Diplomarbeit

Erweiterung einer Webanwendung um eine Plugin-Schnittstelle am Beispiel des CommSy

vorgelegt von

Johannes Schultze

Marthastr. 43a

20259 Hamburg

9schultz@informatik.uni-hamburg.de

Matr.-Nr.: 5202306

am

Arbeitsbereich Softwaretechnik

Fachbereich Informatik

Universität Hamburg

Hamburg, den 23. Februar 2007

Erstbetreuer

Dr. Wolf-Gideon Bleek

Arbeitsbereich Softwaretechnik
Fachbereich Informatik
Universität Hamburg
Vogt-Koelln-Str. 30
22527 Hamburg

Zweitbetreuer

Dr. Axel Schmolitzky

Arbeitsbereich Softwaretechnik
Fachbereich Informatik
Universität Hamburg
Vogt-Koelln-Str. 30
22527 Hamburg

Danksagung

Während des Schreibens dieser Arbeit habe ich von Familie und Freunden viel Unterstützung erhalten. Der größte Teil dieser Unterstützung hatte mit der Arbeit nicht im Geringsten zu tun – war aber während dieser Zeit umso wichtiger. Ihnen allen gilt der größte Dank an dieser Stelle.

Bei meinen beiden Betreuern möchte ich mich für die Bereitschaft bedanken, diese Arbeit zu begleiten. Vor allem danke ich Wolf-Gideon für seine hilfreichen, viel Arbeit nach sich ziehenden aber immer motivierenden Ratschläge.

Vielen aus diesem Kreis gilt außerdem der Dank für sehr schnelles Korrekturlesen vieler Entwürfe und das Streichen unzähliger überflüssiger Kommata. Diese Danksagung ging nur durch meine Hände – die Fehler, die ich dabei gefunden habe, behalte ich für mich.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Kontext

Grundlage dieser Arbeit ist die Migration des Online-Systems *CommSy* (siehe [4]) von der Scriptsprache PHP in die Programmiersprache Java. Im Projekt „WebMig‘‘(Uni Hamburg, Fachbereich Informatik, Arbeitsgruppe SWT, WiSe 2004/05 und SoSe 2005) wurde dieser Prozess mit der vertikalen Migration der Rubrik „Termine‘‘(siehe [5]) begonnen. Seit dem Ende des Projektes wird die Entwicklung des neuen Java-CommSy (im folgenden *JCommSy*) mit studentischen Hilfskräften weitergeführt. Mittlerweile wird sie mit der laufenden PHP-CommSy Entwicklung koordiniert und in ersten Schritten zusammengeführt.

Namenskonvention

In dieser Arbeit wird der Begriff „CommSy‘‘für nichttechnische Konzepte der Benutzung (Rubriken, etc.) sowohl im PHP- als auch im JCommSy benutzt. Architekturbeschreibungen und Darstellungen von Quellcode beziehen sich auf das vorhandene bzw. während dieser Arbeit weiterentwickelte JCommSy.

1.1.1 CommSy

Das CommSy ist ein Online-System, welches die Zusammenarbeit von Mitgliedern einer Gruppe unterstützen soll. Eine solche Gruppe kann ein Universitätskurs oder eine Schulklasse, aber auch eine Projekt- bzw. Arbeitsgruppe in einer Firma sein. In einer CommSy-Installation kann für eine solche Gruppe ein Projektraum angelegt werden (für die verschiedenen Raum-Typen des CommSy siehe [5]). Innerhalb eines Projektraumes können die Teilnehmer der Gruppe Termine koordinieren, Materialien hinterlegen, Diskussionen führen oder auch Ankündigungen zu aktuellen Themen angelegen (für eine vollständige Liste der Funktionen siehe ebenfalls [5]). Das CommSy entstand 1999 als Forschungsprojekt an der Uni Hamburg. Die Entwicklung des im Benutzerbetrieb laufenden Systems erfolgt seitdem in PHP. Zum Zeitpunkt dieser Arbeit ist die CommSy-Version 5.1.0 aktuell.

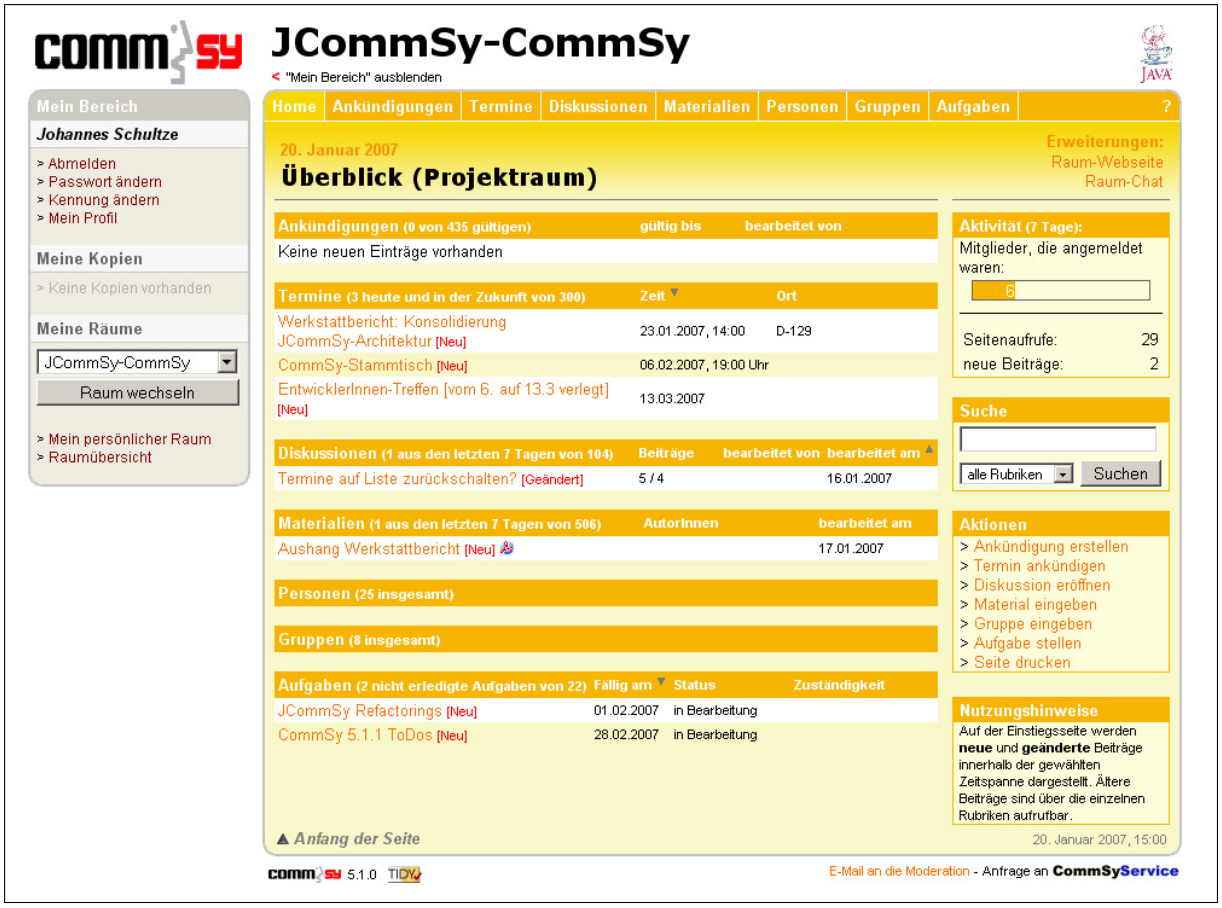


Abbildung 1.1: Ansicht eines CommSy-Projektraums

1.1.2 „WebMig‘‘-Projekt und Umstellung auf Java

In dem Projekt „WebMig‘‘ wurde nach Möglichkeiten einer Migration des CommSy von PHP nach Java gesucht. Nach verschiedenen Prototypen existieren jetzt eine weitestgehend fertig gestellte javabasierte Rubrik „Termine‘‘ sowie die Unterstützung für „Anmerkungen‘‘ . Die Implementierung der Rubriken „Ankündigungen‘‘ und „Diskussionen‘‘ hat begonnen. Diese Java-Rubriken können parallel zum PHP-System benutzt werden. Dazu wird in den Konfigurationsdateien des PHP- und des JCommSy definiert, welche Java-Rubriken genutzt werden sollen. Wird vom PHP-System aus eine solche Java-Rubrik aufgerufen, erfolgt automatisch eine Weiterleitung in das Java-System. Aufrufe an PHP-Rubriken aus dem Java-System werden auf die gleiche Weise zurück an das PHP-System geleitet. Ziel ist das rubrikweise Ablösen des PHP-Systems durch die neuen Java-Rubriken, ohne dass dies von den Benutzern bemerkt wird oder deren Arbeit mit dem CommSy beeinträchtigt. Ziel der Migration ist weiterhin die Erhaltung der Benutzungsschnittstelle des Systems, um das PHP-CommSy ohne Veränderungen der Handhabung abzulösen.

1.2 Motivation der Arbeit

Das PHP-CommSy und das JCommSy können bisher nur mit großem Aufwand um neue Rubriken erweitert werden.

Im JCommSy gibt es keine klaren Schnittstellen, an denen die Rubriken ansetzen, da die vorhandene Architektur keine Trennung zwischen einem in jeder CommSy-Installation benötigten Kernsystem und den erweiternden Rubriken vorsieht. Erweiterungen um Rubriken können nur von CommSy-Entwicklern durchgeführt werden, die sich mit dem Gesamtsystem auskennen. Bestehende Rubriken des CommSy können ebenfalls nur von CommSy-Entwicklern an spezielle Anforderungen angepasst werden - sofern diese gewünschten Anpassungen überhaupt möglich sind. Die Rubriken sind so stark in das Gesamtsystem integriert, dass sie nicht alleine als Erweiterungen ausgeliefert und nachträglich in ein bestehendes CommSy-System integriert werden können.

Manche Rubriken sind in ihrer bestehenden Form nicht ausreichend an die Bedürfnisse einzelner Gruppen angepasst. In einem Softwareentwicklungsprojekt wäre etwa eine Art XPlanner[1]-Rubrik wünschenswert. Für den Einsatz in Schulen könnte z.B. in einer eigenen Rubrik jede/r Schüler/in seinen/ihren eigenen Stundenplan verwalten. Diese Projekte benötigen also Erweiterungen, Veränderungen oder Neuentwicklungen von Rubriken. Neben der normalen Wartung, Fehlerbehebung und planmäßigen Weiterentwicklung des bestehenden CommSy würden diese Neuentwicklungen jedoch einen zu großen Aufwand bedeuten, wenn sie nur von einzelnen Projekträumen genutzt werden.

1.3 Erweiterbarkeit durch eine Plugin-Schnittstelle

Das JCommSy-System soll mit einer Plugin-Schnittstelle versehen werden, um die Möglichkeit einer leichten Erweiterbarkeit des Systems zu schaffen. Das heißt, dass entweder das gesamte System mit einer Plugin-Architektur reimplementiert oder, wie hier durchgeführt, das bestehende System um eine Plugin-Schnittstelle erweitert werden muss.

Die Plugin-Schnittstelle soll es ermöglichen, die Rubriken stärker vom Kernsystem zu lösen und somit die Entwicklung, Weiterentwicklung und Wartung dieser Rubriken unabhängiger ausführen zu können. Es kann dann auch möglich sein, die Rubriken einzeln auszuliefern und in einem bereits installierten und laufenden CommSy zusätzlich einzusetzen. Ein wünschenswerter weiterer Schritt ist dann eine Rubrik, mit der sich die anderen Rubriken installieren, verwalten oder wieder entfernen lassen.

Die Entwicklung der Rubriken, die als Plugins in das System eingehängt werden sollen, soll so einfach werden, dass kein Einblick in die internen Abläufe des Kernsystems notwendig ist. Dadurch soll es auch Programmierern, die nicht aktiv am CommSy-Quellcode entwickeln und die die Architektur und den Quellcode des Kernsystems nicht kennen, ermöglicht werden, eigene Plugins bzw. Rubriken zu entwickeln.

[1] XPlanner - *http://www.xplanner.org/*

Kapitel 2

JCommSy

2.1 Architektur des Systems vor der Erweiterung um eine Plugin-Schnittstelle

In diesem Kapitel soll die Architektur des JCommSy vor der Integration der Plugin-Schnittstelle beschrieben werden. Diese Architektur ist aus dem Projekt „WebMig“ hervorgegangen und stellt einen Zustand des Systems dar, der nach der vorhergehenden prototypischen Projektphase für den späteren Benutzerbetrieb implementiert worden ist.

Die Architektur entspricht grob einer Schichtenarchitektur, allerdings mit gewissen Einschränkungen. Eine Schichtenarchitektur ist *strikt*, wenn Zugriffe von einer Schicht nur auf die direkt darunter liegende Schicht erlaubt sind. Im Falle des JCommSy handelt es sich jedoch um eine *nicht strikte* Schichtenarchitektur, d.h. Zugriffe einer Schicht auf mehrere darunter liegende Schichten sind vorhanden. Zugriffe von Schichten auf weiter oben liegende Schichten sind in Schichtenarchitekturen generell nicht erlaubt. Das JCommSy verletzt diese Regel durch den Zugriff der Steuerungsschicht auf die Darstellungsschicht. Der Zugriff wird jedoch nur für die Weitergabe von Daten an die Beans in der Darstellungsschicht gebraucht. Für eine genaue Diskussion der Architektur siehe [17]. Die Schichten des JCommSy sowie die Zugriffe der Schichte untereinander sind in Abb 2.1 zusammengefasst.

