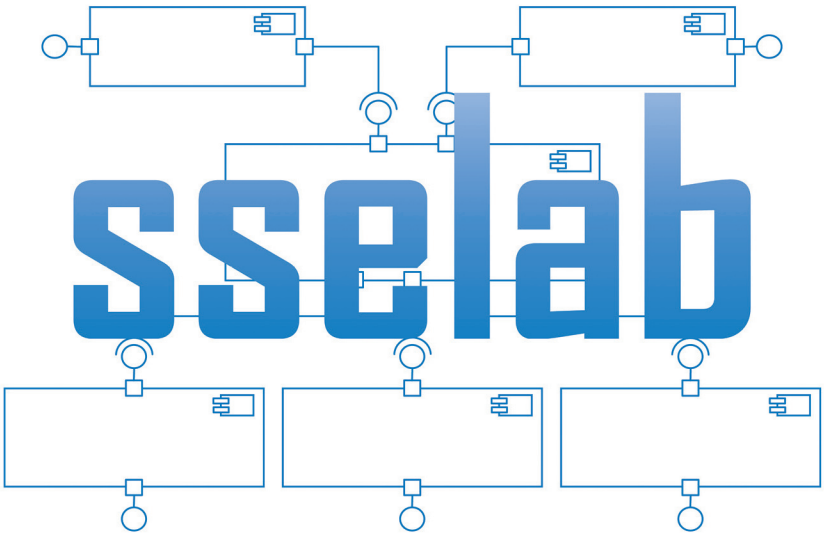
RWTH AACHEN

SE SOFTWARE ENGINEERING

Christoph Herrmann

Integrierte Software Engineering Services zur effizienten Unterstützung von Entwicklungsprojekten



Aachener Informatik-Berichte,
Software Engineering

Band 16

Hrsg: Prof. Dr. rer. nat. Bernhard Rumpe

Integrierte Software Engineering Services zur effizienten Unterstützung von Entwicklungsprojekten

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Dipl.-Inform. Christoph Herrmann
aus Braunschweig

Berichter: Universitätsprofessor Dr. rer. nat. Bernhard Rumpe
Universitätsprofessor Dr. rer. nat. Kurt Schneider

Tag der mündlichen Prüfung: 25. Juli 2013

[Her14] C. Herrmann:
Integrierte Software Engineering Services zur effizienten Unterstützung von Entwicklungsprojekten.
Shaker Verlag, ISBN 978-3-8440-2546-0. Aachener Informatik-Berichte, Software Engineering Band 16. 2014.
www.se-rwth.de/publications

Aachener Informatik-Berichte, Software Engineering

herausgegeben von
Prof. Dr. rer. nat. Bernhard Rumpe
Software Engineering
RWTH Aachen University

Band 16

Christoph Herrmann

Integrierte Software Engineering Services zur effizienten Unterstützung von Entwicklungsprojekten

Shaker Verlag
Aachen 2014

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Zugl.: D 82 (Diss. RWTH Aachen University, 2013)

Printed in Germany.

ISBN 978-3-8440-2546-0
ISSN 1869-9170

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • E-Mail: info@shaker.de

Kurzfassung

Werkzeuge sind ein essentieller Bestandteil von Software Engineering Projekten, da sie einzelne Entwickler und ganze Entwicklerteams bei der Durchführung ihrer vielfältigen und komplexen Aufgaben unterstützen und dadurch zu einer Steigerung der Produktivität beitragen. Werkzeuge zur Produktivitätssteigerung einzelner Entwickler werden vorwiegend in integrierten Entwicklungsumgebungen (engl. *integrated development environments*, IDEs) eingesetzt. Darüber hinaus haben sich in der letzten Zeit webbasierte Entwicklungsumgebungen zur Kollaboration (engl. *collaborative development environments*, CDEs) etabliert, die primär auf eine Verbesserung der Produktivität und Effizienz von Entwicklerteams abzielen.

Im Rahmen dieser Arbeit wird die Unterstützung von Software Engineering Projekten durch eine konsequent servicebasierte Nutzung integrierter Werkzeuge behandelt. Die wichtigsten Ergebnisse lassen sich wie folgt zusammenfassen.

- Basierend auf einer Analyse aktueller Trends und Herausforderungen im Bereich von IDEs und CDEs werden Konzepte erarbeitet, durch die eine servicebasierte Nutzung integrierter Werkzeuge unterstützt werden. Die Konzepte bilden die Grundlage für die Entwicklung der Architektur eines erweiterbaren und verteilten Frameworks, durch das eine Integration von Werkzeugen als Software as a Service (SaaS) sowie ein Zugriff auf deren Funktionen sowohl über desktop- als auch über webbasierte Clients möglich ist.
- Auf Grundlage der Konzepte zur Erweiterung und den zugehörigen Integrationsmethodiken des Frameworks werden mehrere Software Engineering Werkzeuge integriert. Neben server- und webbasierten Werkzeugen werden dabei insbesondere auch Werkzeuge gewählt, die ursprünglich nicht für die Ausführung in Serverumgebungen entwickelt worden sind. Aus den Erfahrungen, die bei der Integration der Werkzeuge gesammelt worden sind, werden Anforderungen abgeleitet, deren Erfüllung zu einer vereinfachten Integration führt.
- Zur Bildung von Instanzen des Frameworks wird das Deployment der Framework-Komponenten sowie der Services und der Clients methodisch behandelt. Auf Grundlage der flexiblen Verteilungsmöglichkeiten des Frameworks werden dabei auch verschiedene Betriebsarten und die a posteriori Integration von Werkzeugen betrachtet.
- Die Validierung der Konzepte des Frameworks basiert auf der Bildung einer konkreten Instanz unter `http://sselab.de`. Darin wurden ausgewählte Services installiert, durch die insbesondere verteilte Projekte mit agilen Vorgehensweisen unterstützt werden. Im Verlauf dieser Arbeit wurde diese Instanz erfolgreich für Projekte eingesetzt und dadurch die Anwendbarkeit der erarbeiteten Konzepte demonstriert.

Insgesamt sind damit Konzepte und darauf aufbauende eine Infrastruktur für eine flexible und leichtgewichtige Integration vielfältiger Werkzeuge sowie deren effektive servicebasierte Nutzung durch desktop- und webbasierte Clients entstanden.

Abstract

Tools are an essential part of software engineering projects. They support individual developers and entire development teams to accomplish their divers and complex tasks and thereby enhance their productivity. Tools that increase the productivity of individual developers are primarily used as part of integrated development environments (IDEs). In addition, web-based collaborative development environments (CDEs), which primarily aim at improving the productivity of developer teams, have become an integral part of software engineering projects in recent years.

This thesis examines the support of software engineering projects by a rigorous service-based utilization of integrated tools. The most important results can be summarized as follows.

- Based on an analysis of recent trends and challenges in the area of IDEs and CDEs, concepts for a service-based utilization of integrated tools are developed. The concepts lay the foundation for the creation of an architecture for an extensible and distributed framework. The framework allows integrating tools as software as a service (SaaS) as well as accessing their features both from desktop and from web-based clients.

- On the basis of the concepts for the extension and the corresponding integration methodologies of the framework, several software engineering tools are integrated. Besides server-based and web-based tools, especially tools that were not originally developed for an execution in server environments are selected. Furthermore, requirements are derived based on the experiences that have been gathered from the integration of the tools. The fulfillment of these requirements leads to a simplified integration.

- The methodology for the creation of framework instances is discussed based on the deployment of the framework components as well as the services and clients. The flexible distribution mechanisms of the framework allow different modes of operation, which are also discussed in addition to the a posteriori integration of tools.

- The validation of the framework concepts builds upon the creation of an instance under `http://sselab.de`. Selected services have been installed in this instance that especially support distributed projects with agile methodologies. In the course of this thesis this instance has been successfully employed for projects, so that the application of the elaborated concepts has been demonstrated.

Altogether, concepts and an infrastructure that allow a flexible and lightweight integration of diverse tools and support their effective service-based utilization from desktop- and web-based clients have been developed.

Danksagung

In der Zeit meiner Promotion haben mich viele Menschen unterstützt und dadurch zum erfolgreichen Abschluss dieser Dissertation beigetragen. An dieser Stelle möchte ich mich herzlich dafür bedanken.

Mein besonderer Dank gilt meinem Doktorvater Prof. Dr. Bernhard Rumpe für die Betreuung meiner Dissertation. Durch viele konstruktive Diskussionen, richtungsweisende Ratschläge und Verbesserungsvorschläge hat er mein Thema geformt und maßgeblich zum Gelingen der Promotion beigetragen. Auch für die vielfältigen und abwechslungsreichen Erfahrungen, die ich durch Lehrtätigkeiten sowie die Durchführung diverser Forschungs- und Industrieprojekte machen konnte, möchte ich ihm danken.

Weiterhin gilt mein Dank Prof. Dr. Kurt Schneider für die Übernahme des Zweitgutachtens und sein Interesse an meinem Thema. Insbesondere durch seine Bereitschaft, meine Dissertation noch vor der Abgabe durchzusehen, war es mir möglich, seine wertvollen Anmerkungen in die Arbeit einfließen zu lassen. Bei Frau Prof. Dr. Erika Ábrahám und Herrn Prof. Dr.-Ing. Stefan Kowalewski möchte ich mich bedanken, dass sie bereit waren, als Mitglieder der Promotionskommission die Prüfung durchzuführen.

Allen Mitarbeitern des Instituts für Software Systems Engineering der TU Braunschweig und des Lehrstuhls für Software Engineering der RWTH Aachen möchte ich danken: für eine erlebnisreiche Zeit mit vielen interessanten Diskussionen und Projekten sowie für ein tolles Arbeitsklima. Insbesondere bei Thomas Kurpick möchte ich mich für die vielen hilfreichen Gespräche und auch die praktische Hilfe bei der Umsetzung des SSELabs bedanken. Auch René Stumm und allen weiteren Kollegen und Studenten, die mich bei der Entwicklung und dem Betrieb tatkräftig unterstützt haben, gilt mein Dank. Hervorheben möchte ich auch alle Kollegen, die sich bereit erklärt haben, Teile der Arbeit zu lesen: Markus Look, Thomas Kurpick, Pedram Mir Seyed Nazari, Andreas Horst, Dimitri Plotnikov, Alexander Roth und Lars Hermerschmidt. Vielen Dank für die Bereitschaft und alle hilfreichen Anmerkungen. Claas Pinkernell hat mir insbesondere in den ersten Monaten als Mitarbeiter bei der synavision GmbH den Freiraum ermöglicht, den ich für die Fertigstellung der Dissertation benötigt habe. Vielen Dank dafür.

