei im Browser verschiebbar, sowie in ihrer Größe frei wählbar durch das „+“-Symbol.

Die Fenster überlagern sich und werden durch Anklicken in den Vordergrund geholt. Über das „X“-Symbol können sie geschlossen werden.

Eine Ausnahme bildet die Prozess-Auswahlliste aus Abbildung 3.6. Diese ist immer im Vordergrund und kann nicht geschlossen werden. Durch Anklicken des „R“-Symbols in der Prozessauswahlliste wird diese aktualisiert.

3.6.2 HTML-Interface

Das HTML-Interface des Web-Frontends kann genutzt werden, wenn der Browser des Nutzers entweder kein JavaScript unterstützt oder der Nutzer JavaScript deaktiviert hat. Wenn eine dieser Möglichkeiten eintritt und der Nutzer trotzdem versucht, das JavaScript-Interface aufzurufen, wird er darauf hingewiesen, dass JavaScript nicht verfügbar sei und er das HTML-Interface nutzen könne.

Prozessauswahl

Wenn das HTML-Interface aufgerufen wird, werden dem Nutzer alle aktiven und registrierten Prozesse, wie in Abbildung 3.10, als Liste gezeigt.

Die Liste ist in dieselben drei Gruppen gegliedert, wie sie schon im JavaScript-Interface verwendet und beschrieben wurden. Wurzel-Prozesse, markiert mit „(Root)“, Blatt-Prozesse, die mit „(Leaf)“ gekennzeichnet sind und Prozess-Knoten, welchen ein „(Node)“ vorangestellt ist.

Zu Beginn der Liste befindet sich eine Möglichkeit, Prozesse zu löschen. Durch Auswahl dieser Option wird eine Liste mit allen Wurzel-Prozessen erstellt und angezeigt.

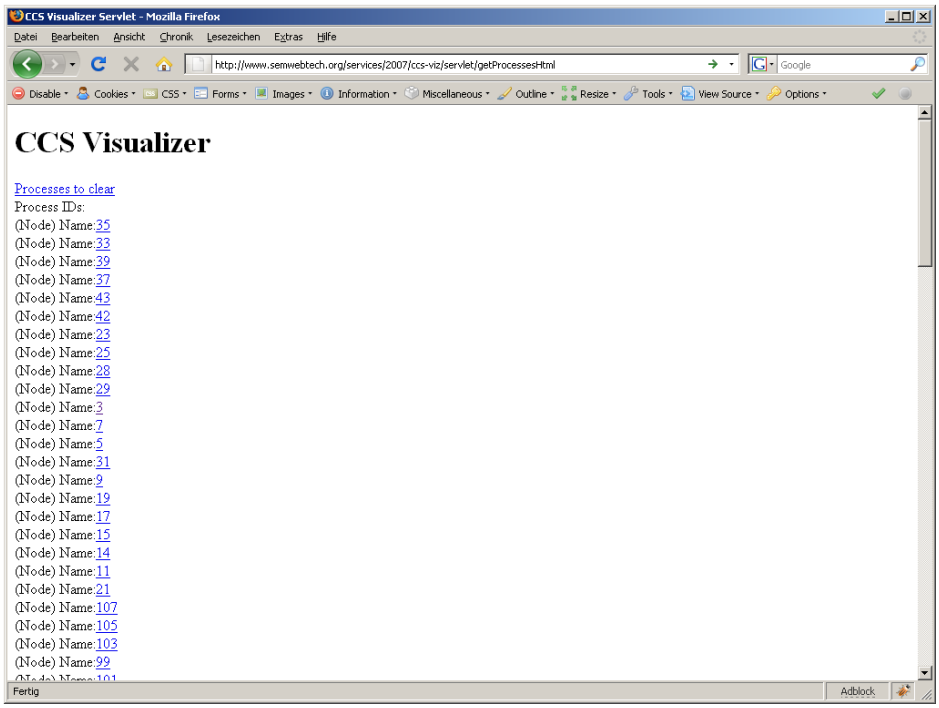

Abbildung 3.10: Die Prozessauswahl des HTML-Interfaces

Durch Anklicken eines dieser Wurzel-Prozesse wird der gesamte Baum dieser Wurzel gelöscht. Es können auch alle bestehenden Prozesse gelöscht werden.

Prozessbaum

Durch das Klicken auf einen der in Abbildung 3.10 angezeigten Prozesse wird der Prozessbaum mit dem ausgewählten Prozess als Wurzel generiert und, wie in Abbildung 3.11 dargestellt, ausgegeben. Der Baum ist im Prinzip gleich aufgebaut, wie der des JavaScript-Interfaces. Auch hier können die Details des Prozesses angezeigt werden, indem sie einfach vom Nutzer angeklickt werden.