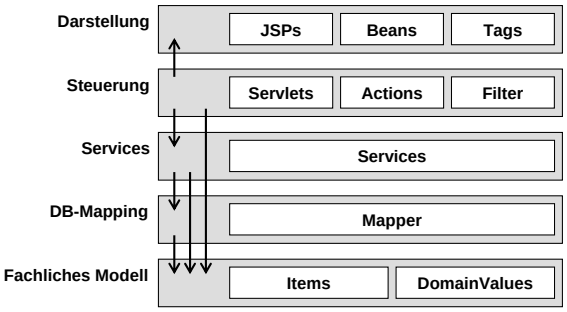


Abbildung 2.1: Schichten des JCommSy-Systems vor der Integration der Plugin-Schnittstelle. Die Pfeile stehen für die Zugriffe von einer Schicht auf eine andere Schicht.

2.1.1 Darstellungsschicht

Die Darstellungsschicht ist für die Erzeugung der HTML-Seiten und somit für die Präsentation der Inhalte zuständig. Die Bestandteile (JSPs, Beans und Tags) werden im Folgenden beschrieben.

Java Server Pages und Tags

Die Erzeugung des HTML-Codes, der mit dem Response an den Browser zurückgegeben wird, erfolgt über *Java Server Pages* (kurz: JSPs). JSP-Dateien ermöglichen die dynamische Erstellung von HTML-Seiten auf dem Server. Die Programmierung erfolgt dabei in Java. Ein Teil der JSPs ist im JCommSy nur für die Erzeugung des Rahmens einer JCommSy-Seite zuständig. Zu diesem Rahmen gehört z.B. die Auswahl des aktuellen CommSy-Raums, sowie die Darstellung der Karteireiter-Optik. Zusätzlich JSPs sind für die Darstellung der Informationen der angeforderten Rubrik zuständig. Solche JSPs gibt es für jede Rubrik im JCommSy. Rahmen- und Inhalt-JPSs werden zu einer Gesamtseite zusammmengesetzt.

Zusätzlich zur Programmierung in Java und den statischen HTML-Tags, die in einer JSP verwendet werden können, gibt es außerdem die Möglichkeit, *Taglibs* in eine JSP-Seite einzubinden. In diesen Taglibs werden verschiedene immer wieder gebrauchte Konstrukte, wie z.B. Schleifen, realisiert. Diese Konstrukte können dann mit nur einem *Tag* in die JSP-Seite eingefügt werden. Der Quelltext bleibt so kleiner und besser überschaubar. Neben Standard-Taglibs gibt es im JCommSy auch Taglibs, die spezielle für das JCommSy benötigte Funktionalität bereitstellen. Ein Beispiel ist ein Link-Tag, welches die CommSy-internen Links generiert, die mehr Informationen enthalten als die normalen HTML-Tags. Da diese JCommSy-Links für jeden Link in JCommSy gebraucht werden, reduziert sich der Code innerhalb der JSP durch das Auslagern dieser Funktionalität deutlich. Die Tags werden zusammen mit ihrer Definition und dem JCommSy ausgeliefert und sind dann auf jedem Server zusammen mit dem JCommSy einsetzbar.

Beans

Die Datenübertragung vom fachlichen Modell über die Steuerungsschicht zu den JSP-Seiten erfolgt mit *Java-Beans* (kurz: Beans). Diese Beans dienen nur zum Transportieren der Daten und erfüllen sonst keine fachliche Funktionalität. Das Füllen der Beans erfolgt in den Actions. Während der Ausführung einer Action werden die Beans erzeugt und nur an dieser Stelle über Setter-Methoden mit den entsprechenden Daten gefüllt. Ein weiterer schreibender Zugriff auf die Beans erfolgt nicht. Dies ist auch der einzige Zugriff, der regelwidrig von der Steuerungsschicht auf die darüber liegende Darstellungsschicht erfolgt. Die Beans werden von den Actions an den Request angehängt und können in den JSPs ausgelesen werden. Dies erfolgt über Getter-Methoden, die nur in den JSPs genutzt werden. Damit auf die Beans in den JSPs zugegriffen werden kann, müssen sie mit dem Bezeichner, mit dem sie an den Request angehängt wurden, und mit ihrem Typ in den JSPs deklariert werden.

2.1.2 Fachliches Modell

In dieser Schicht finden sich die fachlichen Bestandteile des JCommSy. Diese werden in fast allen darüber liegenden Schichten, mit Ausnahme der Darstellungsschicht, genutzt. Diese Schicht würde sich auch vertikal zu den Schichten „Darstellung“ , „Services“ und „DB-Mapping“ anordnen lassen (siehe [17]).

Items

Die *Items* stellen die fachlichen Materialien dar, mit denen ein CommSy-Benutzer arbeitet. Materialien sind die fachlichen, bearbeitbaren Konzepte der Anwendungswelt. Ein Beispiel für ein Item ist ein Termin. In einem Termin können Informationen wie Datum, Zeit, Titel, Bemerkungen und Ort zusammengefasst werden. Ein Termin-Objekt wird z.B. dann erzeugt, wenn ein neuer Termin angelegt wird. In dem Termin-Objekt werden die Daten gehalten, bis sie in der Datenbank gesichert werden. Um die Daten des Termins auf einer JCommSy-Seite anzuzeigen, wird ebenfalls ein Termin-Objekt mit Daten aus der Datenbank erzeugt, aus welchem dann die Daten für die Darstellung ausgelesen werden. Ein Item muss also verändernde und sondierende Methoden für die enthaltenen Daten bereitstellen.

Fachwerte

Fachwerte (engl. Domainvalues) sind fachlich motivierte Wertetypen mit einer festgelegten Wertemenge und erlaubten Operationen auf diesen Werten. Sie unterliegen der Wertsemantik, d.h. ein Wert kann im System nur einmal vorhanden sein (siehe: [22]). Im JCommSy werden z.B. die IDs der Items als Fachwerte behandelt. Jedem Item wird im JCommSy eine ID zugewiesen, über die es im System genau identifizierbar ist und die genutzt werden kann, um dieses Item von einem Service zu holen.

2.1.3 Steuerungsschicht

Der Ablauf einer Anfrage im JCommSy-System wird von verschiedenen *Servlets* gesteuert. Servlets sind in Java implementiert und werden in Webanwendungen für die Steuerung des Kontrollflusses genutzt. Sie nehmen eingehende Requests entgegen und erzeugen dymanisch eine entsprechende Antwort. Innerhalb des JCommSy werden verschiedenen Servlets eingesetzt, die z.T. die Funktion eines Dispatchers erfüllen, also die Verteilung des eingehenden Requests an spezialisiertere Servlets übernehmen.

Servlets

Jede eingehende Anfrage wird, nachdem sie verschiedene Filter durchlaufen hat, zuerst vom `CommsyServlet` entgegengenommen und dann an die Servlets der einzelnen Rubriken weitergeleitet. Neben dieser Weiterleitung ist das `CommsyServlet` auch dafür zuständig, die erforderlichen Informationen für den Rahmen einer CommSy-Seite zusammenzustellen.

Zusätzlich zum `CommsyServlet` gibt es noch weitere Servlets für die einzelnen Rubriken. In diesen Servlets wird die detaillierte Steuerung der eigentlichen Funktionalitäten der Rubriken übernommen. Dazu gehören z.B. das Anlegen neuer Items in einer Rubrik, das Anzeigen der Detailansichten eines Items oder das Löschen oder Bearbeiten eines Items. Die Servlets übernehmen wiederum vor allem eine koordinierende Funktion, da für die eigentliche Bearbeitung der Daten spezielle Action-Klassen vorhanden sind.

Actions

Die eigentliche Arbeit auf den Daten erfolgt nicht in den Servlets, sondern ist in spezielle *Action*-Klassen ausgelagert. Innerhalb des JCommSy-Systems sind diese mit dem Zusatz `Command` gekennzeicht, haben aber keinen Zusammenhang zum Command-Pattern nach [8]. In den Actions erfolgt der Zugriff auf die Services, um die Daten für die Darstellung zu holen, sowie das Erzeugen der Beans und das Befüllen dieser Beans mit Daten. Auch die Weitergabe dieser Beans an den Request erfolgt innerhalb der Action-Klassen.

Filter

Die *Filter* sind für die Vorbereitung des HttpRequests für das JCommSy nötig. Durch den `MigrationFilter` wird schon vor dem Einstieg in das `CommsyServlet` entschieden, ob der Aufruf im JCommSy bearbeitet werden kann. Zusätzlich werden die für den Aufruf wichtigen Informationen von ihrer Repräsentation im PHP-System in eine für das JCommSy nutzbare Repräsentation überführt. Der Tomcat-Webserver[1], auf dem das JCommSy läuft, unterstützt das automatische Ausführen von Filtern vor dem Einstieg in ein Servlet.

2.1.4 Services

Im JCommSy gibt es für die verschiedenen Arten von Items verschiedene *Services*. Über sie können anhand von IDs die gerade benötigten Items angefordert werden. Die Services kapseln die Datenhaltung. Sie greifen nicht direkt auf die Datenbank zu, sondern bekommen die Daten für die Items über Mapper, die den direkten Zugriff zur Datenbank kapseln. Materialien, die gespeichert werden sollen, werden wieder an die Services übergeben, die dann wiederum über die Mapper die Speicherung in der Datenbank koordinieren. Verschiedene Rubriken können auf mehrere Services zugreifen, da Daten von mehreren Rubriken für die Darstellung einer Rubrik nötig sein können. Dies ist der Fall, wenn z.B. ein Termin mit einem Material verknüpft ist. Die Zusammenarbeit mehrerer Rubriken wird im CommSy als Netznavigation (siehe [5]) bezeichnet.

2.1.5 DB-Mapping

In dieser Schicht wird die Verbindung zwischen dem fachlichen Modell und der Datenbank hergestellt. Der Zugriff auf die Datenbank erfolgt nicht direkt von den Services, sondern über

[1] Apache Tomcat - *http://tomcat.apache.org/*

spezielle *Mapper*. Diese kapseln die Übertragung der Daten zwischen den Services und der Datenbank. Nachdem diese Mapper während des „WebMig‘‘-Projektes zuerst mit fest codierten SQL-Anweisungen (dahingehend dynamisch, dass sie z.B für die Suche parametrisiert werden konnten) realisiert wurden, sind sie inzwischen im Rahmen der Diplomarbeit von Marcus Heiden (siehe [9]) auf Hibernate[2] umgestellt worden.

2.2 Ablauf einer Anfrage an die JCommSy-Termine-Rubrik

Im Folgenden soll der Ablauf einer Anfrage an die „Termine‘‘-Rubrik erläutert werden. Das JCommSy folgt bei der Bearbeitung von Anfragen einer MVC-2 Architektur. Die Anfragen werden von einem Servlet entgegengenommen (1 in Abb. 2.2), welches die Rolle des Controllers übernimmt. Es ist dafür verantwortlich, entweder zu spezialisierten Servlets weiterzuleiten, die dann die für die Darstellung benötigten Beans mit Daten füllen, oder es übernimmt diese Aufgabe selbst. Nachdem die Beans zum Transport der Daten erstellt sind (2 in Abb. 2.2), werden die Daten aus dem Model geholt und in die Beans gefüllt (3 in Abb. 2.2). Der Request wird daraufhin an die View, eine JSP, weitergeleitet (4 in Abb. 2.2). Diese JSP bekommt die Beans mit den Daten vom Request und kann aus diesen dann die darzustellenden Daten auslesen (5 in Abb. 2.2). Das aus der JSP erzeugte HTML wird dann mit dem Response an den Browser zurückgegeben (6 in Abb. 2.2).