Herzlicher Dank gilt auch meiner Familie und meinen Freunden, insbesondere meinen Eltern, die mich in allen Phasen meines Lebens mit viel Zeit und Energie gefördert haben und immer für mich da waren. Ganz besonders möchte ich mich bei meiner Ehefrau Dörte bedanken, die mich zu jeder Zeit liebevoll unterstützt und immer wieder motiviert hat, auch wenn das hieß, dass ich viele Samstage an der Dissertation gearbeitet und nicht meine Freizeit mit ihr verbracht habe. Dadurch hat sie erheblich zur Fertigstellung dieser Arbeit beigetragen.

Schließlich möchte ich meinem himmlischen Vater danken, ohne den ich diese Dissertation nicht erfolgreich hätte abschließen können und der meinem Leben und damit auch dieser Arbeit überhaupt den Sinn gibt.

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Kontext der Arbeit . . . 1
- 1.2 Trends im Bereich von Entwicklungsumgebungen . . . 3
- 1.3 Herausforderungen für Entwicklungsumgebungen . . . 7
- 1.4 Wichtigste Ziele und Ergebnisse . . . 9
- 1.5 Aufbau der Arbeit . . . 10

2 Technische Grundlagen der Werkzeugunterstützung im Software Engineering **13**
- 2.1 Einsatz von Werkzeugen in Software Engineering Projekten . . . 13
  - 2.1.1 Integration von Werkzeugen . . . 13
  - 2.1.2 Werkzeuge als Services . . . 16
- 2.2 Anwendungsentwicklung mit erweiterbaren Architekturen . . . 18
  - 2.2.1 Komponenten . . . 18
  - 2.2.2 Frameworks . . . 20
  - 2.2.3 Plug-ins . . . 22
- 2.3 Zusammenfassung . . . 23

3 Nutzungsszenarien mit ihren Rollen und Anforderungen **25**
- 3.1 Nutzungsszenarien . . . 25
  - 3.1.1 Softwareentwicklung in einem agilen verteilten Team . . . 25
  - 3.1.2 Erstellung eines wissenschaftlichen Papiers . . . 27
  - 3.1.3 Integration eines Transformationswerkzeugs . . . 29
  - 3.1.4 Erweiterung der Kapazität einer SSELab-Instanz . . . 29
  - 3.1.5 Interaktion externer Werkzeuge mit dem SSELab . . . 30
- 3.2 Rollen . . . 30
- 3.3 Anforderungen . . . 32
  - 3.3.1 Anforderungen an die SSELab-Instanz aus Sicht der Endanwender . . . 32
  - 3.3.2 Anforderungen an die Services aus Sicht der Endanwender . . . 36
  - 3.3.3 Anforderungen an die SSELab-Instanz aus Sicht der Administratoren . . . 38
  - 3.3.4 Anforderungen an das SSELab-Framework . . . 40
- 3.4 Zusammenfassung . . . 42

4 Architektur eines Frameworks für webbasierte Projektportale und CDEs **43**
- 4.1 Modellierung der Domäne von Projektportalen . . . 43
  - 4.1.1 Services . . . 44
  - 4.1.2 Personen . . . 46

4.1.3 Projekte . . . 46
4.2 Übersicht über die Architektur . . . 48
4.2.1 Komponenten des SSELab-Frameworks . . . 48
4.2.2 Erweiterbarkeit der Komponenten . . . 50
4.2.3 Verteilung der Komponenten . . . 50
4.3 Verwaltung von Services in den Backends . . . 51
4.3.1 Architektur der Backends . . . 51
4.3.2 Funktionalität der Backends . . . 54
4.4 Verwaltungsmechanismen im Frontend . . . 60
4.4.1 Architektur des Frontends . . . 61
4.4.2 Service- und Backend-Verwaltung . . . 62
4.4.3 Kontenverwaltung . . . 70
4.4.4 Projektverwaltung . . . 75
4.4.5 Client-Verwaltung . . . 79
4.5 Architektur und Funktionsumfang der Clients . . . 80
4.5.1 Architektur der Client-APIs . . . 82
4.5.2 Realisierte Clients . . . 83
4.6 Zusammenfassung . . . 85

5 Methodik zur Erweiterung des Frameworks 87
5.1 Unterstützung neuer Service-Kategorien . . . 88
5.1.1 Bereitstellung der Backend-Infrastruktur . . . 89
5.1.2 Entwicklung servicespezifischer Funktionalität . . . 93
5.1.3 Strategien zur effektiveren Unterstützung neuer Service-Kategorien . . . 94
5.2 Integration von Services . . . 95
5.2.1 Gemeinsamkeiten von Service-Plug-ins . . . 96
5.2.2 Integration von Base-Services . . . 101
5.2.3 Integration von Ostp-Services . . . 108
5.2.4 Integration von Social-Services . . . 113
5.3 Entwicklung und Erweiterung von Framework-Clients . . . 120
5.3.1 Entwicklung von Framework-Clients . . . 120
5.3.2 Erweiterung des Eclipse-Clients . . . 122
5.4 Zusammenfassung . . . 123

6 Integrierte Werkzeuge und Clients 125
6.1 Übersicht der integrierten Werkzeuge . . . 125
6.1.1 Base-Services . . . 125
6.1.2 Ostp-Services . . . 127
6.1.3 Social-Services . . . 128
6.2 Integration repräsentativer Werkzeuge . . . 129
6.2.1 Subversion . . . 129
6.2.2 Trac . . . 134
6.2.3 Jenkins . . . 137
6.2.4 Feedback-System . . . 140

6.2.5 MontiCore . . . 142
6.2.6 WikiBot . . . 146
6.2.7 Google . . . 149
6.3 Aktualisierung der integrierten Werkzeuge . . . 150
6.3.1 Base-Services . . . 150
6.3.2 Ostp-Services . . . 151
6.3.3 Social-Services . . . 151
6.4 Anforderungen an Werkzeuge zur vereinfachten Integration . . . 151
6.4.1 Anforderungen an Base-Services . . . 152
6.4.2 Anforderungen an Ostp-Services . . . 154
6.4.3 Anforderungen an Social-Services . . . 155
6.5 Zusammenfassung . . . 156

7 Methodik zur Bildung einer Instanz des Frameworks 159
7.1 Verteilung der Komponenten des Frameworks . . . 159
7.1.1 Betriebsarten einer Framework-Instanz . . . 161
7.1.2 A posteriori Integration von Werkzeugen . . . 163
7.2 Laufzeitumgebungen der Komponenten des Frameworks . . . 164
7.2.1 Laufzeitumgebung von Backend-Instanzen . . . 164
7.2.2 Laufzeitumgebung von Frontend-Instanzen . . . 166
7.2.3 Sicherheitsaspekte . . . 167
7.3 Methodik des Deployments . . . 168
7.3.1 Deployment von Backend-Instanzen . . . 168
7.3.2 Deployment von Frontend-Instanzen . . . 170
7.3.3 Deployment von Services . . . 172
7.3.4 Deployment von Clients . . . 173
7.4 Zusammenfassung . . . 175

8 Nutzungs- und Betriebskonzept der sselab.de-Instanz 177
8.1 Unterstützte Projektarten . . . 179
8.1.1 GloSE . . . 179
8.1.2 Energie Navigator . . . 179
8.1.3 SensorCloud . . . 180
8.1.4 Weitere Projekte . . . 180
8.2 Einführung in das Nutzungskonzept der Instanz . . . 180
8.2.1 Erste Schritte der Nutzung für Gäste und Benutzer . . . 181
8.2.2 Überblick der Funktionen für Administratoren . . . 186
8.2.3 Nutzung der Services durch externe Werkzeuge . . . 188
8.3 Betriebskonzept der Instanz . . . 189
8.4 Zusammenfassung . . . 191

9 Verwandte Arbeiten 193
9.1 SourceForge, GForge, FusionForge und Davenport . . . 193
9.2 Jazz . . . 194

9.3 Projektserver Proanvil 198
9.4 DrProject 200
9.5 ProGET und PicoLibre 201

10 Zusammenfassung und Ausblick **203**
10.1 Zusammenfassung 203
10.2 Ausblick 205

Literaturverzeichnis **207**

Abbildungsverzeichnis **225**

Quellcodeverzeichnis **229**

Tabellenverzeichnis **231**

A Glossar **233**

B Notationen **239**

C Kennzahlen **241**

D Lebenslauf **245**

Kapitel 1

Einführung

In Software Engineering Projekten arbeitet eine Vielzahl von Experten zusammen, um komplexe Software oder softwareintensive Systeme zu realisieren. Der effektive Einsatz von Werkzeugen zur Unterstützung individueller Entwickler und ganzer Entwicklerteams wie auch die Integration der Werkzeuge in kohärente Entwicklungsumgebungen sind deshalb aktuelle Forschungsgebiete. Werkzeuge, die die Produktivität einzelner Entwickler verbessern, werden heutzutage üblicherweise in integrierten Entwicklungsumgebungen (engl. *integrated development environments*, IDEs) kombiniert. In letzter Zeit haben sich darüber hinaus Entwicklungsumgebungen zur Kollaboration (engl. *collaborative development environments*, CDEs) etabliert, die die Produktivität von Entwicklerteams verbessern.

Im Rahmen dieser Arbeit werden aktuelle Trends und Herausforderungen im Bereich von IDEs und CDEs analysiert und auf dieser Grundlage Konzepte für die Integration von Software Engineering Werkzeugen sowie die Architektur eines darauf aufbauenden webbasierten Frameworks erarbeitet. Die Integration der Werkzeuge erfolgt dabei in einer Serverumgebung, so dass diese primär als Dienstleistung, also als Software as a Service (SaaS) angeboten werden. Durch die Migration desktopbasierter Werkzeuge in eine Serverumgebung können sowohl Synergieeffekte durch die zentrale Integration produziert, als auch zukünftige Trends, wie beispielsweise der Einsatz von Web-IDEs unterstützt werden.