Was den Komfort des HTML-Interfaces einschränkt, ist die Tatsache, dass jeder Klick auf einen Prozess oder Details zu einem Neuladen der Seite führt und so die

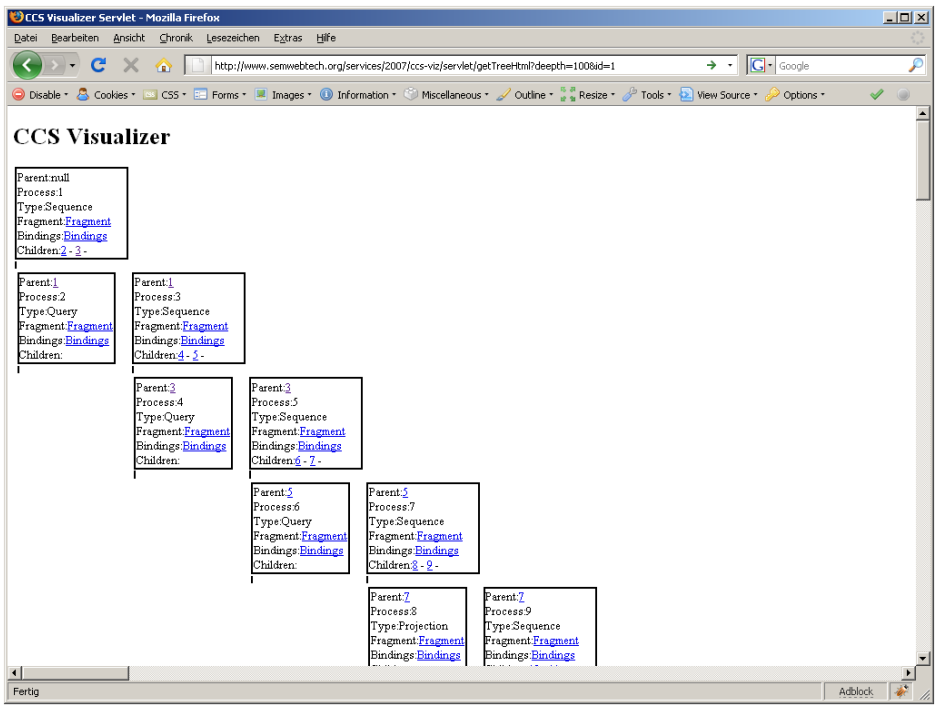

Abbildung 3.11: Anzeige eines Prozessbaums im HTML-Interface

vorherigen Informationen nicht mehr angezeigt werden. Dies erschwert dann den Vergleich von Prozessbäumen oder von Prozess-Details untereinander.

4 Implementierung

4.1 Werkzeuge

Für die Entwicklung dieses Projektes wurde die Integrierte Entwicklungsumgebung Eclipse Ganymede 3.14 [5] verwendet .

Bei der Entwicklung des Javascript-Interfaces wurde der Webbrowser Firefox Version 3.0.10 [10] mit der Erweiterung Firebug 1.3.3 [7] zur Fehlersuche und -behebung verwendet.

Als lokaler Webserver wurde ein Apache Tomcat Version 5.5 [1] genutzt.

4.2 Klassenbeschreibungen

Es folgt nun eine detaillierte Auflistung der Funktionen der in Kapitel 3.1 erwähnten Klassen, sowie eine Beschreibung ihrer Besonderheiten.

4.2.1 Die Klasse *ccs_viz_process*

Wie schon in Kapitel 3.1 erwähnt, stellt jedes Objekt dieser Klasse einen CCS Prozess dar und ermöglicht den Zugriff auf dessen Daten, sowie einige Funktionen. Wie in Abbildung 4.1 zu sehen ist, werden die meisten Daten, die den Prozess direkt betreffen, als Zeichenkette gespeichert. Informationen über die Verwandtschaft zu anderen