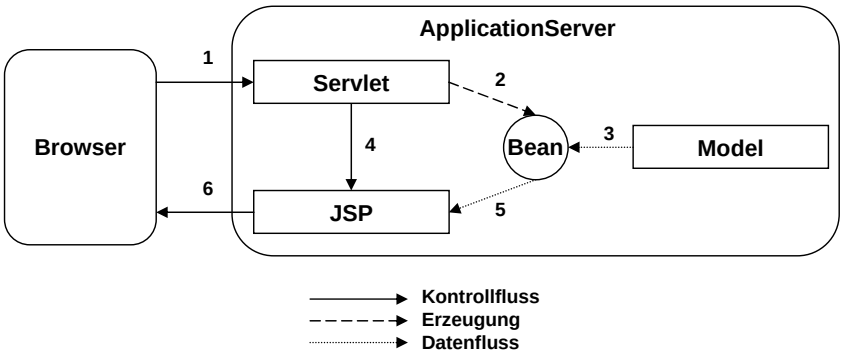


Abbildung 2.2: Ablauf einer Anfrage in einer MVC-2 Architektur

Wird die „Termine‘‘-Rubrik im PHP-CommSy aufgerufen, so prüft das PHP-CommSy, ob diese Rubrik vom JCommSy unterstützt wird. Die unterstützten Rubriken sind in der PHP-CommSy-Konfigurationsdatei `cs_config.php` definiert. Ist die „Termine‘‘-Rubrik dort als Java-Rubrik freigeschaltet, so wird die Anfrage nicht im PHP-CommSy behandelt, sondern an das JCommSy weitergeleitet. Im folgenden Beispiel etwa ist die javabasierte „Termine‘‘-Rubrik freigeschaltet (1), die javabasierte „Neuigkeiten‘‘-Rubrik jedoch auskommentiert (2) und damit nicht als Java-Rubrik sondern weiterhin als PHP-Rubrik benutzbar:

```
//for each migrated module set an array element 'true'
  $jcommsy['date'] = true;  (1)
```

[2]Hibernate - *http://www.hibernate.org/*

```
//$jcommsy['news'] = false; (2)
```

Listing 2.1: Auszug aus PHP-CommSy Steuerdatei

MigrationFilter

In beiden Systemen wird eine Weiche zwischen PHP- und JCommSy-System implementiert. Diese entscheidet, ob der Aufruf im eigenen System bearbeitet wird oder ob der Aufruf an das andere System weitergegeben werden muss. Im JCommSy wird diese Weiche im `MigrationFilter` implementiert. Dem Aufruf werden mit dem Request Informationen über die angeforderte Rubrik und die angeforderte Funktion mitgegeben, z.B. die Rubrik „Termine‘‘ mit der Funktion „neuen Termin anlegen‘‘. Um zu prüfen, welche Rubriken von JCommSy unterstützt werden, werden aus der Kontext-Datei des Tomcat-Servers die unterstützen Rubriken ausgelesen. In dieser Kontextdatei sind diese Rubriken ähnlich definiert wie in der Konfigurationsdatei des PHP-Systems. Der `MigrationFilter` kann anhand der Daten in der Kontext-Datei prüfen, ob die Daten aus dem Request zu einer unterstützten Rubrik gehören. Zusätzlich zu der Kontext-Datei müssen noch die Funktionen, die bei jeder Rubrik erlaubt sind, innerhalb des `MigrationFilters` definiert werden. D.h. die Definition von gültigen Rubrik/Funktion Kombinationen findet an zwei Stellen statt.

Innerhalb des JCommSy wird jede eingehende Anfrage zuerst vom `MigrationFilter` entgegengenommen (1 in Abb. 2.3). Nach der erfolgreichen Probe auf eine gültige Kombination von Rubrik und Funktion, wozu die angeforderte Kombination vom HttpRequest abgefragt wird (2 und 3 in Abb. 2.3), erfolgt die Weiterleitung zum `CommsyServlet`. Liegt keine gültige Kombination vor, wird also z.B. eine Rubrik mit einer nicht unterstützten Funktion aufgerufen, so wird der Request zurück an das PHP-CommSy geleitet und dort in der entsprechenden PHP-Rubrik dargestellt oder ggf. als Fehler ausgegeben. In diesem Mechanismus liegt jedoch bei falscher Einstellung die Gefahr einer Endlosschleife, wenn ständig zwischen PHP- und JCommSy hin und her gesprungen wird. Auch die Seitenaufrufe, die vom JCommSy ausgehen, durchlaufen zuerst den `MigrationFilter`. Dies ist insbesondere dann wichtig, wenn eine javabasierte Rubrik durch eine Anfrage mit einer PHP-Rubrik verknüpft werden muss. Für das Anhängen eines Materials (z.Z. noch PHP-CommSy) an einen Termin (als JCommSy-Rubrik verfügbar) wird eine enge Zusammenarbeit der beiden Systeme vorausgesetzt. Die Koordination eines solchen Aufrufs sowie die Aufteilung zwischen PHP- und JCommSy werden von den beiden Weichen übernommen.

Im `MigrationFilter` wird die HttpSession mit allen für die Bearbeitung der Anfrage im JCommSy benötigten Informationen versehen. Dabei handelt es sich z.B. um die Repräsentation des angemeldeten Benutzers und der Benutzungs-Historie dieses Benutzers (4 und 5 in Abb. 2.3). In der Benutzungs-Historie ist z.B. vermerkt, welche Einträge im CommSy sich der Benutzer schon angesehen hat. Zusätzlich wird vor dem weiteren Einstieg in das JCommSy geprüft, ob der angegebene Benutzer auch aktuell am System angemeldet ist. Dies ist wichtig, da die Angaben im Request z.T. über die Adressleiste des Browsers verändert werden können.

Das neu angelegte `CommsySession`-Exemplar wird über den `CommsySessionService` mit den Daten des Benutzers und der Benutzungs-Historie seiner bisherigen CommSy-Sitzungen aus der Datenbank gefüllt. Die `CommsySession` wird später, mit den neu hinzu gekommenen Daten, wieder in der Datenbank abgelegt, um so immer den aktuellen Stand der Historie des

Benutzers zu halten. Die Einträge auf der zu erzeugenden Seite können so korrekt dargestellt werden (z.B. der Hinweis „neue Anmerkung“ bei Terminen, wenn diese noch nicht vom Benutzer aufgerufen wurden, etc.). Von der `CommsySession` wird außerdem noch die Anzahl der anzuzeigenden Einträge auf der Terminseite an die HttpSession übergeben.

Die HttpSession ist damit für den Einstieg in das JCommSy vorbereitet. Es wird nun ggf. noch ein *FilterPattern* überprüft. Dieses dient z.B. nach einer Suche dazu, auszuwählen, welche und wie viele Termine in der Listenansicht der „Termine“-Rubrik angezeigt werden sollen.

Wenn an einen Termin Materialien angehängt worden sind, wird der Aufruf an dieser Stelle an das PHP-CommSy zurückgegeben, da die dafür benötigte „Material“-Rubrik noch nicht im JCommSy implementiert ist.

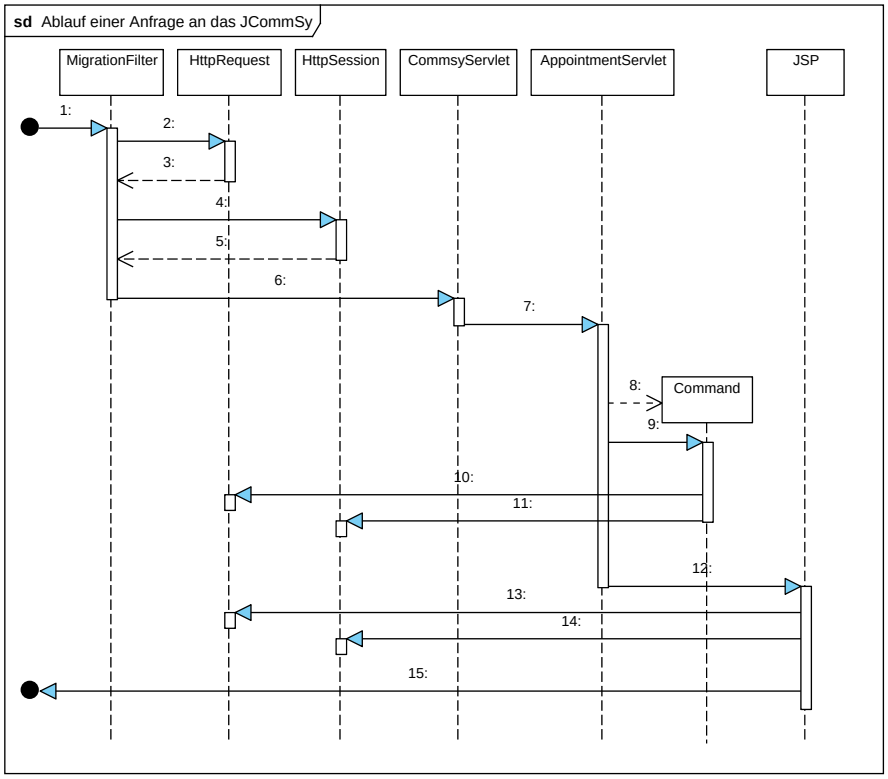


Abbildung 2.3: Ablauf einer Anfrage an die Termine Rubrik im JCommSy

CommsyServlet

Der HttpRequest wird an das `CommsyServlet` weitergeleitet (6 in Abb. 2.3). Dieses fungiert als Dispatcher, der alle eingehenden Requests entgegennimmt und sie dann an die spezialisierten Servlets weiterleitet. Das `CommsyServlet` übernimmt vorher noch die Verarbeitung von Informationen, die für die Darstellung der auf jeder Seite vorhandenen Oberflächenelemente, wie z.B. der LogIn-Bereich, benötigt werden. Im Falle der „Termine“-Rubrik erfolgt dann die Weiterleitung des Requests an das `AppointmentServlet` (7 in Abb. 2.3).

AppointmentServlet

Das `AppointmentServlet` prüft, welche Funktion (`index`, `edit`, `detail`) und, im Falle der Index-Seite, welche Ansicht (Liste oder Teamkalender) dargestellt werden sollen. Für jeden dieser Fälle gibt es eine eigene Action-Klasse, die ausgeführt wird (8 und 9 in Abb. 2.3). Die Action-Klasse stellt die für die Darstellung der Seite erforderlichen Beans mit den darzustellenden Daten zusammen und gibt diese Beans an den Request und weitere Informationen an die Session weiter (10 und 11 in Abb. 2.3).

JSP

Von `AppointmentServlet` wird der Request dann an die entsprechende JSP weitergeleitet (12 in Abb. 2.3). Für jede Funktion einer Rubrik gibt es entsprechende JSPs im System. Die JSPs bekommen über HttpRequest und HttpSession die benötigten Informationen für die Darstellung der Seite (13 und 14 in Abb. 2.3); für die darzustellenden Daten sind das die in den Action-Klassen erzeugten und gefüllten Beans. Die erzeugte HTML-Ausgabe wird mit dem Response zurück an den Browser geschickt (15 in Abb. 2.3).

Kapitel 3

Komponenten und OSGi

3.1 Komponenten

In diesem Kapitel sollen der Begriff *Komponente* und die dafür benötigten Konzepte geklärt werden.

3.1.1 Der Komponentenbegriff

Definition

Eine scharf umrissene Definition für den Begriff Komponente lässt sich in der Softwaretechnik schwer bestimmen. Der Begriff Komponente wird schon lange in der Informatik gebraucht und wurde auf viele verschiedene Weisen definiert. Von real greifbaren Dingen, wie der Dokumentation eines Softwareprojektes (siehe [10]), hat sich der Begriff Komponente heute vor allem zu einem rein technischen Begriff gewandelt. Schon früh wurden Softwarekomponenten mit Hardware, Maschinenteilen oder LEGO-Steinen verglichen.

> The resemblance to hardware components is intentional. [13]

Nach diesen Metaphern sollen sich Softwaresysteme wie die Teile einer Maschine oder wie LEGO-Steine, zusammensetzen lassen. Dabei sollen die gleichen Vorteile wie Wiederverwendbarkeit und universelle Einsetzbarkeit, die vor allem für LEGO-Steine gelten, erreicht werden. Zusätzlich soll es die Möglichkeit geben, alte Teile eines Systems gegen neue auszutauschen und das System durch weitere Teile auszubauen. Die Schnittstellen zu anderen Komponenten erlauben diese Art der Konstruktion und die leichte Änderbarkeit des Gesamtsystems. Ob ein LEGO-Stein aus Metall oder Kuntstoff besteht, ob er innen hohl oder gefüllt ist, ist hier nicht entscheidend. Solange er nach außen stabil und tragend ist und sich mit anderen Steinen zusammenstecken lässt - der Stein also die geforderte Schnittstelle aufweist - sind die verwendeten Materialien und der innere Aufbau für das Funktionieren des Gesamtsystems unerheblich.

Neben der Anschaulichkeit hat vor allem die Erkenntnis, dass sich in vielen Ingenieurwissenschaften irgendwann eine Entwicklung hin zu Komponenten vollzogen hat, zu der Verbreitung des Komponentenkonzepts in der Softwaretechnik beigetragen. Für die Softwaretechnik schien es also nur sinnvoll, ebenfalls ein Komponentenkonzept zu entwickeln. Allerdings unterscheidet sich Software grundlegend von den Gegenständen anderer Ingenieurwissenschaften. Während diese mit konkreten Produkten arbeiten, ist Software ein Metaprodukt, welches erst bei der Ausführung auf einem Computer zu einem konkreten Produkt wird (siehe [19]). Software ähnelt dem Bauplan eines Hauses statt dem Haus selbst. Sie lässt sich in immer anders parametrisierter Weise auf einem Computer ausführen, und das konkrete Produkt wird immer neu erzeugt. Ein Haus kann nicht jeden Tag mit einer anderen Anzahl Zimmer neu gebaut werden. Es gibt also ein grundlegendes Problem mit diesen Metaphern und dem Versuch, Softwarekomponenten durch sie zu beschreiben. ([19], 8-10). Um die Idee von Softwarekomponenten zu vermitteln, reichen diese Metaphern also nur bis zu einem bestimmten Grad aus. Je weiter sie auf die Details ausgeweitet werden, desto ungenauer werden sie (siehe dazu auch [16]).