Dieses erste Kapitel gibt einen Überblick über die vorliegende Arbeit. In den folgenden Abschnitten werden zunächst der Kontext und die Motivation der Arbeit diskutiert. Danach werden aktuelle Trends und wesentliche Herausforderungen im Bereich von Entwicklungsumgebungen aufgezeigt, die die Grundlage für die erarbeiteten Konzepte bilden. Anschließend werden die wichtigsten Ziele und Ergebnisse dieser Arbeit kompakt dargestellt. Zuletzt wird eine Übersicht über den Aufbau der Arbeit gegeben.

1.1 Kontext der Arbeit

Software ist heutzutage ein integraler Bestandteil einer wachsenden Anzahl von Systemen und durchdringt zunehmend alle Bereiche des täglichen Lebens [BM11]. Die Komplexität und der Umfang der Softwareanteile steigen dabei kontinuierlich an, wie es beispielsweise anhand moderner Fahrzeuge beobachtet werden kann [BKPS07]. Dadurch wird die Entwicklung von Software und softwareintensiven Systemen immer anspruchsvoller und der erfolgreiche Abschluss von Projekten – d. h. die rechtzeitige, kostengünstige und insbesondere qualitativ hochwertige Fertigstellung von Softwareprodukten – stellt eine große Herausforderung dar. Obwohl die Ermittlung

der tatsächlichen Zahl gescheiterter und erfolgreicher Entwicklungsprojekte kontrovers diskutiert wird [Gla05, EV10], können heutzutage noch immer eine relativ große Anzahl von Projekten nicht erfolgreich beendet werden [EK08, BM11].

Um Projekte erfolgreich abschließen und die steigende Komplexität im Software Engineering beherrschen zu können, müssen viele Experten unterschiedlicher Domänen in allen Phasen eines Projekts zusammenarbeiten. Die Entwicklung von Softwaresystemen ist demzufolge ein von Kollaboration geprägter Prozess [Lan09], in dem alle Beteiligten ihre Arbeit koordinieren und auf vielen Ebenen miteinander kommunizieren müssen [Whi07]. Neben der Komplexität der zu entwickelnden Systeme stellt daher die Kollaboration in Entwicklerteams eine weitere Herausforderung dar, die den Erfolg oder Misserfolg eines Projekts maßgeblich beeinflussen kann. Insbesondere durch den Trend zu global verteilten Projekten wird der Einfluss effektiver Kollaboration noch intensiviert. Einerseits können durch die verteilte Entwicklung Vorteile entstehen [AFHOC08]. Andererseits wird durch die geographische, zeitliche und soziokulturelle Distanz der Entwicklerteams die Effektivität der Kollaboration entscheidend beeinträchtigt [BBH+09, AFH+05].

Die Komplexität der Systeme und die Kollaboration in großen und verteilten Teams sind jedoch nicht die einzigen Herausforderungen in Software Engineering Projekten. Auch die vielfältigen Aktivitäten einzelner Entwickler müssen optimal unterstützt werden, um einen erfolgreichen Projektverlauf zu gewährleisten. Die Aufgaben der Entwickler reichen von codezentrierten Aktivitäten – wie Programmierung, Refactoring und Qualitätssicherung – über Modellierung und Prädiktion essentieller Eigenschaften bis hin zu Managementaktivitäten [Zel07]. Durch diese verschiedenen Aktivitäten und die daraus resultierenden häufigen Kontextwechsel entstehen Reibungsverluste, die die Effektivität der Entwickler, der Teams und damit des gesamten Projekts negativ beeinflussen [BB03]. Insgesamt stellen die Komplexität der Systeme, die Kollaboration in Projekten sowie die Koordination von Entwicklern und deren Aktivitäten große Herausforderungen für heutige Projekte dar.

Im Software Engineering hat der Einsatz von Werkzeugen, um Methoden zu unterstützen, Sprachen zu verarbeiten, die Komplexitätssteigerung der Systeme zu beherrschen, effektive Kollaboration zu ermöglichen und die Entwickler in ihren vielfältigen Aktivitäten zu unterstützen, eine lange Tradition [Gru94, HOT00, Zel07]. Im Laufe der Zeit wurden dementsprechend diverse Werkzeuge entwickelt, die die meisten Tätigkeiten in der Softwareentwicklung abdecken [HOT00]. Dazu gehören sowohl einzelne Werkzeuge wie Compiler, Generatoren und Editoren als auch integrierte oder webbasierte Entwicklungsumgebungen und serverbasierte Werkzeuge wie Continuous Integration Server. Dieser Trend wird sich voraussichtlich in der Zukunft fortsetzen, so dass alle Tätigkeiten, die vollständig oder teilweise automatisiert werden können, auch durch ein entsprechendes Werkzeug unterstützt werden [Zel07]. Dies führt auch zu einer Diversifizierung und Spezialisierung der Werkzeuge, die in Projekten eingesetzt werden. Insgesamt gilt daher die Aussage, dass der aktuelle Stand der Werkzeuge aufbauend auf den zugrunde liegenden Prozessen, Methoden und Sprachen den Stand der Technik im Software Engineering widerspiegelt [Zel07].

Werkzeuge sind demnach ein essentieller Bestandteil von Entwicklungsprojekten, da sie die Produktivität einzelner Entwickler und ganzer Teams steigern [BMJH96]. Wie diese Produktivitätssteigerung jeweils durch ein Werkzeug unterstützt wird, ist dabei unterschiedlich. Einige

Werkzeuge vereinfachen oder automatisieren bestimmte Aufgaben und verbessern so direkt die Produktivität der Entwickler. Andere Werkzeuge verbessern dagegen indirekt die Produktivität, indem sie Informationen für die Entwickler bereitstellen, die sie für ihre aktuellen Aufgaben benötigen [Rei96]. Der Einsatz von Werkzeugen führt jedoch nicht automatisch zu einer Produktivitätssteigerung. Vielmehr müssen die Werkzeuge unter anderem in Abhängigkeit von der Teamgröße und der genutzten Prozesse ausgewählt werden und selbst eine hohe Qualität besitzen, damit überhaupt eine Produktivitätssteigerung erreicht werden kann. Andernfalls kann sogar eine Produktivitätsminderung eintreten [BMJH96].

Über die angemessene Auswahl einzelner Werkzeuge und deren individuelle Qualität hinaus hat auch insbesondere die Integration der Werkzeuge Auswirkungen auf die Produktivität der Entwickler [Zel07]. Werkzeuge die nicht integriert sind, können zwar in einer koordinierten Art und Weise bzw. gemäß einem Prozess oder einer Methode verwendet werden, bieten jedoch selbst keine direkte Unterstützung dafür [HOT00]. Erst durch eine Integration kann eine optimale Unterstützung erreicht werden. Dementsprechend ist die Integration von Werkzeugen seit vielen Jahren ein aktives Forschungsgebiet im Bereich des Software Engineering [WD07]. Aufgrund der komplexen und damit kostenintensiven Integration, Weiterentwicklung und Anpassung von Werkzeugen [HOT00] ist auch heutzutage die Integration gängiger Entwicklungswerkzeuge noch immer nicht adäquat [PVEP10]. Eine Ausnahme bilden dabei gegebenenfalls Integrationslösungen einzelner Hersteller. Aber diese Lösungen basieren oftmals auf proprietären Datenformaten und forcieren dadurch eine Herstellerabhängigkeit, die erhebliche Nachteile mit sich bringt [PVEP10].

Im Bereich der Entwicklungsumgebungen wird zwischen der Integration von Werkzeugen für einzelne Entwickler und der Integration von Werkzeugen für Entwicklerteams unterschieden [BB03]. Im ersten Fall spricht man von einer IDE und im zweiten von einer CDE oder auch von einem Projektportal. IDEs werden schon seit langer Zeit in der Softwareentwicklung eingesetzt [Zel07]. CDEs haben sich dagegen erst in den letzten Jahren etabliert. Der Begriff CDE wurde erstmals von Booch und Brown in [BB03] eingeführt. Gemäß [BB03, Boo07] ist eine CDE eine Entwicklungsumgebung, in der alle Projektbeteiligten gemeinsam arbeiten können und durch die viele alltägliche und nicht-kreative Aufgaben eliminiert bzw. automatisiert und die Kommunikation innerhalb eines Teams angeregt werden. CDEs unterscheiden sich von einzelnen Kollaborationswerkzeugen unter anderem durch die webbasierte Integration [BB03]. Weiterhin stellen Booch und Brown fest, dass IDEs zwar einige Kollaborationsfunktionen – wie beispielsweise die Synchronisation mit einem Versionsverwaltungssystem – bieten können, aber das eine IDE durch das Hinzufügen solcher Funktionen nicht automatisch zu einer CDE wird [BB03].

1.2 Trends im Bereich von Entwicklungsumgebungen

IDEs und CDEs sind zu essentiellen Bestandteilen heutiger Software Engineering Projekte geworden [Zel07, CW09]. Dadurch unterliegen sie einer kontinuierlichen Evolution, um einzelne Entwickler und Entwicklerteams immer effektiver unterstützen zu können. In diesem Abschnitt werden einige aktuelle Trends im Bereich von Entwicklungsumgebungen diskutiert, die deren Design maßgeblich beeinflussen.

Plug-in-Systeme und Plug-in-Architekturen

Plug-ins und Plug-in-Systeme werden in einigen Anwendungsarten schon seit langer Zeit eingesetzt. Klassische Beispiele dafür sind Web-Browser, Bildverarbeitungsprogramme und Texteditoren [Bir05]. Im Web-Browser Netscape wurde beispielsweise die Verarbeitung der unterstützten Grafikformate jeweils durch ein Plug-in realisiert [Mar01]. Aktuelle Web-Browser können über solche internen Verarbeitungsmechanismen hinaus durch die Benutzer mit Hilfe von Plug-ins erweitert werden. Für den Web-Browser Firefox ist beispielsweise eine Vielzahl sogenannter Add-ons verfügbar [Moz12]. Adobe Photoshop ist ein Beispiel eines Bildverarbeitungsprogramms, dessen Funktionalität durch Plug-ins an die Bedürfnisse der Benutzer angepasst werden kann. Ein frühes Beispiel für einen erweiterbaren Texteditor ist EMACS [Sta81], zu dessen Laufzeit Skripte eingebunden werden können. Die Erweiterungen durch diese Skripte können dabei so umfangreich sein, dass EMACS auch als IDE bezeichnet und genutzt wird [Cur02]. Aktuellere IDEs basieren in vielen Fällen auch auf Plug-in-Systemen oder Plug-in-Architekturen. Ein prominentes Beispiel dafür ist die Eclipse IDE [dRW04].