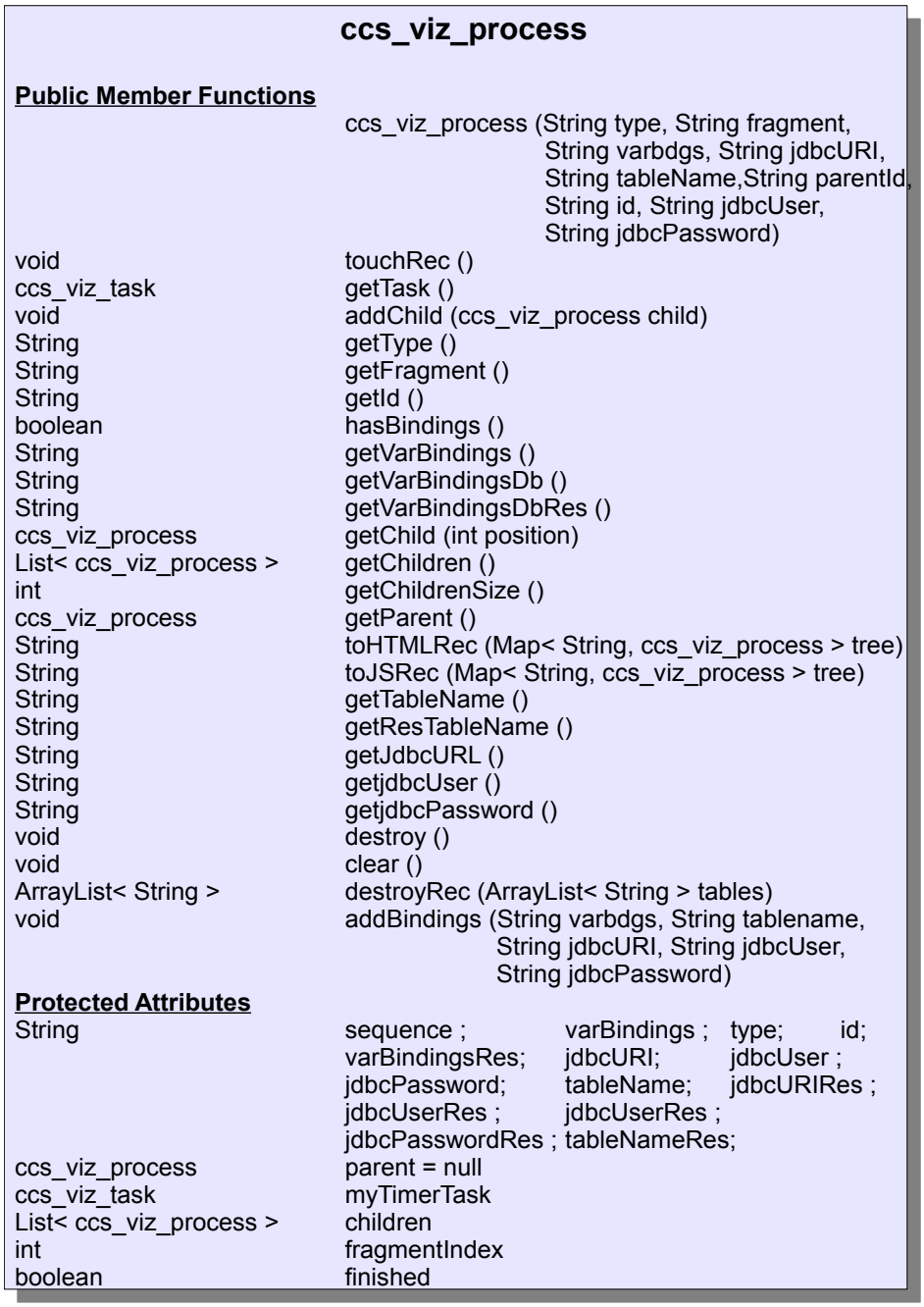

Abbildung 4.1: Klassendiagramm der Klasse ccs_viz_process

Objekten werden als direkte Referenz in „parent“, für den Vater-Prozess oder in einer Liste von Referenzen namens „children“ für die Kind-Prozesse, gespeichert.

Alle Eigenschaften einer Prozessinstanz sind von außen nur über die entsprechenden get-Methoden auszulesen, wie z.B. „getVarBindings()“, welche dann die Variablenbindungen als Zeichenkette zurück liefert. In Abbildung 4.1 ist die Auflistung aller dieser Funktionen zu sehen.

Diese Funktionen sorgen auch dafür, dass die gewünschten Daten gegebenenfalls erst noch aus der entsprechenden Datenbank, gemäß des in Kapitel 3.3 beschriebenen Ablaufs, gelesen werden.

Geschrieben werden die meisten Daten ausschließlich während der Erzeugung des Objektes. Dies geschieht nach dem in Kapitel 3.2 beschriebenen Schema.

Eine Ausnahme bildet die Funktion „addBindings(String varbdgs, String tablename, String jdbcURI, String jdbcUser, String jdbcPassword)“. Sie fügt die übergebenen Daten für die resultierenden Variablenbindungen dem Objekt hinzu.

Um die Funktionalität der in Kapitel 3.5 beschriebene Aufräumroutine zu ermöglichen, ist die Funktion „touchRec()“ entscheidend, da diese den Timer des Wurzelprozesses für die Löschung des Prozessbaums zurücksetzt. Außerdem ist die Funktion „destroy()“ wichtig, welche aufgerufen wird, um dieses Objekt samt Datenbank-Einträgen zu löschen. Für die in Kapitel 3.5 beschriebene rekursive Löschung ist die Funktion „destroyRec(ArrayList< String > tables)“ verantwortlich.