In aktuellen Veröffentlichungen finden sich verschiedene Definitionen für den Begriff Komponente. Dadurch, und durch die steigenden Begeisterung für komponentenbasierte Softwareentwicklung, entsteht eine immer weiter fortschreitende Ungenauigkeit des Begriffs:

> [A]nd the semantics of the term “componentbased” have been diluted to an extent reminiscent of what occurred to the expression “object-oriented” in the previous software generation. [11]

Szyperski spricht von einem bestehenden „massive overloading of the term object“ (siehe [19]), in Folge dessen auch Objekte als Komponenten bezeichnet werden. Im allgemeinen softwaretechnischen Sprachgebrauch und in Veröffentlichungen verschwimmen die Begriffe Objekt, Klasse und Komponente immer mehr und werden synonym verwendet. Gerade bei den Begriffen Objekt und Komponente besteht diese Ungenauigkeit. Allerdings lassen sich Konzepte wie Schnittstellen, die Verwendung von Entwurfsmustern, der Einsatz von Rahmenwerken sowie die Sicht auf Objekte und Komponenten als Black- oder White-Boxes in beiden Bereichen problemlos und korrekt verwenden ([19], S. 35).

Meyer sieht in Komponenten eine „natural extension“ (Zitiert nach [10], S. 28) der objektorientierten Programmierung, obwohl Objektorientierung keine Voraussetzung für komponentenbasierte Softwareentwicklung ist.

> It has its roots in accepted architectural principles such as layering, modularization, and information hiding. [2]

Zusammenfassend finden sich in den verschiedenen Definitionen immer wiederkehrende Anforderungen an eine Komponente. Eine Komponente wird unter anderem dadurch definiert, dass sie (siehe [19], [2], [10], [18]):

- eine greifbare, zusammenhängende Einheit ist,
- allein und unabhängig (ohne zusätzliche Software) ausgeliefert werden kann,

- (ohne Anpassungen) mit anderen Komponenten kombinierbar ist,
- die Grundlage ist für die Konstruktion von größeren und komplexeren Systeme (auch von Dritten, die die Komponente nicht selbst implementieren, sondern nur benutzen),
- eine genau definierte öffentliche Schnittstelle hat,
- die Implementierung getrennt von der Schnittstellendefinition gehalten wird,
- unabhängig von anderen Komponenten eines Systems entwickelt werden kann,
- genau definierte Anforderungen an die (Laufzeit-)Umgebung stellt,
- in einem System eine genau abgrenzbare Einheit bildet.

Eine Gewichtung dieser Aspekte findet sich bei Hopkins:

> The definitions given here focus on the deployment aspect of components and their ability to be combined to form larger systems. [10]

Die erste Ausprägung einer softwaretechnischen Komponente, die dem heutigen Begriff nahe kommt, sind Prozedurbibliotheken. Komponenten können in verschiedensten Systemen vorliegen. Sie können z.B. aus einzelnen Prozeduren, einzelne Klassen, Modulen, oder ganzen Anwendungen bestehen. Als Beispiel sei hier die Eclipse-Umgebung genannt, die einzelne Funktionen einer Anwendung auf Komponenten (im Eclipse-Kontext werden sie als *Plugins* bezeichnet) aufteilt. Aber auch ein System aus Betriebssystem (als Komponentenplattform) und den darauf laufenden Anwendungen (als Komponenten) kann als Komponentensystem angesehen werden (siehe [19]).

Die Definition, die für diese Arbeit gelten wird, folgt Szyperski:

> A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties. [19]

3.1.2 Schnittstelle

Die Kommunikation und Zusammenarbeit mit dem System und anderen Komponenten erfolgt nur über die Schnittstellen, die von der Komponente erfüllt werden. Durch sie wird bestimmt, welche Services und Recourcen eine Komponente anderen Komponenten zur Verfügung stellt.

Eine Komponente implementiert mindestens eine Schnittstelle. Wie diese Implementierung genau aussieht, ist für das Benutzen der Komponente nicht von Bedeutung. Andere Komponenten werden explizit daran gehindert, die interne Implementierung zu sehen. Das Konzept des Information Hiding (siehe [15]) wird hier also konsequent umgesetzt, und Komponenten werden als Blackboxes betrachtet. Dies führt dazu, dass sich eine Komponente bei der Kommunikation und Zusammenarbeit mit anderen Komponenten nur auf deren Schnittstelle verlässt und keine Annahmen über die interne Implementierung dieser Komponenten macht. Auch der interne Zustand dieser Komponenten ist nicht nach außen sichtbar und für

die Benutzung der Komponente nicht wichtig. Der Aufruf eines Service einer Komponente liefert mit den gleichen Parametern immer das gleiche Ergebnis. Die Schnittstellen sollten so definiert werden, dass auch über sie so wenig Informationen wie möglich über die interne Implementierung der Komponenten nach außen gelangen. Bestehen zwischen Komponenten Annahmen über die interne Implementierung, wird es wahrscheinlicher, dass der Austausch von Komponenten nicht mehr problemlos erfolgen kann, da solche Annahmen nicht auf eine andere Komponente, die dieselben Schnittstellen implementiert, übertragen werden können. Die Schnittstelle ist der einzige verlässliche Faktor bei der Zusammenarbeit zwischen Komponenten. Die Implementierung der Schnittstelle erfolgt streng getrennt von der Schnittstellendefinition.

Schnittstellen zwischen Komponenten haben den Charakter eines Vertrags. So werden die Interfaces nicht direkt in den Komponenten, sondern getrennt von diesen für den jeweils gegebenen Kontext definiert. Zusammenarbeitende Komponenten verlassen sich gegenseitig darauf, dass genau die Dienste angeboten werden, die in dieser Schnittstellendefinition festgelegt sind. Auf der anderen Seite verlässt sich die Komponente, die einen Dienst anbietet, auch darauf, dass alle nötigen Vorbereitungen für das Nutzen dieses Dienstes von den Nutzern getroffen werden.

> [T]he specification of the interface becomes the mediating middle that lets the two parties work together. ([19], 50)

Für das korrekte Benutzen von Schnittstellen können noch mehr Informationen nötig sein, als nur die Definition der über die Schnittstelle angebotenen Dienste. So können auch Informationen über die Beziehungen, die zwischen zwei Diensten bestehen, z.B. zwischen „füge in Liste ein“, und „hole aus Liste heraus“. Letzterer Dienst funktioniert nur, wenn schon etwas in die Liste eingefügt worden ist. „Funktionieren“ ist nicht auf der technischen Ebene gemeint, sondern in einem Szenario, in dem ein konkretes Objekt aus einer Liste benötigt wird. Die von einer Komponente erfüllten Schnittstellen, die über eine Schnittstelle angebotenen Dienste, die Zusammenhänge zwischen Schnittstellen und Diensten sowie Informationen über vorausgesetzte Techniken und andere Komponenten im Einsatzkontext, werden in einer Komponentenspezifikation definiert. Genau wie Schnittstellen können auch solche Komponentenspezifikationen von mehreren Komponenten erfüllt werden.

Komponenten können dynamisch im System geladen werden. Welche konkrete Komponente über ein Interface angesprochen wird, wird erst zur Laufzeit entschieden. Das Konzept des späten Bindens ist hier also dasselbe wie in der objektorientierten Programmierung. Komponentensoftware ist damit auch zur Laufzeit sehr flexibel.

3.1.3 Komponentenmodell und Komponentenplattform

Komponenten werden nach einen definierten Komponentenmodell, und für eine bestimmte Komponentenplattform implementiert. Komponentenmodell und -plattform bilden die Grundlage für das Zusammensetzen von Komponenten zu einem Gesamtsystem und für die Zusammenarbeit von Komponenten untereinander.

Ein Komponentenmodell definiert, wie Komponenten andere Komponenten und deren bereitgestellte Services finden, und wie die Kommunikation und die Interaktion zwischen den Komponenten erfolgt.

Die Komponentenplattform ist für die Installation und den Einsatz von Komponenten zuständig. Sie bildet die Grundlage für die laufende Komponnentensoftware.

Für den realen Einsatz in einem Softwaresystem, werden Komponentenmodell und Komponentenplattform dann von einem (oder mehreren) Rahmenwerken realisiert. Eclipse Equinox implementiert etwa die Spezifikationen des OSGi-Rahmenwerks. Solche Rahmenwerke sind die Grundlage für große und komplexe, aus Komponenten zusammengesetzte, Softwaresysteme. Konkrete Komponenten werden dann meist durch Erweiterungen von Klassen eines Rahmenwerks erstellt.

3.1.4 Schichtenarchitektur und Modularisierung

Eine Komponente kann atomar oder aus anderen Komponenten zusammengesetzt sein. Die Architektur eines Komponentensystems kann als Hierarchie von solchen atomaren und zusammengesetzten Komponenten beschrieben werden.

In einem Komponentensystem ist es möglich, dass sich, durch die Nutzung von Funktionalität, eine klare Hierarchie zwischen den Komponenten und dadurch auch gut voneinander unterscheidbare horizontale Schichten bilden. Eine solche Schichtenarchitektur kann genutzt werden, um auf einem bestehenden Grundsystem neue Systeme aufzubauen. Die weiter oben liegenden, oft fachlich motivierten Schichten aus Komponenten („System 1“ in Abb. 3.1), können durch andere fachlich motivierte Schichten („System 2“ in Abb. 3.1) ersetzt werden. Die Schichten darunter, z.B. technische Schichten mit Komponenten für Datenbankanbindungen, können auf diese Weise wiederverwendet werden (Grundsystem in Abb. 3.1). Als Beispiel sei hier die Eclipse-Umgebung genannt. Deren Komponentenplattform und viele technische Komponenten lassen sich als Grundlage für beliebige RCP-Anwendungen wiederverwenden.

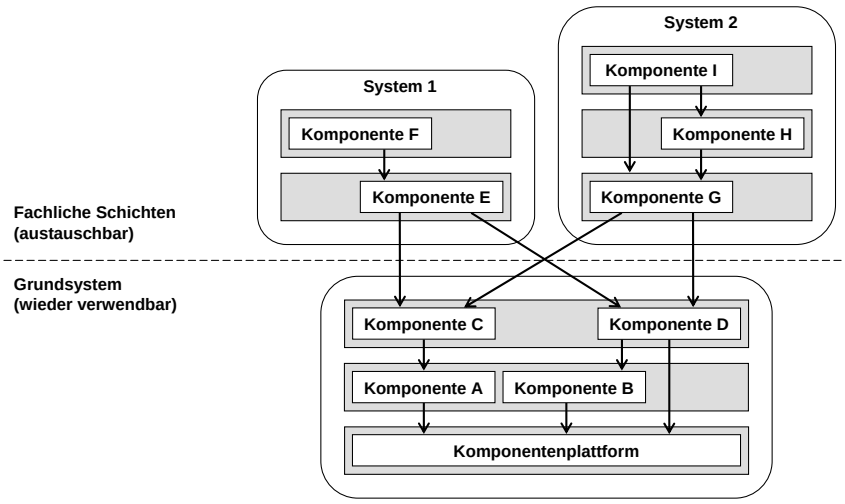


Abbildung 3.1: Wiederverwendbarkeit durch Schichtenarchitektur

Modularisierung beschreibt die Aufteilung des Softwaresystems in Module, die jeweils eine möglichst genau abgegrenzte fachliche oder technische Aufgabe übernehmen (siehe: [15]; [19], 39-40). Modularisierung ist eine Voraussetzung für eine komponentenbasierte Architektur.

Vor allem für die Wartung von Softwaresystemen ist eine gute Modularisierung hilfreich, da sich bestimmte funktionale Codestellen nur in einem Modul (einer Komponente) wiederfinden, also eine hohe Kohäsion des Systems erreicht wird. Bei Veränderungen des Systems müssen nur diese zentralen Codestellen geändert werden, anders als bei nicht modularisierten Systemen, in denen eine bestimmte Funktionalität evtl. über das ganze System verstreut implementiert ist. Ein anschauliches Beispiel dazu gibt z.B. Parnas in [15]. Zusammen mit der durch die Verwendung der Interfaces erreichten, losen Kopplung sind diese Systeme gut wartbar, testbar und erweiterbar.

> Such a [component-based] approach to software building should result in systems of higher quality and lower costs. Software plug-and-play also promises more evoivable systems — a result of the capability to easily replace constituent components. [2]

Komponentenbasierte Softwaresysteme ermöglichen Systeme, die sich leicht evolutionär weiterentwickeln. Bestehende Komponenten können durch Weiter- oder Neuentwicklungen ausgetauscht werden, und das System als Ganzes kann um neue Funktionalität erweitert werden. Teile der Systeme können als Grundlage für ganze Familien von Softwaresystemen genutzt werden (siehe Eclipse).

Für Komponenten ist aber auch eine Ausgewogenheit zwischen dem Umfang an Funktionalität, den sie selbst implementiert, und der Funktionalität, die sie in einem System voraussetzt wichtig. Bringt eine Komponente zu viel Funktionalität selbst mit, wird sie schnell schwergewichtig, und das Konzept der Aufteilung von Funktionalität wird nicht genutzt. Die Kohäsion der Komponente und des gesamten System nimmt ab. Setzt eine Komponente dagegen zu viel Funktionalität im System voraus, wird es schwerer, diese Komponente wieder zu verwenden, da zuerst alle Kontextabhängigkeiten der Komponente im System bereitgestellt werden müssen.