Plug-in-Systeme bestehen aus einer Host-Anwendung, die Kernfunktionalität in einer bestimmten Domäne bereitstellt [Mar01]. Plug-ins können beim Start oder zur Laufzeit der Host-Anwendung geladen werden und diese um Funktionalität ergänzen. Die Host-Anwendung muss dabei für die Integration von Plug-ins entwickelt worden sein und entsprechende Schnittstellen zur Verfügung stellen. Die Schnittstellen definieren, wie ein Plug-in aufgebaut sein muss, damit es integriert werden kann und welche Funktionalität die Host-Anwendung den Plug-ins zur Verfügung stellt [MV03, Mar01]. Eine weitere wesentliche Eigenschaft von Plug-in-Systemen ist, dass die Plug-ins zur Kompilierzeit der Host-Anwendung nicht verfügbar sein müssen, so dass keine spezifischen Plug-ins in der Host-Anwendung bekannt sind [MMS03]. Die Plug-ins selbst sind keine eigenständigen Anwendungen, sondern können nur durch die Host-Anwendung ausgeführt und orchestriert werden [Bir05].

Stellt die Host-Anwendung nur eine minimale Laufzeitumgebung bereit und ist alle weitere Funktionalität durch Plug-ins realisiert, spricht man von einer reinen Plug-in-Architektur [Bir05]. Neben den zuvor beschriebenen klassischen Plug-in-Systemen haben sich in letzter Zeit auch reine Plug-in-Architekturen etabliert [Bir05, Rat11]. Die Architektur der Eclipse IDE basiert beispielsweise auf einer reinen Plug-in-Architektur. Seit der Veröffentlichung von Eclipse 3.0 besteht der Kern aus einer minimalen OSGi-Laufzeitumgebung [WHKL08], der die Plug-ins verwaltet, durch die alle weitere Funktionalität zur Verfügung gestellt wird [GHM+05].

Während in der Vergangenheit Plug-in-Systeme und -Architekturen vorwiegend in Desktop-Anwendungen verwendet worden sind, beispielsweise für die oben beschriebenen Anwendungsarten oder zur Entwicklung von Rich-Client-Anwendungen, werden sie in der letzten Zeit zunehmend auf der Serverseite eingesetzt. Ein Beispiel dafür ist das Plug-in-Framework Plux, das ursprünglich für dynamisch rekonfigurierbare Desktop-Anwendungen konzipiert worden ist, jetzt jedoch auch für die Entwicklung von Web-Anwendungen genutzt wird [JWLM13, JWM10]. Ein Beispiel im Bereich von Entwicklungsumgebungen ist IBM Jazz [Fro07]. Die Client- und die Serverkomponenten von Jazz basieren jeweils auf Eclipse-Technologie, so dass auf der Serverseite ein Plug-in-System zum Einsatz kommt. Der Trend, Plug-in-Systeme für Entwicklungsumgebungen auch auf der Serverseite einzusetzen, erfordert, dass die benötigten Entwicklungswerkzeuge und -artefakte auch auf einem Server verfügbar sind.

Software as a Service, Hosted Services und webbasierte Systeme

CDEs und webbasierte Projektportale sind heutzutage wichtige Bestandteile der Entwicklungsinfrastrukturen von Projekten. Gemäß der Definition in [BB03] ist eine CDE nicht ein einzelnes allumfassendes Werkzeug, sondern eine Sammlung „einhundert kleiner Sachen“, also mehrerer Anwendungen, die auf Grundlage von Prozessen oder Methoden kombiniert genutzt werden. Booch und Brown stellen darüber hinaus fest, dass die Anwendungen, welche die Kerninfrastruktur für ein Projekt bereitstellen, wie beispielsweise Versionsverwaltungssysteme, als Hosted Services betrieben werden sollten. Andere Anwendungen, insbesondere solche die höhere Anforderungen an die Interaktivität stellen, sollten dagegen als Desktop-Anwendungen genutzt werden [BB03].

Unter anderem durch Fortschritte in den Bereichen der Server-Virtualisierung [BDF+03] und des Cloud Computings [AFG+10] besteht jedoch in letzter Zeit ein Trend darin, auch Desktop-Anwendungen vermehrt als Hosted Services bzw. Software as a Service (SaaS) anzubieten [Jaz07, BHL08]. Beispielsweise bietet Google klassische Desktop-Anwendungen wie E-Mail und Office-Produkte als Service an. Beispiele aus dem Bereich der Entwicklungsumgebungen sind aktuelle webbasierte IDEs wie Eclipse RAP [Lan08], Adinda [vDMC+10], WebIDE [DJCD11] und Ace [ACE12], die versuchen, Rich-Client-Anwendungen mit Browser-basierten Clients zu ersetzen. Entgegen der Aussagen von [BB03] zeichnet sich also ab, dass auch Anwendungen mit hohen Anforderungen an die Interaktivität vermehrt als Hosted Services betrieben werden. Insgesamt zeigt sich, dass dadurch verstärkt webbasierte Systeme in allen Phasen des Entwicklungszyklus von Software Engineering Projekten eingesetzt werden [Whi07, CW09].

Dieser Trend in Richtung Hosted Services kann jedoch nicht nur im Bereich einzelner Anwendungen sondern auch im Bereich vollständiger Desktop-Systeme beobachtet werden. Beispielsweise stellt die Verwendung virtueller Maschinen für die Software-Entwicklung sicher, dass alle Entwickler mit einer konsistenten Umgebung arbeiten und diese nicht erst kosten- und zeitintensiv aufsetzen müssen [BASB+10]. Solche virtuellen Maschinen können entweder auf den lokalen Rechnern der Entwickler oder auf Servern betrieben werden.

Einige Vorteile von Hosted Services, die zu diesem Trend beitragen, sind, dass das Deployment neuer Versionen einer Anwendung relativ leicht erfolgen kann [Whi07] und dementsprechend neue Funktionalität und Bug-Fixes in kleinen und inkrementellen Schritten ausgeliefert werden können [Jaz07, CW09]. Darüber hinaus wird die Administration durch die Zentralisierung erleichtert [Whi07], so dass insgesamt die Kosten für den Betrieb und die Wartung der Anwendungen gesenkt werden können [CW09]. Ein weiterer Vorteil ist die Möglichkeit, Legacy-Werkzeuge in moderne Werkzeugketten einzubetten. Dies ist insbesondere im industriellen Kontext von großer Bedeutung, da dort Legacy-Anwendungen weit verbreitet sind [LEPV10]. Weiterhin können Funktionen von Hosted Services den Benutzern nicht nur über eine Web-Oberfläche, sondern auch als Web-Service bereitgestellt werden, so dass ein höherer Grad an Automatisierung erreicht werden kann [Jaz07].

Insgesamt zeichnet sich ab, dass in Zukunft eine steigende Anzahl von Entwicklungswerkzeugen entweder vollständig oder teilweise auf Servern laufen werden. Trotz dieses Trends haben die oben genannten webbasierten IDEs noch nicht den Funktionsumfang lokaler Entwicklungsumgebungen erreicht, so dass Desktop-IDEs noch immer vorrangig eingesetzt und noch nicht vollständig von webbasierten Systemen ersetzt werden. Daher gelten die Aussagen von [BB03]

und [Whi07] auch heute noch, dass zukünftige Projekte eine Kombination aus webbasierten und desktopbasierten Werkzeugen nutzen werden. Das Verhältnis wird sich voraussichtlich jedoch immer mehr in Richtung Hosted Services verschieben.

Integration von IDEs und CDEs

Durch die weite Verbreitung von Desktop-IDEs und dem zunehmenden Einsatz webbasierter CDEs ist die Integration von IDEs und CDEs ein weiterer aktueller Trend. Die Autoren von [CW09] stellen fest, dass zum Zeitpunkt ihrer Analyse die Integration von IDEs und CDEs mit einigen Ausnahmen nur rudimentär vorhanden war. Als Grund dafür nennen sie, dass diese Idee noch nicht eine kritische Masse erreicht habe und behaupten, dass die Integration von IDEs und CDEs bzw. Projektportalen in den nächsten Jahren zunehmen werde.

Ein Beispiel einer solchen Integration von IDE und CDE ist Mylyn [KM06]. Mylyn ist ein Plug-in für Eclipse, das unter anderem Informationen aus Issue-Tracking-Systemen wie Trac [Tra12], Bugzilla [Bug12] oder Jira [Jir12] verarbeitet und so eine aufgabenbasierte Verwendung von Eclipse ermöglicht. Dazu speichert Mylyn für jede Aufgabe, die an ein Ticket gekoppelt ist, unter anderem die benötigten Artefakte, Editoren und Views in Eclipse und kann so bei einem Aufgabenwechsel den entsprechenden Kontext wiederherstellen. Ein weiteres Beispiel ist die Integration von sogenannter Awareness-Funktionalität in IDEs, mit deren Hilfe beispielsweise die Tätigkeiten anderer Entwickler oder Änderungen an Artefakten in Echtzeit oder zeitnah angezeigt werden können. In [CGL09] beschreiben die Autoren wie sie Jazz erweitert haben, um Group-Awareness zusätzlich zu den bereits vorhandenen Mechanismen zur Presence- und Workspace-Awareness hinzuzufügen.

In verteilten Entwicklerteams kann durch den Einsatz von CDEs der negative Einfluss der Distanz auf die Produktivität gemindert und effektive Kollaboration gefördert werden [NWD08]. Gemäß [Whi07] ist darüber hinaus die enge Kopplung webbasierter und desktopbasierter Entwicklungsumgebungen ein weiterer wichtiger Mechanismus, um die Kollaboration in Software Engineering Projekten zu verbessern. Insgesamt werden in Zukunft weitere Kollaborationswerkzeuge entwickelt und in CDEs integriert werden [LEPV10]. Durch die weite Verbreitung von IDEs werden auch Integrationskomponenten für diese Werkzeuge entstehen und dadurch die Kopplung zwischen IDEs und CDEs weiter ausgebaut werden. Das zuvor beschriebene Beispiel der Erweiterung von Jazz deutet auch auf den folgenden Trend hin, nämlich der Nutzung von Social Media im Software Engineering.