4.2.2 Die Klasse *ccs_viz_manager*

In Kapitel 3.1 wurde schon erwähnt, dass es sich bei der Klasse *ccs_viz_manager* um die Verwaltungseinheit des Projektes handelt. Dies bewerkstelligt diese statische Klasse, indem sie Referenzen auf alle *ccs_viz_process* Objekte in zwei Hash-Maps, „instances“ und „rootProcesses“ hält, zu sehen in Abbildung 4.2. Hier wird die Referenz auf das

Objekt, erreichbar unter seiner „Id“, bei dessen Erzeugung abgelegt, wie in Kapitel 3.2 beschrieben. Die Hash-Map „instances“ enthält alle erzeugten Prozesse, die Hash-Map „rootProcesses“ nur diejenigen, bei denen es sich um Wurzel-Prozesse handelt.

Die in Abbildung 4.2 zu sehende Funktion „addProcess(ccs_viz_process process)“ wird bei der Erzeugung eines *ccs_viz_process* Objektes aufgerufen und trägt dieses Objekt gemäß Kapitel 3.2 ein.

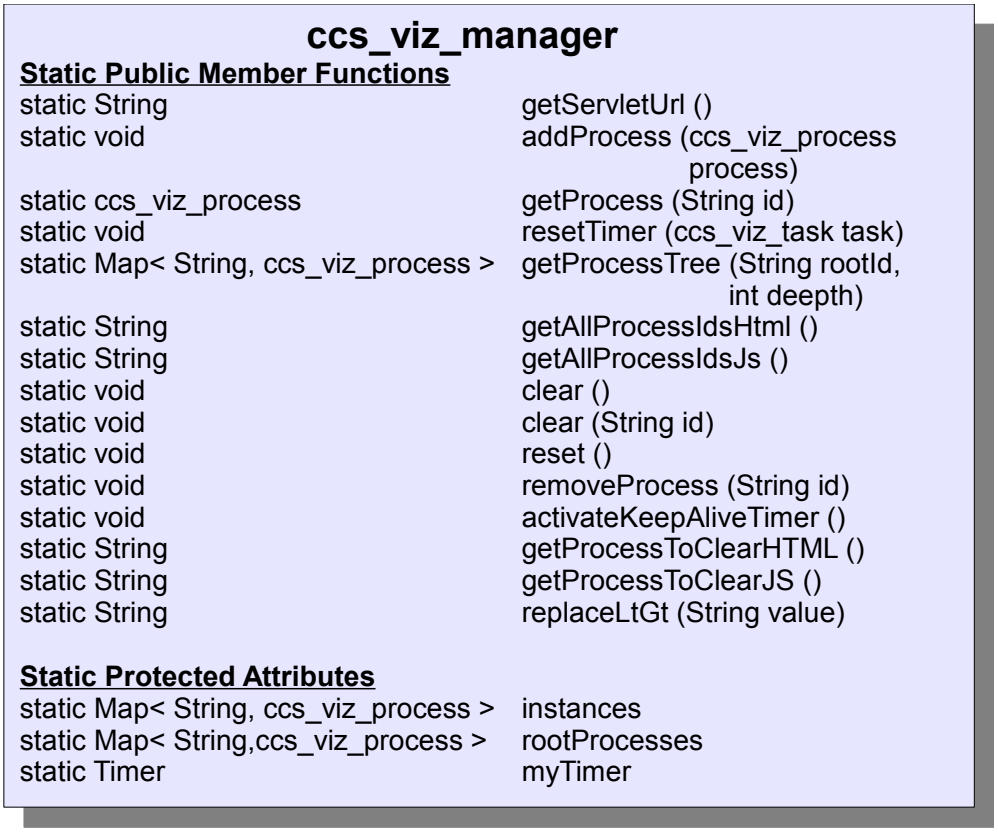

Abbildung 4.2: Klassendiagramm der Klasse ccs_viz_manager

Die Funktionen „getAllProcessIdsHtml()“ und „getAllProcessIdsJS()“ sorgen für die Erzeugung der Prozessliste, wie sie in Kapitel 3.4 Teil 1 beschrieben wurde und als Beispiel in Abbildung 3.10 zu sehen ist.

Diese Klasse beinhaltet auch den Timer, „myTimer“ in Abbildung 4.2, welcher für die Ausführung der in Kapitel 3.5 beschriebenen Aufräumroutine wichtig ist. Mit Hilfe der Funktion „resetTimer(ccs_viz_task task)“ wird er für die entsprechenden Wurzelprozesse gestartet.

Um Prozessbäume mit Hilfe der in Abbildung 1.2 zu sehenden Liste zu löschen, ist die Funktion „getProcessToClearJS()“ nötig um diese Liste zu erzeugen. Im HTML-Interface übernimmt dies die Funktion „getProcessToClearHTML()“.

Das Löschen eines Prozessbaums wird mit Hilfe der Funkion „clear (String id)“ bewerkstelligt. Hierbei wird der Prozessbaum des angegebenen Wurzel-Prozesses rekursiv gelöscht.