3.2 Technische Grundlage der JCommSy-Erweiterung - Die OSGi Service Platform

In diesem Kapitel werden die Grundlagen der *OSGi Service Platform* (siehe [20] und [21]) beschrieben. Diese Rahmenwerk-Spezifikation ist die technische Grundlage für die in dieser Arbeit durchgeführte Erweiterung des JCommSy.

3.2.1 Einführung

Die OSGi Service Platform ist die Spezifikation für ein Rahmenwerk, welches die Entwicklung und den Einsatz von erweiterbaren Softwaresystemen ermöglicht. Die einzelnen Komponenten einer solchen Anwendungen werden im OSGi-Kontext als *Bundles* bezeichnet. Die bekannteste

Anwendung, die die OSGi-Spezifikation mit einer eigenen Implementierung (*Equinox*, siehe [7]) benutzt, ist die Eclipse Entwicklungsumgebung (siehe [6]). Diese Implementierung der OSGi-Spezifikation wird mittlerweile als Referenzimplementierung angesehen.

Ein OSGi-Rahmenwerk übernimmt als Komponentenplattform sowohl den Start einer aus Bundles bestehenden Anwendung, sowie die Installation und das Entfernen von Bundles in bzw. aus einer laufenden Anwendung. Zum Startzeitpunkt einer Anwendung müssen zudem nicht alle Bundles geladen werden. Dem Rahmenwerk reicht es, zu diesem Zeitpunkt zu wissen welche Bundles vorhanden sind. Diese können dann bei Bedarf zur Laufzeit in das System nachgeladen werden. Das Rahmenwerk muss beim Starten nur die in Manifest-Dateien definierten Komponentenspezifikationen der Bundles auslesen, um die Abhängigkeiten zu anderen Bundles und der Umgebung überprüfen zu können.

3.2.2 Bestandteile der OSGi Service Platform

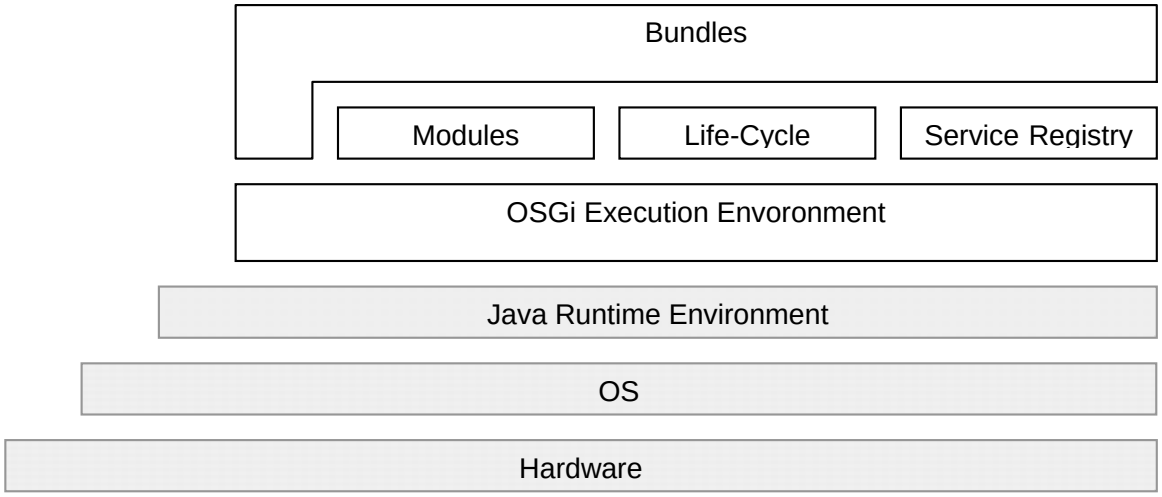


Abbildung 3.2: Schichten des OSGi-Rahmenwerkes

Module

Im OSGi-Rahmenwerk wird eine neue Art der Modularisierung definiert. Diese Module werden als Bundles bezeichnet. Ein Bundle besteht aus Java-Klassen und weiteren benötigten Recourcen. Durch den Export aus einem, und den Import in ein anderes Bundle, können die Java-Packages eines Bundles in anderen Bundles wieder verwendet werden.

Ein Bundle ist eine JAR-Datei,

- die Recourcen enthält, die die Funktionalität des Bundles ermöglichen. Dies sind die Java Klassen der Anwendung, es können zusätzlich aber auch andere Dateien wie Bilder, HTML- oder JSP-Dateien sein.