Web 2.0 Anwendungen und Social Software

Unter dem Begriff Web 2.0 werden Anwendungen und Technologien zusammengefasst, bei denen die Interaktion und Kollaboration aller Nutzer im Vordergrund stehen [Mur07]. Einige Beispiele dafür sind Wikis, Blogs, RSS-Feeds und Tagging. Social Software ist dagegen ein allgemeinerer Begriff, der für Anwendungen gebraucht wird, die die Interaktion und Kommunikation von Benutzergruppen ermöglichen [Bäc06]. Dies reicht von klassischer E-Mail bis hin zu sozialen Netzwerken, so dass der Begriff Social Software sowohl Kollaborationsanwendungen des Web 1.0, wie beispielsweise Chats, Foren und Mailing-Listen, als auch Web 2.0 Anwendungen umfasst [ACGL08].

Der gesamte Bereich des Web 2.0 und der Social Software entwickelt sich in einem rasanten Tempo weiter, so dass in der Forschung noch viele Fragestellungen offen sind und erst behandelt werden müssen [Chi08]. Insbesondere ist der Einsatz dieser Anwendungen in Software Engineering Projekten noch nicht ausreichend erforscht [STvDC10]. Unabhängig davon wurde Social Software in Entwicklungsprojekten schon immer intensiv eingesetzt [AJLN08, STvDC10]. Beispielsweise wird Social Software in verteilten Entwicklerteams verwendet, um die in [AFH+05] beschriebene Distanz zwischen den verteilten Teams zu verringern und die informelle Interaktion zu fördern [STvDC10] sowie Vertrauen und persönliche Beziehungen zwischen den Mitgliedern der verteilten Teams aufzubauen [CGL09]. Dies soll unter anderem dazu führen, dass agile Methoden [BBvB+12] auch in verteilten Projekten erfolgreich eingesetzt werden können [ACGL08].

Web 2.0 Anwendungen und Funktionen sozialer Netzwerke werden bereits in vielen CDEs und Projektportalen angeboten [CW09, LEPV10] und sind insbesondere in Open Source Projekten weit verbreitet [LEPV10]. Dementsprechend wird in Zukunft vermehrt Social Software in Projekten eingesetzt und in die Entwicklungsumgebungen integriert werden [AJLN08, STvDC10]. CDEs werden dabei projektrelevante Daten aus verschiedenen Quellen aggregieren [CW09], so dass diese durch die zuvor beschriebene Integration von IDEs und CDEs in lokalen Entwicklungsumgebungen genutzt werden können.

1.3 Herausforderungen für Entwicklungsumgebungen

Neben den im vorhergehenden Abschnitt skizzierten Trends existieren weitere Herausforderungen, die die Konzeption und die Realisierung von Entwicklungsumgebungen maßgeblich beeinflussen. Diese Herausforderungen werden im Folgenden zusammengefasst.

In Software Engineering Projekten wurde schon immer eine große Menge an Werkzeugen und Technologien eingesetzt [Gru94], da gerade im Bereich der Werkzeuge Neuerungen mit einer hohen Geschwindigkeit entwickelt und vermarktet worden sind [HOT00]. Wie im vorigen Abschnitt beschrieben, gibt es heutzutage eine ähnlich schnelle Entwicklung im Bereich von Social Software [STvDC10]. Eine Herausforderung besteht dementsprechend darin, flexibel auf sich ändernde Anforderungen reagieren und neue heterogene Werkzeuge, Anwendungen und Technologien in kurzer Zeit in Entwicklungsumgebungen integrieren zu können.

Neben der Integration neuer Werkzeuge ist die Integration bestehender Werkzeugketten ein wichtiger Aspekt bei der Konzeption von Entwicklungsumgebungen. Die Autoren von [LEPV10] stellen fest, dass moderne CDEs diesen Aspekt nicht ausreichend berücksichtigen und daher aufgrund von Legacy-Werkzeugen oft nicht für den Einsatz in Unternehmen geeignet sind. Darüber hinaus verwenden Entwicklerteams bestimmte Werkzeuge oftmals aus historischen oder persönlichen Gründen und haben ihre bevorzugten Werkzeuge, die eventuell sogar nicht optimal für die Teams geeignet sind. Dennoch werden diese Werkzeuge weiter genutzt, da der Aufwand und die Kosten für die Evaluation neuer Werkzeuge den Nutzen übersteigen oder weil einfach eine Abneigung gegen die Einführung neuer Werkzeuge vorhanden ist [Sar05]. Die Integration existierender Werkzeuge ist dementsprechend essentiell, aber auch herausfordernd, da diese Werkzeuge oftmals keine Mechanismen für eine a posteriori Integration zur Verfügung stellen [ADS02].

Eine weitere Herausforderung für Entwicklungsumgebungen ist, dass viele Werkzeuge schon eigene Integrationsmechanismen anbieten. Ein Beispiel dafür ist die zuvor beschriebene Integration von Mylyn in Eclipse sowie der Zugriff auf Trac. Trac selbst bietet eine Integration mit gängigen Versionsverwaltungssystemen wie beispielsweise Subversion. Diese vorhandenen und heterogenen Integrationsmechanismen bieten einen Mehrwert für die Benutzer und sollten auch nach der Integration in eine Entwicklungsumgebung erhalten bleiben. Das gilt auch dann, wenn nur eines der Werkzeuge in die Entwicklungsumgebung integriert ist und die anderen unabhängig davon genutzt werden.

Bei der Integration verschiedener Werkzeuge muss immer ein Kompromiss zwischen der Funktionalität und der Langlebigkeit der Integrationslösung gefunden werden. Auf der einen Seite kann eine enge Integration von Werkzeugen Synergieeffekte produzieren, die den Funktionsumfang und damit den Wert der Integrationslösung erhöhen. Auf der anderen Seite ist eine enge Integration mit einem hohen Aufwand für die Wartung und die Aktualisierung verbunden, so dass die Langlebigkeit der Integrationslösung beeinträchtigt wird. Diese Abhängigkeit von Funktionsumfang und Langlebigkeit wird in [ADDK03, WD07] diskutiert und ist in Abbildung 1.1 schematisch dargestellt.

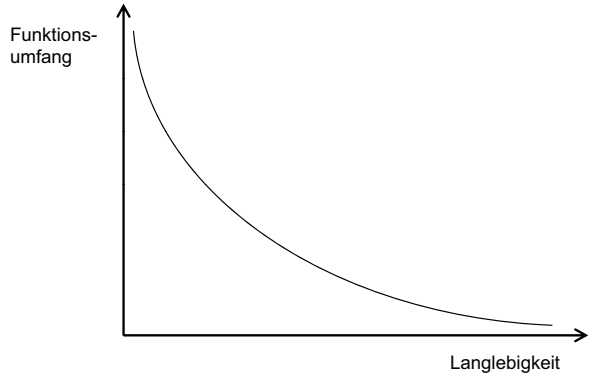


Abbildung 1.1: Abhängigkeit von Funktionsumfang und Langlebigkeit von Integrationslösungen (adaptiert aus [WD07])

Im Bereich von Entwicklungsumgebungen existiert diese Abhängigkeit zwischen Funktionalität und Langlebigkeit ebenfalls und stellt eine weitere Herausforderung dar. Einerseits sollte sich eine Entwicklungsumgebung auf die eigentlichen Integrationsaspekte beschränken [ADDK03]. Andererseits sind eine nahtlose Integration [BB03, Sar05] und übergeordnete Funktionen, wie beispielsweise eine zentrale Suche oder Tagging, über alle integrierten Werkzeuge wichtig für die Akzeptanz der Benutzer [KK09]. Eine nahtlose Integration und übergeordnete Funktionen erfordern jedoch unter Umständen eine Anpassung der integrierten Werkzeuge, falls diese nicht entsprechende Schnittstellen bereitstellen. Dies führt wiederum zu einem deutlich erhöhten Wartungs- und Aktualisierungsaufwand.

1.4 Wichtigste Ziele und Ergebnisse

Die in den vorigen Abschnitten beschriebenen Trends und Herausforderungen sind wesentliche Einflussfaktoren und Rahmenbedingungen bei dem Entwurf von Entwicklungsumgebungen. Abbildung 1.2 stellt diesen Sachverhalt noch einmal schematisch dar.

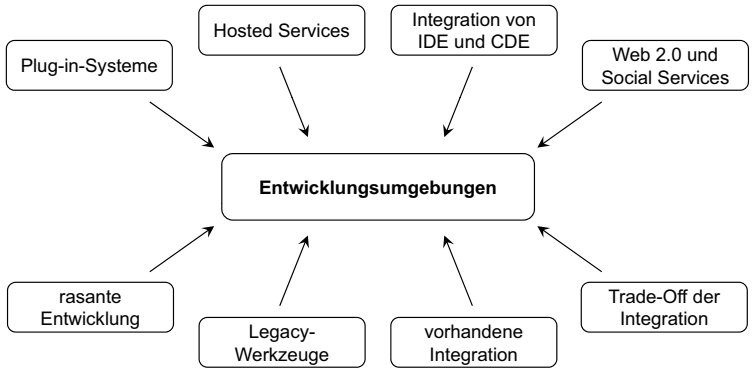


Abbildung 1.2: Trends und Herausforderungen im Bereich von Entwicklungsumgebungen

Ausgehend von der Analyse dieser Trends und Herausforderungen wurde die Forschungsfrage, die das Kernziel dieser Arbeit beschreibt, folgendermaßen formuliert.

> *Wie können Werkzeuge als Services bereitgestellt und in Entwicklungsumgebungen integriert werden, um verteilte und agile Software Engineering Projekte effektiv zu unterstützen?*

Der Fokus dieser Arbeit liegt also einerseits auf der Erarbeitung von Konzepten zur Migration und Integration von Werkzeugen in eine Serverumgebung und andererseits auf der Bereitstellung dieser Werkzeuge für konkrete Projekte. Insgesamt können die wichtigsten Ziele und Ergebnisse der vorliegenden Arbeit wie folgt zusammengefasst werden.