4.2.3 Die Klasse *ccs_viz_servlet*

Diese Klasse implementiert das in Kapitel 2.3.1 vorgestellte Java Servlet und ist damit der nach außen hin sichtbare Ansprechpunkt des CCS Visualisierungs-Werkzeugs. Alle Anfragen, die über das HTTP-Protokoll an den Visualisierer gestellt werden, werden von der Klasse *ccs_viz_servlet* entgegengenommen.

Von außen werden alle Anfragen an eine bestimmte Adresse vom Webserver an das Servlet weitergegeben. Wie in Abbildung 4.3 zu sehen wird durch den Aufruf einer der beiden Methoden „doGet(HttpServletRequest request, HttpServletResponse response)“ oder „doPost(HttpServletRequest request, HttpServletResponse response)“, die Auswertung, der vom Webserver im „HttpServletRequest“-Objekt, „request“, übergebenen Daten, begonnen.

Hier werden dann die entsprechenden Funktionen der Klasse *ccs_viz_manager* aufgerufen, um die angeforderten Daten zusammenzustellen, oder die gewünschten Aktionen auszuführen.

Mit Hilfe des „HttpServletResponse“ Objectes, „response“, werden die angeforder-

ten Daten an den Webserver zurück gegeben, der sie dann an den Browser des Nutzers sendet.

Die in Abbildung 4.3 zu sehende Funktion „getPostParameter(HttpServletRequest request)“ dient zur Aufbereitung der durch die Post-Übergabe gelieferten Daten. Sie wird von der Funktion „doPost(HttpServletRequest request, HttpServletResponse response)“ aufgerufen und macht es möglich, die Daten einfach zu verarbeiten.

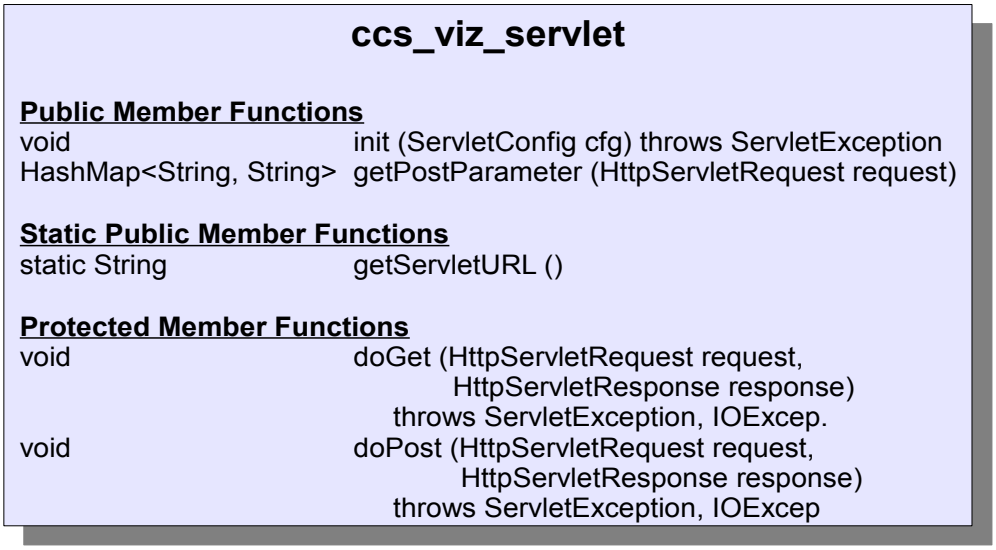

Abbildung 4.3: Klassendiagramm der Klasse ccs_viz_servlet

Die Funktion „getServletURL()“, gibt den „Uniform Resource Locator“ (URL), unter dem das Visualisierungswerkzeug über das HTTP Protokoll erreicht werden kann, zurück. Dies ist für das Frontend wichtig, damit die Anfragen des Nutzers von diesem an des richtige Servlet geschickt werden können.

4.2.4 Die Klasse *ccs_viz_dbHelper*

Bei der Klasse *ccs_viz_dbHelper*, dargestellt in Abbildung 4.4, handelt es sich um eine statische Hilfsklasse.

Sie stellt Funktionen zur Verfügung, die Datenbankoperationen vereinfachen. Andere Klassen greifen auf diese Funktionen zu, um die in Kapitel 3.3 beschriebenen Datenbankzugriffe zu bewerkstelligen.

Die Funktion „getBindings(String encodedJDBC, String tablename)“ in Abbildung 4.4 wird von Objekten der Klasse *ccs_viz_process* verwendet, um die Daten ihrer Variablenbindungen aus der Datenbank zu lesen.