- die eine Manifest-Datei enthält, in der das Bundle und die enthaltenen Recourcen und Bedingungen an die Umgebung (etwa vorausgesetzte Bundles und zu exportierenden bzw. zu importierte Packages) definiert sind.
- die ein Verzeichnis OSGI-OPT enthalten kann, welches optionale Recourcen, etwa den Quelltext des Bundles, enthält.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: JCommSy Plug-in
Bundle-SymbolicName: JCommSy
Bundle-Version: 1.0.0
Bundle-Localization: plugin
Bundle-Activator: jcommsy.plugin.control.Activator
Import-Package: org.osgi.framework;version=1.3.0,
Export-Package: jcommsy.plugin.plugin,
 jcommsy.plugin.servlet
```

Listing 3.1: Beispiel einer Manifest-Datei für ein OSGi-Bundle

Diese Manifestdatei beschreibt das Bundle „`JCommSy Plug-in`“(`Bundle-Name`) in der Version 1.0.0 (`Bundle-Version`), welches im System als „ `JCommSy`“(`Bundle-SymbolicName`) bekannt ist. Beim Start des Bundles wird die Aktivator-Klasse „`jcommsy.plugin.control.Activator`“benutzt. Das Bundle importiert nur das Basis-Package des OSGi-Rahmenwerks „`org.osgi.framework;version=1.3.0`“, und exportiert (für die Plugins, die dieses Bundle erweitern) die Packages „`jcommsy.plugin.plugin`“und „`jcommsy.plugin.servlet`“.

Life Cycle

In der Life-Cycle Schicht wird die Installation, Aktivierung, Deinstallation etc. von Bundles übernommen. Ein Bundle kann durch ein anderes Bundle, oder per Hand, über die *OSGi-Konsole* (siehe [14]) installiert und gestartet werden. Die für das Bundle relevante Startsequenz wird in einer speziellen Klasse, dem *Bundle-Activator*, definiert. Diese Klasse implementiert das Interface `BundleActivator`, wobei in den Methoden `start` und `stop` die jeweils benötigten Anweisungen für das Starten und Stoppen eines Bundles implementiert werden.

Ein Bundle kann verschiedene Zustände einnehmen:

- INSTALLED = Das Bundle ist installiert worden und dem System somit bekannt. Es kann aber noch nicht gestartet werden, da nicht alle Voraussetzungen erfüllt sind (z.B. nicht alle vorausgesetzten Bundles vorhanden, bzw. als RESOLVED gekennzeichnet sind).
- RESOLVED = Alle Voraussetzungen (andere Bundles bzw. zu importierende Packages andere Bundles) sind vorhanden, so das dieses Bundle gestartet werden kann. Ein Bundle nimmt auch diesen Zustand ein, wenn es gestoppt worden ist.
- STARTING = Mit der `start`-Methode der Activator-Klasse werden die erforderlichen Schritte zum Starten des Bundles ausgeführt.
- ACTIVE = Das Bundle wurde gestartet und läuft in der Anwendung.

- STOPPING = Mit der `stop`-Methode der Activator-Klasse werden die erforderlichen Schritte zum Stoppen des Bundles ausgeführt.
- UNINSTALLED = Das Bundle ist deinstalliert worden, und kann vom System vor einer erneuten installation nicht verwendet werden.

Folgender Zustandsgraph ergibt sich:

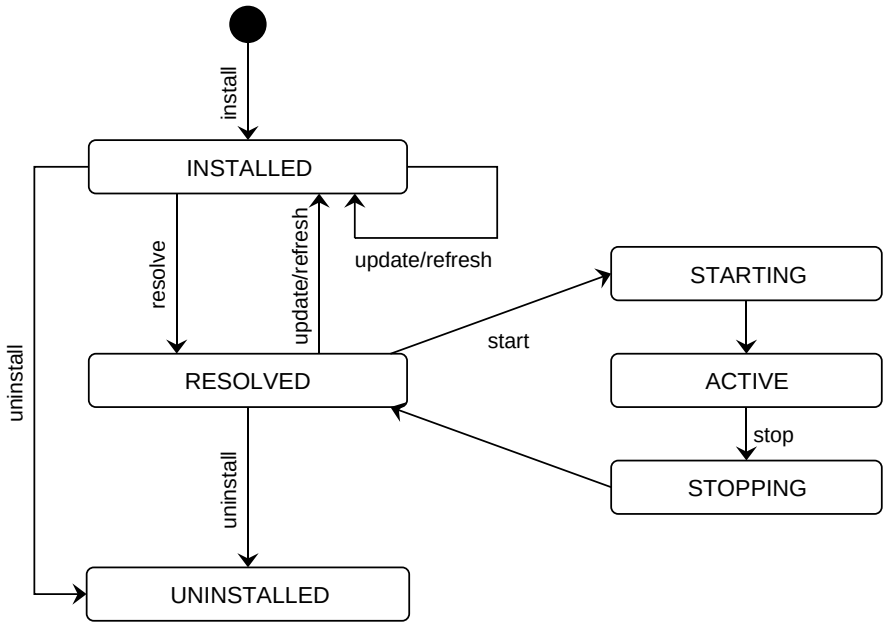


Abbildung 3.3: Zustandsgraph eines OSGi-Bundles

Services

Die Service Schicht eines OSGi-Rahmenwerks arbeitet eng mit der Life-Cycle-Schicht zusammen. Die Service-Schicht implementiert ein *publish find and bind*-Modell (siehe [20], S. 103). Die Komponente (*Service Provider*), die einen Service im System anbieten möchte, registriert diesen Service bei einem zentralen *Service Broker* (1 in Abbildung 3.4). Diese Funktion wird vom OSGi-Rahmenwerk selbst übernommen. Bei dieser zentralen Instanz können andere Komponenten (*Service Requester*) nach einem registrierten Service suchen (2 und 3 in Abbildung 3.4), und dann die Verbindung zu dem Anbieter herstellen, um diesen Service nutzen zu können (4 in Abbildung 3.4). Ein Service selbst ist ein Java-Objekt, welches unter einem oder mehreren Service-Interfaces in der Service-Registry des Rahmenwerks registriert wird. Die Bundles in der Anwendung können Services registrieren, nach ihnen suchen, und bei Änderungen des Status eines Service benachrichtigt werden.

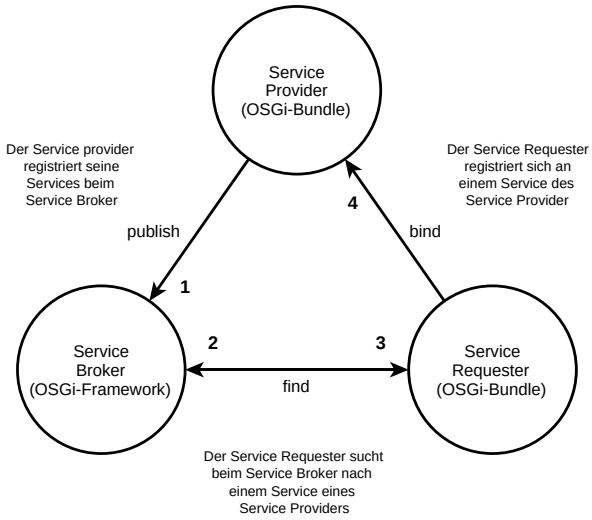


Abbildung 3.4: *Publish, find and bind*-Modell

Service

Ein Service in einer OSGi-Anwendung wird semantisch durch die öffentlichen Methoden seiner Service Schnittstelle definiert, und durch sein Service Objekt implementiert. Jeder Service läuft innerhalb seines Bundles. Dieses Bundle ist dafür verantwortlich, den Service in der Service Registry des Rahmenwerks zu registrieren, damit der Service anderen Bundles zur Verfügung stehen kann.

Die Zusammenarbeit zwischen dem Bundle, welches den Service angemeldet hat, und den Bundles die diesen Service nutzen wird vom Rahmenwerk gesteuert. Das Rahmenwerk übernimmt dabei z.B die Aufgabe das Service Interface auf des Service Objekt abzubilden, das Suchen nach Service zu ermöglichen und Bundles über Änderungen des Status eines Services zu informieren. Das Rahmenwerk stellt dafür einen Event-Mechanismus bereit, an dem sich Bundles registrieren können.

3.2.3 Server-Side OSGi

Nachdem sich das Equinox-Rahmenwerk auch für die Entwicklung von Rich Client Platform-Anwendungen auf Basis von Eclipse weit verbreitet hat, hat die Portierung dieser Technik auf Webserver begonnen. Dadurch sollen die Vorteile von Erweiterbarkeit und Wiederverwendbarkeit auch für Webanwendungen genutzt werden können.

Während der Entwicklungsphase von OSGi-Webanwendungen kann die OSGi-Umgebung von Eclipse als Laufzeitumgebung benutzt werden. In dieser Umgebung laufen dann die Bundles der Anwendung neben Bundles die z.B. einen Webserver realisieren (z.B. Jetty[1]). Die einzelnen Bundles müssen dann nicht nach jeder Änderung aus den Eclipse-Projekten exportiert und auf einem Webserver installiert und gestartet werden.

[1] Jetty Webserver - *http://jetty.mortbay.org/*

Das fertig laufende System auf dem Webserver (z.B. Tomcat) benutzt als Basis die *Servlet Bridge* von Equinox. Dies ist eine Webanwendung, die auf dem Webserver installiert wird und dort eine OSGi-Umgebung startet. Die Verzeichnisstruktur dieser Webanwendung ähnelt der einer Eclipse-Installation. Die Bundles der Webanwendung werden in ein spezielles Verzeichnis kopiert, von wo aus sie vom OSGi-Rahmenwerk benutzt werden können. Die dort hinterlegten Bundles werden zwar automatisch gefunden, in der OSGi-Konsole allerdings zuerst nur als INSTALLED bzw. als RESOLVED angezeigt. Ein Start der Bundles erfolgt erst durch einen expliziten Befehl auf der OSGi-Konsole oder automatisch beim Neustart der Servlet Bridge nach einem entsprechendem Eintrag in der `config.ini` (Ähnelt auch hier einer Eclipse-Installation).

Zum Zeitpunkt dieser Arbeit werden alle gängigen Technologien für Webanwendungen - JSP, HTML, XML, Datenbankanbindungen, etc. - auch aus den jar-Dateien der Bundles heraus unterstützt. Für die Registrierung dieser Recourcen werden *ExtensionPoints* (siehe [3]) des Rahmenwerks genutzt. Manche Techniken, wie etwa das automatische Ausführen von Filtern in einem Tomcat-Webserver, funktionieren noch nicht und müssen per Hand in den Kontrollfluß integriert werden.

Kapitel 4

Realisierung der Plugin-Schnittstelle

4.1 Erster Integrationsversuch einer externen Rubrik in das JCommSy

Bei diesem Versuch, der z.T. noch in der Themenfindungsphase konstruiert wurde, ging es darum einen ersten Überblick über die benötigten Schnittstellen zwischen Erweiterungen und Kernsystem zu finden. Als Erweiterung wurde die von Christof Kubosch in seiner Baccalaureatsarbeit (siehe [12]) entworfene XPlanner-Rubrik in das System eingebaut.

4.1.1 Erfolgte Änderungen an PHP- und JCommSy

Um die XPlanner-Rubrik in das JCommSy zu integrieren, waren nur an wenigen Stellen Änderungen an bestehenden Klassen des JCommSy nötig. Dies liegt aber vor allem daran, dass es sich um die Einbindung einer externen Anwendung handelte, und keine Verknüpfungen zwischen dieser Anwendung und anderen CommSy-Rubriken beachtet werden mussten. Diese Änderungen betrafen nur das `CommsyServlet` und den `MigrationFilter`. Alle weiteren Komponenten der XPlanner-Rubrik wurden zum JCommSy-Projekt hinzugefügt, ohne direkten Einfluss auf die bereits vorhandenen Systemteile zu haben.

Im `MigrationFilter` musste nur die bestehende Weiche zwischen PHP- und JCommSy erweitert werden, so dass eine Umleitung zum Servlet der Xplanner-Rubrik möglich wurde. Für die XPlanner-Rubrik wurde der Bezeichner der schon vorhandenen Ankündigungen-Rubrik des CommSy genutzt, da diese Rubrik im Entwicklungs-Projektraum nicht eingesetzt wurde. D.h. die XPlanner-Rubrik konnte mit der Rubrik „todo ‘‘ in Kombination mit einer der Funktionen „index‘‘, „detail‘‘ oder „edit‘‘ aufgerufen werden. Im `CommsyServlet` wurde die Weiterleitung zu den spezialisierten Servlets auf das `XPlannerServlet` erweitert. Die gleiche Erweiterung der Weiche war auch im PHP-CommSy nötig.

Zusätzlich zu diesen Änderungen am bestehenden System mussten noch die eigenständigen Teile der neuen Rubrik - Servlets, Beans, JSPs, etc. - hinzugefügt werden. Diese Teile der

XPlanner-Rubrik haben keine Verbindung zu den bereits bestehenden Teilen des Systems. Sie sind während der Integration zu den fachlich passenden Packages hinzugefügt worden, hätten aber auch ein eigenes Package bilden können.

4.1.2 Erfahrungen der prototypischen Einbindung

Das Einbinden von externen Anwendungen in den Kontrollfluss des JCommSy war in diesem Fall leicht möglich. Allerdings ist dafür ein direkter Eingriff in Klassen des JCommSy nötig. Da diese externen Anwendungen eigenständig laufen, müssen sie für ihre Datenhaltung nicht auf die bestehende Datenbank des CommSy zugreifen. Sie sind nur lose an das bestehende System gekoppelt. Eine Einbindung anderer eigenständiger Anwendungen in die Oberfläche des JCommSy sollte also ähnlich möglich sein, wie die Einbindung des XPlanners. Welche Anpassungen für die Zusammenarbeit solcher Anwendungen mit anderen Rubriken nötig sind, hat dieser erste Versuch nicht gezeigt. Die in dem XPlanner-Versuch angezeigten Seiten sind nur graphisch überarbeitete XPlanner-Seiten. Eine direkte Verbindung zum JCommSy ist nicht vorhanden. Daher gib es keine Möglichkeit, einzelne Einträge der XPlanner-Rubrik mit Anmerkungen zu versehen oder mit anderen Rubriken zu verknüpfen. Verknüpfungen dieser Art sollen für die Erweiterungen jedoch möglich gemacht werden. Es stellen sich hier also schon Kandidaten für Verbindungen zwischen dem JCommSy und den Erweiterungen heraus. Das JCommSy sollte Schnittstellen für

- die Einbettung der Erweiterung in die umgebende Seite,
- das Anhängen von Anmerkungen an Einträge in Erweiterungsrubriken,
- die Netznavigation

anbieten. Als Netznavigation wird im CommSy die Vernetzung der Rubriken untereinander bezeichnet. So können einem Termin z.B. bestimmte Personengruppen zugeordnet werden, die dann in der Detailansicht des Termins aufgelistet werden. Die Terminrubrik muss also für die vollständige Darstellung der Inhalte auch Daten von anderen Rubriken nutzen.

4.2 Prototypische Erweiterung des JCommSy mit Equinox

4.2.1 Bestandteile des OSGi-Prototypen

Während der Bearbeitungszeit dieser Arbeit wurde im Rahmen des Eclipse-Projektes begonnen, die Equinox-Umgebung auch für Webserver bereitzustellen. Trotz des frühen Status dieser Entwicklung habe ich mich entschlossen, einen JCommSy-Prototypen mit dieser Technologie zu bauen. Der größte Vorteil schien mir dabei zu sein, eine erprobte und mächtige Plugin-Schnittstelle zu nutzen. Zudem sollte mit dem Prototyp herausgefunden werden, ob alle benötigten Technologien (Servlets, JSPs, Taglibs, Datenbankanbindung) problemlos aus den jar-Dateien der Bundles heraus funktionieren.

Durch das Equinox-Rahmenwerk ist es möglich, Erweiterungen in ein laufendes System einzuspielen, bzw. wieder aus diesem System zu entfernen, ohne es herunterfahren zu müssen.

Speziell für ein Online-System mit einer großen Benutzerzahl ist dies ein Vorteil. Rubriken können so bei Bedarf gestartet und gestoppt oder auch durch Weiterentwicklungen ersetzt werden, ohne das System herunterfahren zu müssen. Für JCommSy-Erweiterungen, die mit anderen Rubriken im System zusammenarbeiten, könnte auch die Möglichkeit von Extension-Points oder Services im OSGi-Rahmenwerk genutzt werden, um über diese Mechanismen die Funktionalität des Grundsystems zu erweitern.

Der Prototyp für die Realisierung mit OSGi-Bundles bestand aus zwei Bundles. Einem Haupt-Bundle, welches stellvertretend für das später zu implementierende JCommSy-Bundle als zentrale Instanz der Bundle-Architektur funktionierte, und einem Erweiterungs-Bundle. Das Haupt-Bundle wird, genauso wie das Erweiterungs-Bundle, am Equinox-System angemeldet und registriert dort seine Recourcen (Servlets und JSPs). Zusätzlich registriert das Haupt-Bundle einen Listener, der auf das Registrieren von Services am System reagiert. Dieser Listener reagiert nur auf das Registrieren solcher Service-Objekte, die das Interface `JCommsyPlugin` implementieren. Zusätzlich zur Registrierung am Equinox-System, wird das Erweiterungs-Bundle dadurch auch im Haupt-Bundle in einem Plugin-Service registriert. Dem Haupt-Bundle sind somit alle installierten Erweiterungs-Bundles bekannt. Die spätere Zusammenarbeit von Haupt- und Erweiterungs-Bundle bzw. die Zusammenarbeit von Erweiterungs-Bundles untereinander, können auf diese Weise vom Haupt-Bundle koordiniert werden.

Für die technische Realisierung sind im Haupt-Bundle Klassen in mehreren Packages zuständig. Im Package `de.unihamburg.informatik.jcommsy.plugin.control` sind die Klassen für den Start des Haupt-Bundles und die Kontrolle über die Erweiterungs-Bundles zusammengefasst. D.h. die Klassen zum Starten und Registrieren der Recourcen, die Listener für das Registrieren von Erweiterungs-Bundles und der Plugin-Service, in den die Erweiterungs-Bundle eingetragen werden und über den sie später von anderen Klassen im JCommSy zugreifbar sind. Dazu das Package `de.unihamburg.informatik.jcommsy.plugin.plugin`, welches abstrakte Klassen enthält, die von den Klassen der Erweiterungs-Bundles erweitert werden. Alle benötigten Methoden für das Anmelden der Plugins am System sind in den abstrakten Klassen implementiert, so dass lediglich bestimmte Werte über Einschubmethoden in den erweiternden Klassen gesetzt werden müssen. Auch die Registrierung der Recourcen der Erweiterungs-Bundles ist in den abstrakten Klassen schon implementiert und erfolgt auf diese Weise automatisch beim Starten des Bundles. In einem Unterverzeichniss des Projektes sind die Web-Recourcen untergebracht, die ebenfalls am Rahmenwerk registriert werden. Dazu gehören z.B. JSPs und Taglibs.