- Entwicklung von Konzepten für eine servicebasierte Nutzung integrierter Werkzeuge auf Grundlage aktueller Trends und Herausforderungen im Bereich von Entwicklungsumgebungen. Darauf aufbauend wird die Architektur eines erweiterbaren und verteilten Frameworks – das SSELab-Framework – für webbasierte Projektportale und CDEs konzipiert. Das Framework ermöglicht eine Integration von Werkzeugen als Software as a Service sowie einen Zugriff auf deren Funktionen über desktop- und webbasierte Clients.
- Integration mehrerer Werkzeuge auf Grundlage der Erweiterungsmechanismen und den zugehörigen Integrationsmethodiken des Frameworks. Neben server- und webbasierten Werkzeugen werden dabei insbesondere auch Werkzeuge gewählt, die ursprünglich für die Ausführung auf lokalen Maschinen und nicht in Serverumgebungen konzipiert worden sind. Bei der Integration wird dabei ein leichtgewichtiger Ansatz verfolgt, so dass die Werkzeuge möglichst wenig angepasst werden müssen.

- Ableitung von Anforderungen an Entwicklungswerkzeuge aus den Erfahrungen bei der Instanziierung des Frameworks, deren Erfüllung zu einer vereinfachten Integration von Werkzeugen in webbasierte CDEs und Projektportale führt.
- Beschreibung einer Methodik zur Bildung von Instanzen des SSELab-Frameworks. Dabei werden sowohl das Deployment der Framework-Komponenten als auch der Services und der Clients behandelt. Außerdem werden die a posteriori Integration von Werkzeugen sowie verschiedene Betriebsarten, die auf den Verteilungsmöglichkeiten des Frameworks basieren, betrachtet.
- Validierung der Konzepte des Frameworks durch die Bildung einer Instanz unter `http://sselab.de` und deren Verwendung in zahlreichen, teilweise industriellen Projekten. Die Instanziierung erfolgt durch ein Deployment des Frameworks und die Installation ausgewählter Werkzeuge, die insbesondere verteilte Projekte mit agilen Vorgehensweisen unterstützen. Durch den erfolgreichen Einsatz dieser Instanz konnte die Anwendbarkeit der erarbeiteten Konzepte demonstriert werden.

Die Begriffe (SSELab-)Framework und (SSELab-)Instanz werden in dieser Arbeit immer dann verwendet, wenn Eigenschaften des Frameworks bzw. einer Instanz des Frameworks im Vordergrund stehen. Der Begriff SSELab wird gebraucht, wenn beide Sichtweisen eingeschlossen werden.

1.5 Aufbau der Arbeit

Diese Arbeit ist wie folgt aufgebaut.

Kapitel 1 *Einführung*

Das erste Kapitel gibt eine Übersicht über den Kontext der Arbeit, beschreibt aktuelle Trends im Bereich von IDEs und CDEs und diskutiert Herausforderungen bei der Konzeption und Realisierung von Entwicklungsumgebungen. Außerdem werden die wichtigsten Ziele und Ergebnisse der vorliegenden Arbeit zusammengefasst.

Kapitel 2 *Technische Grundlagen der Werkzeugunterstützung im Software Engineering*

Dieses Kapitel beschreibt als Grundlage dieser Arbeit die Unterstützung von Software Engineering Projekten durch Werkzeuge. Es werden sowohl die Integration von Werkzeugen als auch die Nutzung von Werkzeugen als Services diskutiert. Darüber hinaus werden mehrere Ansätze zur Anwendungsentwicklung mit erweiterbaren Architekturen eingeführt und in den Kontext der Arbeit gestellt.

Kapitel 3 *Nutzungsszenarien mit ihren Rollen und Anforderungen*

Dieses Kapitel gibt eine Übersicht über die wichtigsten Anforderungen an ein Framework für webbasierte Projektportale und CDEs sowie an eine Instanz des Frameworks, die auf agile, verteilte Entwicklungsprojekte zugeschnitten ist. Die Anforderungen basieren auf mehreren Nutzungsszenarien, die zu Beginn des Kapitels eingeführt werden.

Kapitel 4 *Architektur eines Frameworks für webbasierte Projektportale und CDEs*

In diesem Kapitel werden die Konzepte und die wesentlichen Elemente der Architektur aller Komponenten des SSELab-Frameworks auf Grundlage der im vorhergehenden Kapitel diskutierten Architekturtreiber beschrieben.

Kapitel 5 *Methodik zur Erweiterung des Frameworks*

Dieses Kapitel behandelt die Methodik zur Erweiterung des realisierten Frameworks. Dabei wird detailliert auf die Konzepte zur Erweiterung des Frameworks, auf die Integration von Werkzeugen sowie auf die Realisierung von Clients eingegangen.

Kapitel 6 *Integrierte Werkzeuge und Clients*

In diesem Kapitel wird zunächst eine Übersicht über alle im Rahmen dieser Arbeit integrierten Werkzeuge gegeben. Darauf aufbauen wird die Integration einiger ausgewählter Werkzeuge genauer diskutiert. Abschließend werden Anforderungen aufgestellt, die zu einer vereinfachten Integration von Werkzeugen führen.

Kapitel 7 *Methodik zur Bildung einer Instanz des Frameworks*

Dieses Kapitel erläutert die Bildung einer Instanz des Frameworks. Dazu werden auf Grundlage der Verteilungsmöglichkeiten des Frameworks zuerst verschiedene Betriebsarten sowie die a posteriori Integration von Werkzeugen behandelt. Anschließend werden die Laufzeitumgebung der Komponenten des Frameworks sowie die Methodik des Deployments erläutert.

Kapitel 8 *Nutzungs- und Betriebskonzept der sselab.de-Instanz*

Aufbauend auf dem vorigen Kapitel beschreibt dieses Kapitel die Instanz des Frameworks, die unter `http://sselab.de` gebildet worden ist. Dabei wird sowohl auf das Nutzungskonzept für Benutzer und Administratoren sowie auf die gewählte Betriebsart eingegangen.

Kapitel 9 *Verwandte Arbeiten*

In diesem Kapitel erfolgt eine Diskussion und Abgrenzung der Konzepte des SSELab-Frameworks von einigen wichtigen verwandten Arbeiten.

Kapitel 10 *Zusammenfassung und Ausblick*

Dieses Kapitel schließt die Arbeit mit einer Zusammenfassung der Ergebnisse ab und diskutiert mögliche weiterführende Arbeiten, die auf dem vorgestellten Ansatz aufbauen könnten.

Kapitel 2

Technische Grundlagen der Werkzeugunterstützung im Software Engineering

In diesem Kapitel werden grundlegende Aspekte diskutiert, die das Verständnis der Inhalte aller weiteren Kapitel ermöglichen. Da der Fokus dieser Arbeit die effiziente Unterstützung von Software Engineering Projekten durch integrierte Services ist, werden im folgenden Abschnitt zunächst die Integration von Werkzeugen und danach deren servicebasierte Nutzung behandelt. Sowohl die Integration als auch die Bereitstellung der Werkzeuge als Services erfolgt im Rahmen dieser Arbeit über ein erweiterbares und verteiltes Framework. In diesem Kapitel werden daher auch Grundlagen von Frameworks und von zwei weiteren relevanten Ansätzen zur Erstellung erweiterbarer Architekturen beschrieben.

2.1 Einsatz von Werkzeugen in Software Engineering Projekten

Der effektive Einsatz von Werkzeugen ist ein wichtiger Erfolgsfaktor von Software-Entwicklungsprojekten [HOT00]. Dementsprechend wurden in der Forschung bereits diverse Aspekte der Werkzeugunterstützung im Software Engineering diskutiert. In den folgenden Abschnitten werden die für die vorliegende Arbeit relevanten Grundlagen bezüglich der Integration von Werkzeugen und deren Nutzung als Services betrachtet.

2.1.1 Integration von Werkzeugen

Die Integration von Werkzeugen in Entwicklungsumgebungen ist ein Forschungsgebiet, das bereits seit den 1980er Jahren und auch noch heute in der Literatur behandelt wird, so dass eine umfangreiche Menge von Veröffentlichungen entstanden ist [WD07, Wic06]. Im Laufe der Jahre wurden sowohl für den Begriff Werkzeugintegration als auch für den Begriff Entwicklungsumgebung unterschiedliche Bezeichnungen und Definitionen verwendet. Entwicklungsumgebungen wurden beispielsweise als Integrated Project-Support Environment (IPSE) [TN92], Computer-Aided Software Engineering Environment (CASEE) [Ear90], Software Engineering Environment (SEE) [BP92] oder Programming Environment [Zel07] bezeichnet. Im Rahmen dieser Arbeit werden hauptsächlich die beiden Begriffe Integrated Development Environment (IDE) und Collaborative Development Environment (CDE) verwendet, um Umgebungen für einzelne Entwickler

bzw. Umgebungen für Entwicklerteams zu beschreiben. Der Begriff Entwicklungsumgebung wird immer dann verwendet, wenn sowohl IDEs als auch CDEs gemeint sind.

Die verschiedenen Definitionen für den Begriff Werkzeugintegration (engl. tool integration) [Was90, TN92, BP92] werden im Rahmen dieser Arbeit wie folgt zusammengefasst. Die Integration von Werkzeugen beschreibt die Gemeinsamkeiten und die Beziehungen von Werkzeugen. Dies bezieht sich unter anderem auf die genutzten Datenformate, auf Konventionen für Benutzeroberflächen und auf die Verwendung gemeinsamer Funktionen. Das Ziel beim Einsatz von integrierten Werkzeugen ist die effektive Unterstützung von Prozessen, Methoden und Techniken in allen Phasen der Softwareentwicklung [Ste87], so dass eine Produktivitätssteigerung einzelner Entwickler und Entwicklerteams sowie eine Qualitätssteigerung der Produkte erreicht werden kann [WD07].

Im Forschungsgebiet der Werkzeugintegration wurden zwei grundlegende Arbeiten im Jahr 1990 von Wasserman [Was90] und Earl [Ear90] veröffentlicht, auf denen viele der in den folgenden Jahren publizierten Arbeiten aufbauen. Die wesentlichen Beiträge dieser beiden Veröffentlichungen werden im Folgenden kurz zusammengefasst.

Wasserman beschreibt in [Was90] ein Klassifikationsschema für Integrationslösungen, indem er die folgenden fünf Dimensionen der Werkzeugintegration definiert.

Platform Integration beschreibt die Umgebung bzw. Plattform und deren angebotene Dienste, die von den integrierten Werkzeugen genutzt werden können.