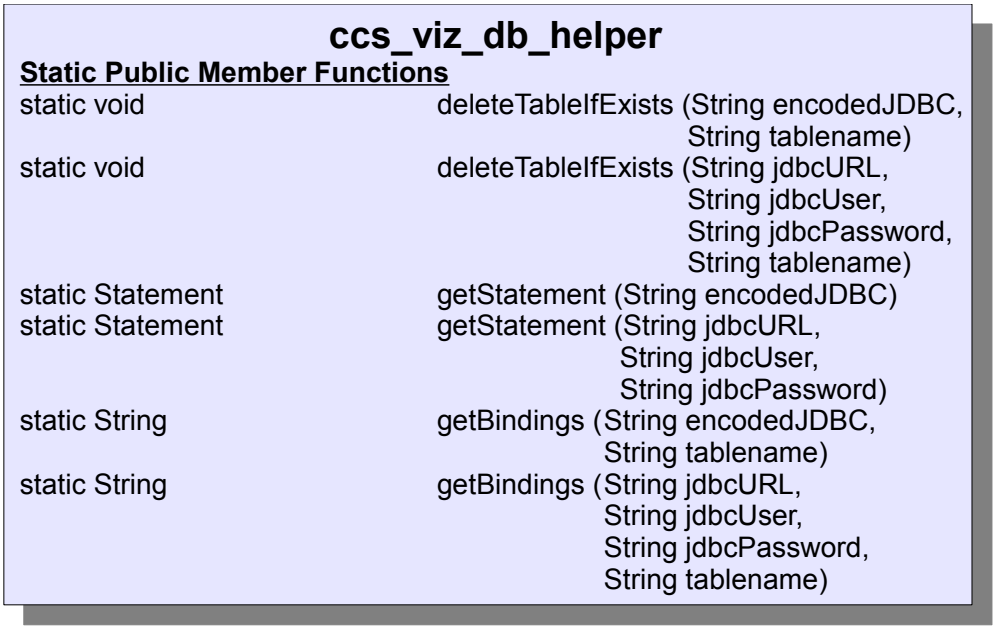

Abbildung 4.4: Klassendiagramm der Klasse ccs_viz_dbHelper

Die Funktion „deleteTableIfExists(String encodedJDBC, String tablename)“ aus Abbildung 4.4 wird während der Aufräumroutine aus Kapitel 3.5 genutzt, um die Datenbanktabellen, welche die Variablenbindungen der einzelnen Prozesse enthalten, zu löschen.

4.2.5 Die Klasse *ccs_viz_task*

Objekte dieser Klasse werden von Wurzel-Prozessen erzeugt und dem Timer „myTimer“ der Klasse *ccs_viz_manager*, siehe Abbildung 4.2, übergeben. Jedes *ccs_viz_task*-Objekt bekommt bei seiner Erzeugung eine Referenz auf den entsprechenden Wurzel-Prozess durch den Konstruktor „ccs_viz_task(ccs_viz_process process)“, zu sehen in Abbildung 4.5, übergeben.

Wenn die gesetzte Zeit des Timers für das Objekt abgelaufen ist, wird die Methode „run()“, aus Abbildung 4.5, aufgerufen. Diese initiiert die Löschung des Wurzel-Prozesses, welcher durch die Referenz „process“ im Konstruktor übergeben wurde, sowie all seiner Kind-Prozesse.

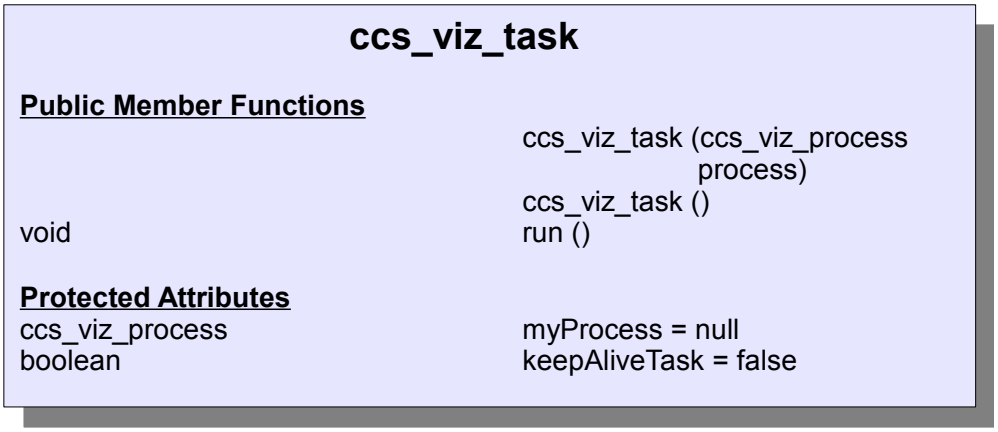

Abbildung 4.5: Klassendiagramm der Klasse ccs_viz_task

Der Standard-Konstruktor „ccs_viz_task()“ und die Variable „keepAliveTask“ werden benötigt, um bei der Initialisierung des Visualisierungs-Werkzeugs ein Objekt im Timer des *ccs_viz_managers* zu starten. Dieses sorgt dafür, dass der Timer sich nicht selbst beendet, wenn aktuell keine Prozesse vorliegen.