```
@Override
public void setData() {
  name = "Plugin1";
  link = "/jcommsy/plugin1/Plugin1";
  servlet = new PluginServlet1();
  alias = "/jcommsy/plugin1";
  dir = "/WebContent";
}
```

Listing 4.1: Überschriebene `setData`-Methode in der Plugin-Klasse des Erweiterungs-Bundles

Die Architektur dieses Prototypen ließ sich in der späteren Realisierung der Plugin-Schnittstelle weiter nutzen. Sie wird in Kapitel 4.3.1 genauer erläutert.

4.2.2 Erfahrungen durch den OSGI-Prototypen

Der Prototyp hat gezeigt, dass die im JCommSy verwendeten Web-Techniken auch aus einem OSGi-Bundle heraus funktionieren. Für Erweiterungs-Bundles müssen nur wenige Klassen erweitert werden, und es ist kein Einblick in die Implementation des JCommSy nötig. Als Grundlage für die Realisierung lässt sich festhalten:

- Die Kommunikation zwischen Haup- und Erweiterungs-Bundle findet über nur eine Schnittstelle statt. Diese muss vom Erweiterungs-Bundle implementiert werden. Für die Erweiterungs-Bundles ist nur das Vorhandensein des Haut-Bundles eine Voraussetzung. Die Erweiterungs-Bundles sind lose an das Haupt-Bundle gekoppelt.
- Wie gewünscht funktioniert das Hinzufügen und Entfernen der Erweiterungs-Bundle zu bzw. aus der laufenden JCommSy-Anwendung problemlos.
- Neben dem Haupt- und den Erweiterungs-Bundles sind nur Standart-Bundles nötig, die das OSGi-Grundsystem und Unterstützung für bestimmte Techniken (z.B. JSP-Unterstützung) bereitstellen. Diese sind aber überschaubar und können in der Standartkonfiguration genutzt werden.
- Allem Anschein nach scheint sich durch die Unterstützung des Rahmenwerks auch das Hochladen von neuen Plugins über ein Webformular realisieren zu lassen.

4.3 Architektur des Systems nach der Umstellung

JCommsy-Bundle steht hier für das Haupt-Bundle, welches grob dem aktuellen JCommSy entspricht.

4.3.1 Architektur des JCommSy mit Plugin-Schnittstelle auf OSGi-Basis

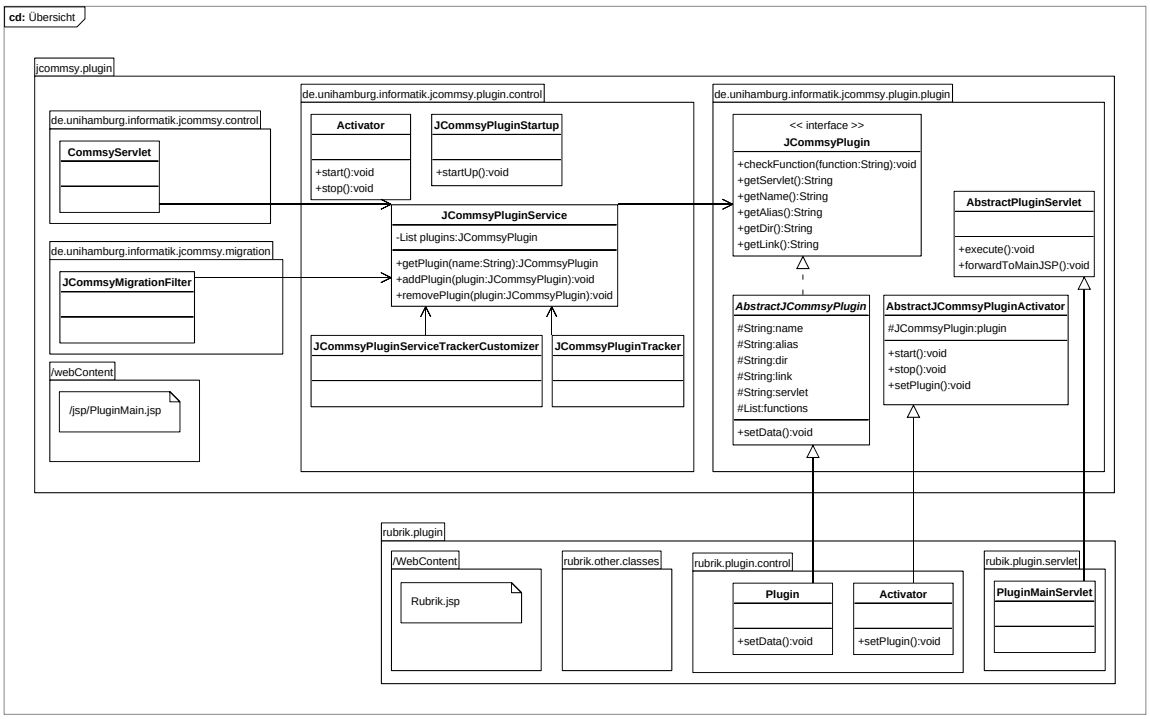


Abbildung 4.1: Klassendiagramm der Bestandteile der Plugin-Schnittstelle

Die Architektur des Prototypen ließ sich für die Realisierung der Plugin-Schnittstelle im JCommSy weiter verwenden. Die im Prototypen implementierten Klassen wurden dazu dem JCommSy-Projekt hinzugefügt und angepasst. Zusätzlich zum Test-Servlet des Prototypen wurde nun auch das `CommsyServlet` des JCommSy am OSGi-Rahmenwerk registriert, so dass eine Weiterleitung vom PHP-System zu dem als Bundle laufenden JCommSy möglich war. Allerdings scheiterte ein erfolgreicher Aufruf zuerst an nicht unterstützten Techniken (z.B. Filtern), die beim bisherigen Betrieb des JCommSy sonst automatisch vom Tomcat-Webserver übernommen wurden.

Packages für die Umstellung

Das Package `de.unihamburg.informatik.jcommsy.plugin.control` ist für das Starten des JCommsy-Bundles und das Auffinden sowie Verwalten von Erweiterungs-Bundles zuständig.

Die Klasse `Activator` (siehe: Abb. 4.1) implementiert das Interface `BundleActivator` und wird beim Starten des JCommsy-Bundles verwendet. Bei jedem OSGi-Bundle muss eine Klasse, in der die erforderlichen Schritte für das Starten und Stoppen eines Bundles implementiert werden, in der Manifest-Datei als Bundle-Activator definiert werden.

In der Klasse `JCommsyPluginStartup` werden die Filter, die sonst automatisch bei jedem Seitenaufruf durch den Tomcat-Webserver ausgeführt werden, aufgerufen. An einer Un-

terstützung von Equinox für solche Filter wird gearbeitet, diese lag aber während der Zeit der Diplomarbeit noch nicht vor. Der `JCommsyPluginTracker` ist für das Registrieren der Recourcen des JCommsy-Bundles verantwortlich. Zusätzlich zu den Servlets werden hier die Inhalte des Verzeichnisses `/WebContent` (hier nur JSPs, sonst auch: HTML, Bilder, etc.) am OSGi-Rahmenwerk registriert.

Der Event-Listener, der auf das Registrieren von Services am OSGi-Rahmenwerk reagiert, wird in der Klasse`JCommsyPluginServiceTrackerCustomizer` so angepasst, dass er nur auf Services reagiert, die das Interface `JCommsyPlugin` implementieren. Diese Klasse erweitert ebenfalls eine Rahmenwerk-Klasse. Die Services werden dann beim `JCommsyPluginService` registriert. Das Konzept der Services ist hier nicht komplett so genutzt wie in einer OSGi-Anwendung vorgesehen, aber nach Rücksprache mit den Equinox-Entwicklern ist dies eine gangbare Alternative zu ExtensionPoints. Letztere scheinen eher für eine losere Kopplung gedacht zu sein, als sie hier sinnvoll erschien.

Im Package `de.unihamburg.informatik.jcommsy.plugin.plugin` befinden sich die Klassen, die die Grundlage neuer Plugins bilden. Der `AbstractJCommsyPluginActivator` (siehe: Abb. 4.1) benötigt zum Starten des Erweiterungs-Bundles nur ein `JCommsyPlugin`-Objekt. Alle Funktionen zum Registrieren der Recourcen und der Servlets am System sind in dieser Klasse realisiert (und funktionieren genau wie beim JCommSy-Bundle). Die Programmierer müssen nur zwei Unterklassen anlegen (siehe: `AbstractJCommsyPlugin` und `Plugin` sowie `AbstractJCommsyPluginActivator` und `Activator` in Abb. 4.1). Die benötigte Konfiguration für das Plugin wird in Einschubmethoden implementiert.

```
@Override
protected void setPlugin() {
  plugin = new Plugin();
}
```

Listing 4.2: Setzen des Plugins in der Einschubmethode der Activator-Klasse