Presentation Integration beschreibt die Gemeinsamkeiten von Werkzeugen bezüglich des Erscheinungsbildes ihrer Benutzeroberflächen sowie der Bedienungsparadigmen („Look-and-feel“).

Data Integration bezieht sich sowohl auf den Austausch von und den Zugriff auf Daten über gemeinsame Datenstrukturen als auch auf die Verwaltung der Beziehungen zwischen Daten verschiedener Datenstrukturen.

Control Integration gibt an, inwieweit die integrierten Werkzeuge Funktionen anderer Werkzeuge aufrufen bzw. sich gegenseitig Benachrichtigungen über Ereignisse senden können.

Process Integration beschreibt die Unterstützung und die Rolle der integrierten Werkzeuge innerhalb eines Entwicklungsprozesses.

Insgesamt können mit Hilfe dieser Dimensionen die Gemeinsamkeiten integrierter Werkzeuge ermittelt werden. Aufbauend auf den Arbeiten von Wasserman wurden im Laufe der Zeit sowohl weitere Klassifikationsschemata definiert als auch das Schema von Wasserman verfeinert. Thomas und Nejmeh [TN92] beschreiben beispielsweise eine Erweiterung der Integrationsdimensionen von Wasserman, indem sie zusätzliche Eigenschaften definieren, bei denen die Beziehungen zwischen jeweils zwei integrierten Werkzeugen im Vordergrund stehen. In [ABEKT11] werden auf Grundlage der Integrationsdimensionen unter anderem Metriken für den Grad einer Integrationslösung definiert.

In der zweiten grundlegenden Arbeit zur Werkzeugintegration beschreibt Earl als eine technische Lösung das sogenannte Toaster-Referenzmodell [Ear90], das in Abbildung 2.1 schematisch

dargestellt ist. In diesem Referenzmodell wird die Integrationsumgebung in mehrere Schichten aufgeteilt, die jeweils zusammengehörige Funktionen bzw. Services kapseln und den darauf aufbauenden Schichten anbieten. Die Services werden in der folgenden Auflistung kurz zusammengefasst.

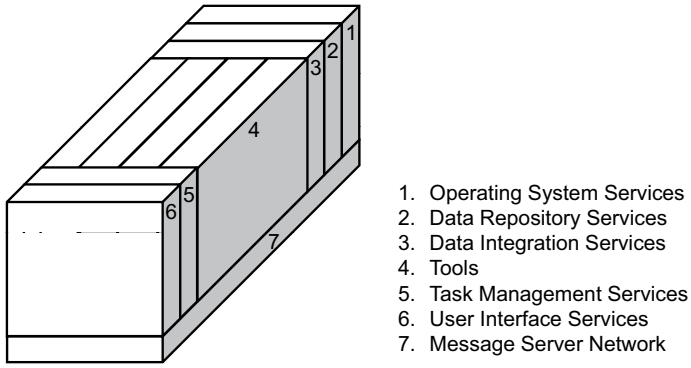


Abbildung 2.1: Toaster-Referenzmodell (adaptiert aus [Ear90])

1. Operating System Services sind grundlegende Betriebssystem- und Netzwerkdienste, die den anderen Schichten als betriebssystemunabhängige Funktionen angeboten werden.

2. Data Repository Services ermöglichen die Verwaltung von Daten und deren Beziehungen untereinander. Dazu gehören die Speicherung von Daten sowie der Zugriff auf Daten.

3. Data Integration Services bieten den übrigen Schichten Zugriff und Manipulationsmöglichkeiten für die Daten sowie von Metadaten an. Dazu gehört unter anderem ein Query-Service, über den Anfragen formuliert werden können.

4. Tools sind gemäß der Definition von [Ear90] eine bestimmte automatisierte Funktion oder Aktivität, die ein Entwickler ausführen bzw. durchführen muss.

5. Task Management Services abstrahieren von den integrierten Werkzeugen und bieten die Möglichkeit, spezifische Aktivitäten oder Prozesse auszuführen, ohne dabei die benötigten Werkzeuge einzeln aufrufen zu müssen.

6. User Interface Services dienen dazu, die Funktionalität der integrierten Werkzeuge den Entwicklern über eine konsistente Benutzeroberfläche zu präsentieren. Daraus folgt, dass die Funktionalität der Werkzeuge unabhängig von einer konkreten Benutzeroberfläche angeboten werden muss.

7. Message Server Network stellt die Verbindung aller Schichten her und ermöglicht die Registrierung von Werkzeugen im Framework sowie die Auslieferung von Nachrichten zwischen den Werkzeugen bzw. den Schichten.

Das Referenzmodell von Earl gibt keine konkreten Technologien oder Entwicklungsmethoden vor. Es erfordert jedoch, dass alle Werkzeuge konform zu der Schichtenarchitektur entwickelt werden, damit überhaupt eine Integration möglich ist. Dadurch können ausschließlich Werkzeuge integriert werden, die dem Modell entsprechen [WD05]. Aufbauend auf der Arbeit von Earl wurden im Laufe der Zeit mehrere Realisierungen des Modells vorgeschlagen. Zwei Beispiele dafür sind SoftBench [FW93] und ToolNet [ADS02, ADDK03].

Trotz dieser beiden frühen grundlegenden und der darauf aufbauenden Arbeiten ist auch heute die Integration von Werkzeugen noch immer nicht optimal [PVEP10, ABEKT11]. Einer der Gründe dafür ist die in Kapitel 1 beschriebene rasante Entwicklung im Bereich der Werkzeuge. Durch die große Zahl und die Heterogenität der Werkzeuge entstehen im Zuge einer Integration üblicherweise Inseln der Automatisierung, also einzelne Gruppen integrierter Werkzeuge [ABEKT11], die sogar auf unterschiedlichen Integrationsansätzen basieren können. Gemäß Wicks und Dewar ist diese Herangehensweise an eine Werkzeugintegration jedoch keineswegs ineffektiv. Basierend auf einer ausführlichen Literaturrecherche [Wic06] behaupten sie in [WD07, WD05], dass in der Forschung oftmals Werkzeugintegration aus rein technischer Sicht betrachtet wird und dabei soziale und ökonomische Aspekte weitestgehend ignoriert werden. Sie weisen beispielsweise darauf hin, dass bei der Integration von Werkzeugen die Erfahrung eines Unternehmens bzw. der Endnutzer berücksichtigt werden sollte [WD05] und eine Integration nur dann sinnvoll ist, wenn sie durch konkrete Anforderungen getrieben wird [WD07, WD05]. Ein weiterer Grund für fehlende optimale Integrationslösungen ist der ebenfalls in Kapitel 1 beschriebene Trade-Off zwischen der Funktionalität und der Langlebigkeit gemäß [ADDK03, WD07]. Die Umsetzung und Wartung einer aus Benutzersicht optimalen Integrationslösung erfordert erheblichen Aufwand und verursacht hohe Kosten [ABEKT11], was bei der schnellen Entwicklung im Bereich der Werkzeuge üblicherweise nicht gerechtfertigt werden kann.

Insgesamt kann für den Bereich der Werkzeugintegration festgehalten werden, dass die Vielfalt und die Heterogenität sowohl der Werkzeuge als auch der Integrationslösungen in absehbarer Zeit bestehen bleiben oder noch weiter zunehmen werden. Im Rahmen dieser Arbeit wurden daher eine inkrementelle sowie eine möglichst leichtgewichtige Integration der Werkzeuge angestrebt. Das Ziel war dabei, den Aufwand für die Integration und die Wartung möglichst gering zu halten und vorhandene Integrationsmechanismen bzw. verfügbare Inseln der Automatisierung zu erhalten und – im Gegensatz zum Referenzmodell von Earl – keine Konformität der Werkzeuge zu dem konzipierten Framework zu forcieren. Außerdem wurden sowohl bei der Wahl der zu integrierenden Werkzeuge als auch bei der Entwicklung des Integrationsframeworks die Anforderungen der Endnutzer berücksichtigt.

2.1.2 Werkzeuge als Services

Software as a Service (SaaS) beschreibt ein Nutzungs- bzw. ein Auslieferungskonzept von Software, bei dem ein Anbieter dedizierte Anwendungen auf einer Serverinfrastruktur als Dienstleistung bereitstellt und dessen Kunden über das Internet darauf zugreifen [Jac05, BHL08, CK09]. Durch dieses Konzept wird der Besitz von der Nutzung der Software getrennt [TBB03]. Obwohl SaaS heutzutage üblicherweise mit Cloud Computing in Verbindung gebracht wird, existierte das Konzept bereits in den frühen Jahren der Informatik.

SaaS war bereits Mitte 1960 bis Anfang 1980 weit verbreitet [CK09]. Damals wurde Rechenzeit und auch Software auf zentralen Mainframe-Maschinen von Unternehmen der sogenannten Time-Sharing Industrie zur Miete angeboten. Diese Ressourcen konnten von Kunden entweder über Batch-Verfahren oder interaktiv genutzt werden [CKGS08]. Damals wurde unter anderem der Begriff „utility computing“ verwendet, der auch heute wieder im Kontext des Cloud Computing gebräuchlich ist [AFG+10]. Die gesamte Time-Sharing Industrie brach schließlich Anfang 1980 durch die zunehmende Verbreitung des Personal Computers (PC) zusammen [CK09], da die bisherigen Kunden eigene PCs anschafften und daher nicht mehr die Dienstleistung der Time-Sharing Unternehmen nutzten. Zwei Gründe für diese Entwicklung waren, dass PCs einerseits günstiger als die Anmietung von Rechenzeit oder Software und andererseits deutlich einfacher zu warten waren.