4.2.6 Die Klasse *ccs_viz_plugin*

Wie in Kapitel 3.1 beschrieben, stellt diese Klasse die Verbindung zwischen dem MARS-Framework und dem Visualisierungswerkzeug her. Diese Verbindung wird über das HTTP-Protokoll realisiert.

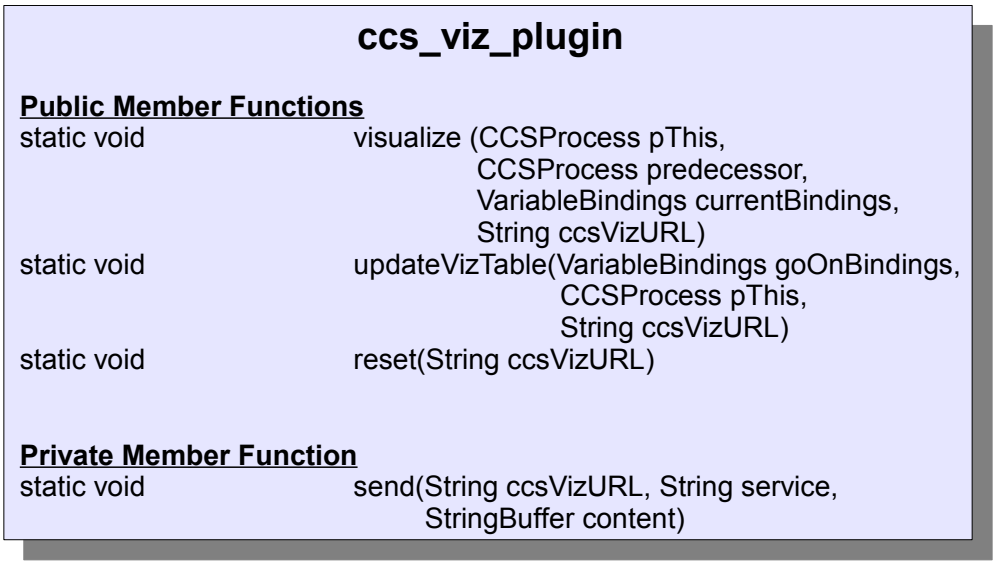

Abbildung 4.6: Klassendiagramm der Klasse ccs_viz_plugin

Die zu übertragenden Daten werden mit Hilfe der Funktion „visualize(CCSProcess pThis, CCSProcess predecessor, VariableBindings currentBindings, String ccsVizURL)“ oder „updateVizTable(VariableBindings goOnBindings, CCSProcess pThis, String ccsVizURL)“, zu sehen in Abbildung 4.6, an die Klasse *ccs_viz_plugin* übergeben.

Der Übergabeparameter „pThis“ ist eine Referenz auf den zu übergebenden CCS Prozess. Mit Hilfe dieser Referenz wird die notwendige Kopie der Datenbank-Tabelle des Prozesses angelegt und dann alle notwendigen Daten für die Übergabe zusammengestellt.

Wenn alle Daten vorhanden sind, werden sie in eine Post-Übergaben-konforme-

Form gebracht und mit Hilfe der privaten Funktion „send(String ccsVizURL, String service, StringBuffer content)“ über das HTTP Protokoll an das *ccs_viz_servlet* gesendet.

4.3 Implementierung des JavaScript-Interfaces

Das JavaScript-Interface bekommt die zu zeigenden Daten in einer einfachen, durch Kommata getrennten Formatierung übergeben und erstellt daraus selbständig den in Abbildung 3.7 zu sehenden Prozessbaum, oder die Prozessliste, aus Abbildung 3.6. Die Neuzeichnung des Baumes, wenn Prozess-Knoten aufgeklappt werden, wird ebenfalls komplett vom JavaScript-Interface übernommen. Zusätzlich wird die Darstellung und Kontrolle der Fenster von ihm übernommen.

Einmal übertragene Daten werden im JavaScript-Interface gespeichert, solange der entsprechende Baum dargestellt wird. Durch diese Maßnahmen bleibt der Server von den aufwändigen Grafikdarstellungen und Berechnungen unberührt und die Datenübertragungen zwischen dem Web Browser des Nutzers und dem Server werden so gering wie möglich gehalten.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Dieses Projekt ist als kompaktes Paket entwickelt, welches sich sehr einfach in eine bestehende Version des MARS-Frameworks einfügen lässt, da es lediglich in den Webserver als Erweiterung eingebunden werden muss und durch die Plugin-Klasse nur minimale Eingriffe im Quellcode des Frameworks nötig sind.