```
@Override
public void setData() {
  name = "Literatur␣Rubrik";
  link = "/jcommsy/plugins/Literatur";
  servlet = new LiteraturServlet();
  alias = "/jcommsy/plugins";
  dir = "/WebContent";
  // + mögliche weitere Funktionen...
}
```

Listing 4.3: Setzen der Werte in der Einschubmethode der Plugin-Klasse

Im `AbstractPluginServlet` (siehe: Abb. 4.1) wird eine Weiterleitung der `doPost`- und `doGet`-Methoden zu einer einzigen `execute`-Methode vorgenommen. Zusätzlich definiert die `forwardToMainJSP`-Methode dieser Klasse zu welcher Rahmenseite im JCommsy-Bundle weitergeleitet werden soll, sobald die darzustellenden Inhalte im Erweiterungs-Bundle-Servlet verarbeitet worden sind. Der Link zur JSP-Seite des Erweiterungs-Bundles, die den Inhalt darstellt, wird dem Request als Parameter mitgegeben und später per `include` in die Rahmenseite des JCommSy-Bundles eingefügt.

Um einen Zugriff auf die Datenbank per Hibernate möglich zu machen musste noch ein Wrapper-Bundle um die Standard-jars von Hibernate gebaut werden, welches über den Buddy-Modus (siehe [6]) Zugriff auf den Classloader des JCommSy-Bundles hat, und so das Mapping der Recourcen möglich macht.

Auf Komponenten-Ebene ergibt sich folgende Architektur:

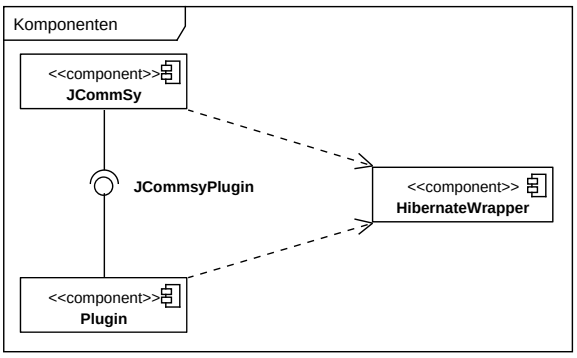


Abbildung 4.2: Komponenten nach der Erweiterung um eine Plugin-Schnittstelle

4.3.2 Ablauf einer Anfrage

Ein Aufruf einer Erweiterungs-Rubrik wird zuerst an das `CommsyServlet` weitergeleitet (1 in Abb. 4.3). Dort werden zuerst die Filter ausgeführt (2 in Abb. 4.3). Für die Probe auf eine unterstützte Rubrik/Funktion-Kombination muss ein Zugriff auf die `JCommsyPlugin`-Klasse über den `PluginService` erfolgen (3. und 4. in Abb. 4.3). Auch die Information zu welchem Plugin-Servlet vom `CommsyServlet` weitergeleitet wird, wird von der `JCommsyPlugin`-Klasse abgefragt (6. und 7. in Abb. 4.3).

Wie in der Architektur des JCommSy vor der Erweiterung um eine Plugin-Schnittstelle, wird auch hier zuerst eine Weiche durchlaufen. Die Rubriken und Funktionen der Erweiterungen werden nicht, wie vorher, in einer Kontext-Datei und einer Java-Klasse eingetragen, sondern die Weiche wurde so erweitert, dass alle installierten Erweiterungs-Bundles automatisch überprüft werden. Sie werden dazu über den `PluginService` abgerufen.

Nachdem im CommsyServlet die Informationen für die Umgebende Seite (in Abb. 4.3 die `PluginMain.jsp`) bearbeitet worden sind, wird an das entsprechende Erweiterungs-Bundle-Servlet weitergeleitet, um dort die Informationen für den Inhalt der Seite zu bearbeiten (8. in Abb. 4.3). Mit diesen Daten folgt dann ein `forward` zur `PluginMain.jsp` (9. und 10. in Abb. 4.3). Ein Link zu der JSP auf der die Inhalte dargestellt werden (in Abb. 4.3 die `Literatur.jsp`), wird als Parameter an den Request gehängt, so dass diese Seite dann per `include` in die `PluginMain.jsp` eingefügt werden kann.

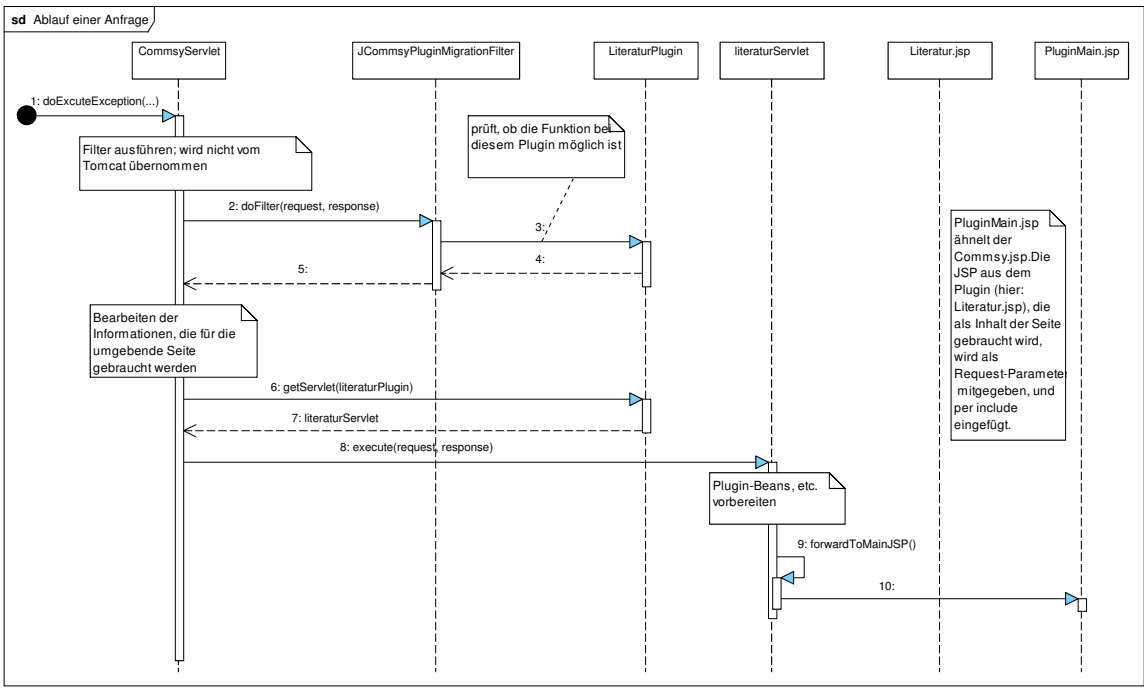
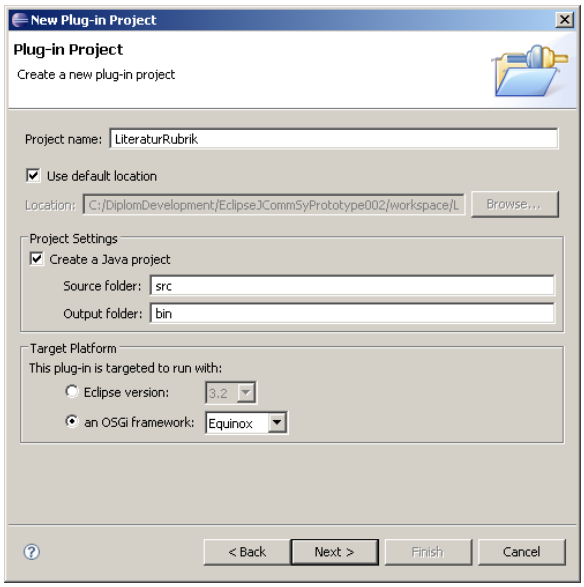


Abbildung 4.3: Sequenzdiagramm einer Anfrage nach der Erweiterung um eine Plugin-Schnittstelle

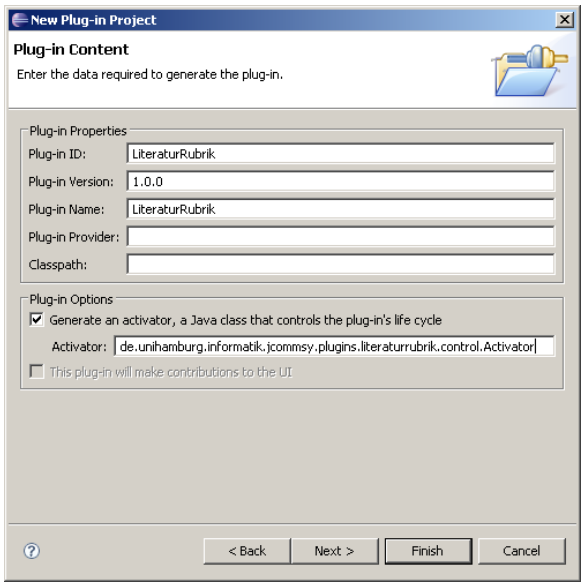
Den Rahmen der Seite stellt das JCommSy-Bundle als JSP bereit. Als Inhalt werden JSPs der Erweiterungs-Bundles verwendet. Das Zusammensetzen per `include` von JSPs aus verschiedenen Plugins klappt problemlos. Wichtig ist hier eine genaue Konvention, wie das WebContent-Verzeichnis der einzelnen Erweiterungs-Bundles ansprechbar sein soll, um nicht mit schon vorhandenen URL-Teilen in Konflikt zu geraten. Für das Registrieren von JSPs am Rahmenwerk wird von den Erweiterungs-Bundles eine Klasse im JCommSy verwendet. Dieser Klasse könnte ein Mechanismus hinzugefügt werden, der prüft, ob ein URL-Teil schon vergeben ist, falls sich der Autor des Plugins nicht an bestimmte Konventionen hält (z.B. „`/jcommsy/PluginnameVersionDatum`“)

4.3.3 Erstellung eines Plugins nach der Umstellung

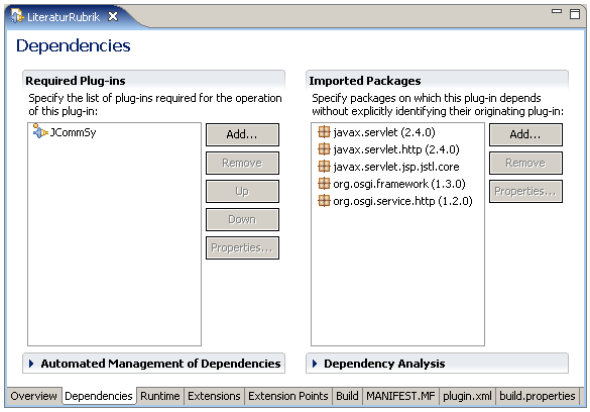
Zuerst wird ein neues Plugin-Projekt in Eclipse erstellt. In diesem Beispiel: `LiteraturRubrik`. Das Plugin wird mit der targetPlatform „`OSGi-Equinox` “angelegt:



Als Aktivator-Klasse wird `de.unihamburg.informatik.jcommsy.plugins.literaturrubrik.control.` benutzt.



Diese Klasse erweitert die Klasse `AbstractJCommSyActivator`. Als einzige Methode wird `setPlugin` überschrieben. In dieser Einschubmethode wird nur ein Exemplar der Klasse erzeugt, die im Erweiterungs-Bundle die Klasse `AbtractJCommsyPlugin` erweitert.



Als Required Plug-in wird das `JCommSy-Plugin` eingestellt, und folgende Packages müssen zusätzlich aus Standart-Bundles der OSGi-Umgebung importiert werden.

- `javax.servlet;version=2.4.0`,
- `javax.servlet.http;version=2.4.0`,
- `javax.servlet.jsp.jstl.core`,
- `org.osgi.framework;version=1.3.0`,
- `org.osgi.service.http;version=1.2.0`

Im Package `de.unihamburg.informatik.jcommsy.plugins.literaturrubrik.servlet` wird die Klasse `LiteraturServlet` angelegt, die von `AnstractJCommsyPluginServlet` erbt.

Ein Verzeichnis `/WebContent` wird im Projekt angelegt, in dem die JSPs und weitere Recourcen gespeichert werden.

Das Plugin kann jetzt mit dem JCommSy-Plugin gestartet werden. Es wird im JCommSy automatisch einer neuer Karteireiter erstellt, über den in die neue Rubrik gewechselt werden kann. Die neue Rubrik kann nun um beliebige Funktionalität erweitert werden.

4.3.4 Entwicklungsumgebung und Live-System

Entwicklungsumgebung

Für die Entwicklung von Erweiterungs-Bundles für das JCommSy wird Eclipse in der Version 3.2 benötigt. In dieser Umgebung können OSGi-Bundles direkt ausgeführt werden. Für die Installation dieser Bundles können *Team Project Sets* von Equinox genutzt werden. Durch einen Import dieser Dateien in Eclipse werden automatisch alle benötigten Bundles von Eclipse-CVS-System heruntergeladen. Diese Team Project Sets liegen der Arbeit auf DVD bei, und sind aktuell zum Zeitpunkt der Abgabe der Arbeit. Aktuellere Versionen finden sich auf den Webseiten des Equinox-Projektes. Zusätzlich zu diesen Bundles wird noch das aktuelle JCommSy-Bundle benötigt. Dies liegt als Eclipse-Projekt ebenfalls der Arbeit auf DVD

bei. Um während des Entwicklungsprozesses nicht nach jeder Änderung im Quelltext eine Installation der Bundles auf den Tomcat-Webserver vornehmen zu müssen, läuft das System während dieser Zeit direkt in Eclipse mit dem Jetty-Webserver.

Kapitel 5

Ausblick

In dieser Arbeit wurde die Java-Version des CommSy um eine Plugin-Schnittstelle erweitert. Es wurde dadurch die Grundlage geschaffen, um das JCommSy ohne Einblicke in die Implementierung um neue Funktionalitäten erweitern zu können. Auch eine weitergehende Aufteilung des JCommSy auf verschiedene Rubriken ist damit möglich. So könnten z.B. auch die bestehenden Rubriken als eigene Plugins ausgelagert werden. Folgende Schritte sind als Weiterentwicklungen der hier beschriebenen Grundlagen denkbar:

- Anmerkungen
- Netznavigation
- Datenbank
- Plugin-Rubrik

Anmerkungen

Anmerkungen sollen zu den Einträgen jeder Erweiterung möglich sein. Da in jeder Rubrik das Konzept der Anmerkungen dasselbe ist, scheint es sinnvoll diese Funktionalität im Haupt-Bundle zu halten. Anmerkungen könnten dann entweder automatisch jeder Erweiterungs-Rubrik hinzugefügt werden oder als einen frei wählbaren Service den Erweiterungen angeboten werden. Anmerkungen lassen sich auf diesem Weg wohl ähnlich wie im bisherigen JCommSy realisieren.

Netznavigation

Die Netznavigation zwischen den Rubriken des JCommSy und den Erweiterungsrubriken sowie zwischen Erweiterungsrubriken untereinander soll möglich sein. Auch diese Funktionalität sollte im Haupt-Bundle gehalten werden, da sie ebenfalls für jede Rubrik dieselbe ist und das Haupt-Bundle, gerade bei der Netznavigation von Erweiterungen untereinander, eine koordinierende Funktion einnimmt.

Datenbank

Die Speicherung von Einträgen der Erweiterungen soll möglich sein. Dazu könnte das Haupt-Bundle einen Service anbieten, da eine selbstständige Kontrolle innerhalb der Plugins hier dem Ziel widerspricht das JCommSy möglichst einfach erweitern zu können. In den Erweiterungen würde sonst zu viel Wissen über das System vorausgesetzt werden. Ohne die Datenbank bei der Neuinstallation eines Plugins verändern zu müssen, sollen die Einträge der Erweiterungen in der Datenbank gespeichert werden können.

Plugin-Rubrik

In einer Plugin-Rubrik können die installierten Bundles verwaltet werden und neue Bundles über ein Webformular dem System hinzugefügt werden.

Weiterer Ausblick

Im Rahmen der Diplomarbeit von Claus Andersen (siehe [1]) wurde das CommSy mit der Lernumgebung LAssi[1] verbunden. LAssi selbst baut als RCP-Anwendung auf Eclipse auf. Durch die Equinox-Umgebung, die nun auch im CommSy eingesetzt werden kann, lassen sich rein technische Bundles ohne Oberflächenfunktionalität evtl. in beiden Systemen einsetzen. So kann zum einen die Zusammenarbeit noch weiter ausgebaut werden, zum anderen könnten technische Bundles in beiden Systemen eingesetzt werden. Evtl. lassen sich auf diese Weise auch generell schon bestehende Plugins für RCP-Anwendungen im CommSy wiederverwenden.

[1] LAssi - *http://www.lassitools.org/*

Literaturverzeichnis

[1] Andersen, Claus: *Anschluss eines Rich-Clients an eine Web-Applikation mit Hilfe einer Service-Architektur*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 2006.

[2] Bronsard, Francois, Douglas Bryan, W. Kozaczynski, Edy S. Liongosari, Jim Q. Ning, Asgeir Olafsson und John W. Wetterstrand: *Toward software plug-and-play*. In: *SSR '97: Proceedings of the 1997 symposium on Software reusability*, Seiten 19–29, New York, NY, USA, 1997. ACM Press.

[3] Clayberg, Eric und Dan Rubel: *Eclipse. Building Commercial-Quality Plug-Ins*. the eclipse series. Addison Wesley, New York, 2 Auflage, 4 2006.

[4] *CommSy*, 2005. http://www.commsy.de.

[5] *CommSy BenutzerInnenhandbuch*, 2005. CommSy-Team.

[6] *Eclipse Foundation*, 2005. http://www.eclipse.org.

[7] *Eclipse Equinox*, 2005. http://www.eclipse.org/equinox.

[8] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*, 2004.

[9] Heiden, Markus: *Umstellung einer Web-Anwendung von direkter Datenbank-Ansprache mit der Verwendung eines O/R-Mapping-Frameworks am Beispiel von Hibernate*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, 2006.

[10] Hopkins, Jon: *Component primer*. Commun. ACM, 43(10):27–30, 2000.

[11] Kobryn, Cris: *Modeling components and frameworks with UML*. Commun. ACM, 43(10):31–38, 2000.

[12] Kubosch, Christof: *Softwareprojektunterstützung in CommSy*. Baccalaureatsarbeit, Universität Hamburg, Fachbereich Informatik, 2005.

[13] Lorenz, David H. und John Vlissides: *Designing components versus objects: a transformational approach*. In: *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, Seiten 253–262, Washington, DC, USA, 2001. IEEE Computer Society.

[14] *The OSGi Alliance*, 2005. http://www.osgi.org.

[15] Parnas, David. L.: *On the criteria to be used in decomposing systems into modules.* Commun. ACM, 15(12):1053–1058, 1972.

[16] Ran, Alexander: *Software isn't built from Lego blocks.* In: *SSR '99: Proceedings of the 1999 symposium on Software reusability*, Seiten 164–169, New York, NY, USA, 1999. ACM Press.

[17] Scharping, Arne: *Architekturregeln des JCommSy.* Baccalaureatsarbeit, Universität Hamburg, Fachbereich Informatik, 2005.

[18] Sreedhar, Vugranam C.: *York: programming software components.* In: *ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, Seiten 305–306, New York, NY, USA, 2001. ACM Press.

[19] Szyperski, Clemens: *Component Software.* Addison-Wesley, New York, zweite Auflage, 2002.

[20] The OSGi Alliance: *Core Specification.* OSGi Service Platform. The OSGi Alliance, 4 Auflage, 8 2005.

[21] The OSGi Alliance: *Service Compendium.* OSGi Service Platform. The OSGi Alliance, 4 Auflage, 8 2005.

[22] Züllighoven, Heinz: *Das objektorientierte Konstruktionshandbuch. Nach dem Werkzeug- und Materialansatz.* Dpunkt Verlag, Heidelberg, 1998. Korrigierter Nachdruck.

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

(Johannes Schultze)
Hamburg, den 23. Februar 2007