Durch die Konzipierung als Web Service kann das Visualisierungs-Werkzeug auch auf einem ausgelagerten Webserver installiert sein, um so die Ressourcen des Servers, auf dem das Framework läuft, nicht zu beeinträchtigen. Darüber hinaus können verschiedene Frameworks den gleichen Visualisierer nutzen.

Die Bereitstellung des auf JavaScript basierten Frontends gestattet den Nutzern ein komfortables Arbeiten. Durch die Existenz des HTML-Frontends ist eine maximale Kompatibilität mit allen gängigen Web-Browsern gewährleistet.

5.2 Ausblick

Das Werkzeug wird durch seine einfache und komfortable Bedienbarkeit und übersichtliche Darstellung von CCS-Prozessen die Entwicklung solcher für das MARS-Framework sehr erleichtern.

Es ist jetzt möglich, einen schnellen Überblick über das Verhalten von laufenden Pro-

zessen zu erhalten. So kann während der Entwicklung von neuen Prozessen schnell ein Fehler im Konzept des neuen Prozesses erkannt werden.

Außerdem ist durch den schnellen Zugriff auf die Daten eines Prozesses die Lokalisierung von Fehlern einfacher möglich, da der Prozessknoten mit fehlerhaften Daten schnell ausgemacht werden kann und die Quelle des Fehlers so schnell aufgespürt und behoben werden kann.

Durch den gleichzeitigen Vergleich von verschieden Prozessbäumen kann der parallele Ablauf von verschiedenen Prozessen beobachtet werden und so das Zusammenspiel von Prozessen einfach überwacht werden.

Mit Hilfes dieses Projektes wird das Entwickeln neuer CCS-Prozesse im MARS-Framework durch die komfortable Möglichkeit der Fehlerauffindung sehr vereinfacht.

Literaturverzeichnis

[1] Apache Software Foundation. Tomcat 5.5. http://tomcat.apache.org/download-55.cgi, 2008.

[2] Bruce W. Perry. *Java Servlet & JSP Cookbook*. O'Reilly Media, Inc., 2004.

[3] Dave Raggett, Arnaud Le Hors, Ian Jacobs. HTML 4.01 Specification. http://www.w3.org/TR/REC-html40/intro/intro.html#h-2.2, 1999.

[4] Dave Raggett, Arnaud Le Hors, Ian Jacobs. HTML 4.01 Specification. http://www.w3.org/TR/REC-html40/interact/scripts.html, 1999.

[5] Eclipse contributors and others. Eclipse Platform Version: 3.4.1. http://www.eclipse.org/platform, 2008.

[6] Erik Behrends, Oliver Fritzen, Wolfgang May, Franz Schenk. MARS: Modular Active Rules in the Semantic Web. http://www.dbis.informatik.uni-goettingen.de/Publics/general/framework-talk.pdf, 2007.

[7] Joe Hewitt, Rob Campbell. Firebug 1.3.3. http://getfirebug.com/, 2008.

[8] Maydene Fisher, Jon Ellis, Jonathan Bruce. *JDBC API Tutorial and Reference, Third Edition*. Prentice Hall, 2003.

[9] Mozilla Developer Center. Ajax. https://developer.mozilla.org/en/AJAX, 2009.

[10] Mozilla Europe und Mozilla Foundation. Firefox Version 3.0.10. http://www.mozilla-europe.org/de/firefox/3.0.1/releasenotes/, 2008.

[11] Robin Milner. *Calculi for synchrony and asynchrony*. Theoretical Computer Science, 1983.

[12] Shelley Powers. *Adding Ajax*. O'Reilly Media, Inc., 2007.

[13] Sun Microsystems. Java Servlet Technology Overview. http://java.sun.com/products/servlet/overview.html, 2009.

[14] Sun Microsystems. JavaServer Pages Overview. http://java.sun.com/products/jsp/overview.html, 2009.

[15] Sun Microsystems. JDBC Overview. http://java.sun.com/products/jdbc/overview.html, 2009.

[16] Thomas Hornung, Wolfgang May, Georg Lausen. *Process Algebra-Based Query Workflows*. Springer LNCS 5565, 2009.

[17] W3C Communications Team. XML in 10 points. http://www.w3c.de/Misc/XML-in-10-points.html, 2003.

Abbildungsverzeichnis

Abkürzungsverzeichnis

AJAX	Asynchronous JavaScript and XML
CCS	Calculus of Communicating Systems
CGI	Common Gateway Interface
CSS	Cascading Style Sheet
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
Id	Identifikation
JDBC	Java Database Connectivity
JSP	Java Server Pages
MARS	Framework Modular Active Rules for the Semantic Web
TCP/IP	Transmission Control Protocol/Internet Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language