Im Laufe der 1990er Jahre wurde SaaS wieder populär. Ein wesentlicher Faktor war, dass die Rechnerinfrastrukturen erheblich komplexer geworden und dadurch schwieriger zu warten waren. In dieser Zeit wurden Unternehmen, die Anwendungen als Dienstleistung anboten, als Application Service Provider (ASP) bezeichnet [GTHM01]. Salesforce.com, das 1999 gegründet wurde, war eines der ersten Unternehmen, die deutliche Erfolge im SaaS-Sektor mit dem Angebot von Enterprise Software as a Service aufweisen konnten [CK09]. Insgesamt war jedoch das ASP-Modell zu dieser Zeit nicht so erfolgreich wie vorhergesagt, so dass der Begriff Application Service Provider heutzutage nicht mehr gebräuchlich ist. Ein wesentlicher Grund für den fehlenden Erfolg war, dass die Architektur der damaligen Systeme noch nicht ausreichend skalierte [Got08]. Dies führte letztendlich dazu, dass immer wieder neue Architekturstile im Kontext von SaaS entwickelt und genutzt worden sind, wie beispielsweise Grid-Computing [TBB03], serviceorientierte Architekturen (SOA) [LZV08] und schließlich Cloud Computing [AFG+10]. Insgesamt kann also die Bereitstellung von Software als Service sowohl mit Cloud-Infrastrukturen als auch mit herkömmlichen Serverinfrastrukturen erfolgen, da SaaS ein Ansatz für die Auslieferung beschreibt, der unabhängig davon ist, welche Konzepte der Architektur zugrunde liegen und auf welcher Plattform die Software läuft.

Wie in Abschnitt 1.2 diskutiert, werden in Software Engineering Projekten zunehmend Entwicklungswerkzeuge als Services im Kontext von Projektportalen und CDEs genutzt. Diese Entwicklung wurde maßgeblich durch Open Source Projekte beeinflusst [BB03, Rob05], die anfänglich einzelne Werkzeuge wie Versionsverwaltungssysteme und Mailing-Listen zur Koordination von Projekten eingesetzt haben. Später wurden die Infrastrukturen um weitere Werkzeuge ergänzt, wie beispielsweise Bug Tracking Systeme und Wikis, die letztendlich zu deren Integration in Projektportalen und CDEs geführt haben [Rob05]. SourceForge [SF12] – eines der ersten Projektportale – wurde beispielsweise auf Grundlage einer Analyse der Praktiken von Open Source Projekten entwickelt, bei der unter anderem die Bedeutung webbasierter Werkzeuge hervorgehoben wurde [ABS02]. Durch den Erfolg von Open Source Projekten im Allgemeinen und SourceForge und vergleichbaren Portalen im Speziellen wurden vermehrt CDEs und Projektportale entwickelt. Dabei wurden vorwiegend webbasierte Anwendungen als Services angeboten, die gemäß [BB03] die Kerninfrastruktur von Entwicklungsprojekten bilden.

Mittlerweile haben sich CDEs und Projektportale mit ihren server- und webbasierten Anwendungen als grundlegende Bestandteile von Projektinfrastrukturen etabliert [CW09]. Darüber hinaus werden auch zunehmend Werkzeuge, die üblicherweise nicht als Websystem konzipiert

worden sind, als Services angeboten. Aktuell werden in der Forschung beispielsweise die in Kapitel 1 erwähnten webbasierten IDEs behandelt [LNK+12, AAE+11, vDMC+10]. Im Rahmen dieser Arbeit wird dieser Trend im Software Engineering aufgegriffen und ein Framework konzipiert, das sowohl die Integration etablierter als auch bisher nicht als Services genutzter Werkzeuge unterstützt. Dieses Framework wird als Framework für webbasierte Projektportale und CDEs bezeichnet und ist dementsprechend eine SaaS-Plattform für die Erstellung von Projektportalen.

2.2 Anwendungsentwicklung mit erweiterbaren Architekturen

Die rasante Entwicklung im Bereich der Entwicklungswerkzeuge führt zu einer Reihe von Anforderungen an die Architektur von Plattformen, in die Werkzeuge integriert werden sollen. Eine der wichtigsten Anforderungen ist dabei die Verwendung einer erweiterbaren Architektur, so dass eine effiziente Integration neuer Werkzeuge erreicht werden kann.

Im Software Engineering existieren verschiedene Ansätze, die die Erweiterbarkeit der Architektur von Anwendungen ermöglichen. Dieser Abschnitt fasst drei verwandte und kooperierende Ansätze zusammen, die im Kontext dieser Arbeit relevant sind. Zunächst werden komponentenbasierte Softwarearchitekturen, danach Frameworks und schließlich Plug-in-Architekturen diskutiert. Diese Ansätze bilden die Grundlage für die Architektur des SSELab-Frameworks, auf die in Kapitel 4 detailliert eingegangen wird.

2.2.1 Komponenten

Bereits im Jahr 1968 wurde der Einsatz von Komponenten im Software Engineering von McIlroy vorgeschlagen, um eine Wiederverwendung von vorgefertigten Implementierungen als Black-Box zu ermöglichen [McI68]. Heutzutage ist die komponentenbasierte Softwareentwicklung [Szy02] ein wichtiger Ansatz zur Entwicklung komplexer Systeme. Die Motivation für diesen Ansatz basiert darauf, dass einerseits die Neuentwicklung von Systemen zu kostspielig ist und andererseits der Einsatz von konfigurierbarer Standardsoftware oftmals nicht alle Anforderungen adäquat abdeckt [Szy02]. Die komponentenbasierte Entwicklung stellt daher einen Kompromiss dar, bei dem einzelne unabhängig voneinander entwickelte und qualitätsgesicherte Komponenten zu einem Gesamtsystem kombiniert werden [BW98].

Im Laufe der Jahre wurden verschiedene Definitionen für den Begriff Softwarekomponente vorgeschlagen [BW98]. Eine weit verbreitete Definition ist in [Szy02] angegeben. Darin wird eine Komponente als eine Kompositionseinheit definiert, die ihre Abhängigkeiten und Schnittstellen explizit angibt und durch ein unabhängiges Deployment durch Dritte mit anderen Komponenten kombiniert werden kann. In [BW98] und [Szy02] werden noch einige weitere Definitionen diskutiert, die andere Aspekte in den Vordergrund stellen. Diese Definitionen sind jedoch im Kontext dieser Arbeit nicht relevant und werden nicht weiter betrachtet.

Bei der komponentenbasierten Entwicklung werden ein gesamtes System oder auch einzelne Komponenten aus existierenden Komponenten erzeugt [Szy03]. Dadurch entsteht eine hierarchische Struktur, wie durch Abbildung 2.2 illustriert wird. Dieses Vorgehen resultiert in einer Entkopplung des Entwicklungsprozesses der Komponenten von dem Entwicklungsprozess eines

kompositional entwickelten Systems [CCL06]. Bei der Entwicklung der Komponenten stehen die Wiederverwendung und die Qualitätssicherung im Vordergrund. Bei der Entwicklung des Systems dagegen die Auswahl und Komposition existierender Komponenten. Die Komposition eines Systems oder einer Komponente erfordert üblicherweise die Entwicklung oder Generierung von Glue-Code, durch den die Komponenten verbunden werden [CCL06]. Der Glue-Code sollte dabei im optimalen Fall keine Adaption der Komponenten, sondern ausschließlich die Komposition realisieren [Szy03].

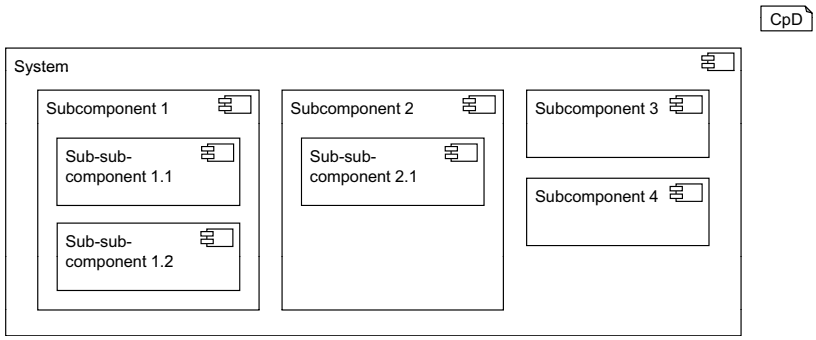


Abbildung 2.2: Hierarchische Struktur eines komponentenbasierten Systems

Die kompositionale Entwicklung von Softwaresystemen besitzt einige charakteristische Eigenschaften und bietet mehrere Vorteile [Vö11], von denen einige im Folgenden zusammengefasst werden.

Modularität: Die modulare Entwicklung von Software ist ein wichtiger Ansatz zur Beherrschung von Komplexität. Komponenten unterstützen Modularität, da durch die expliziten Schnittstellen nur bestimmte Funktionalität angeboten wird und interne Details gemäß dem Prinzip des „information hiding“ [Par72] verborgen sind. Dadurch können Komponenten ausschließlich auf Grundlage ihrer Schnittstelle genutzt werden, so dass ein Austausch von Komponenten mit kompatiblen Schnittstellen möglich ist, ohne dass das gesamte System geändert werden muss.

Wiederverwendbarkeit: Die Wiederverwendbarkeit von Komponenten ist ein Ergebnis der Modularität. Eine Komponente ist sowohl eine Abstraktion als auch eine Implementierung einer konkreten Funktionalität [BW98]. Die Abstraktion wird durch die Schnittstelle erreicht, die die Implementierung der Komponente kapselt, so dass diese nur auf Grundlage der Schnittstelle wiederverwendet werden kann.

Erweiterbarkeit: Die Erweiterbarkeit eines Systems wird durch einen komponentenbasierten Ansatz auf mehrere Arten unterstützt [Vö11]. Einerseits kann die Funktionalität einzelner Komponenten bei stabil gehaltener Schnittstelle verfeinert werden. Dadurch wird die Funktionalität des gesamten Systems erweitert, ohne dass andere Komponenten verändert

werden müssen. Eine zweite Möglichkeit zur Erweiterung ist die Integration neuer Komponenten in ein System. Dies erfordert jedoch üblicherweise die Anpassung des restlichen Systems, damit die Funktionalität der neuen Komponente genutzt werden kann.

2.2.2 Frameworks

Objekt-orientierte Frameworks sind eine wichtige und weit verbreitete Technik zur Wiederverwendung von erweiterbaren Architekturen [Joh97]. Viele aktuelle Anwendungen werden auf Basis diverser Frameworks entwickelt, die Aspekte wie die Erstellung graphischer Benutzeroberflächen, die Persistenz von Daten, die Generierung von Code aus domänenspezifischen Sprachen oder die Entwicklung von Tests abdecken. Frameworks beeinflussen dementsprechend die Eigenschaften und die Architektur von Anwendungen auf verschiedenen Ebenen.

Wie im Fall von Komponenten existieren verschiedene Definitionen für den Begriff Framework, die jeweils unterschiedliche Aspekte in den Vordergrund stellen und dadurch die verschiedenen Sichtweisen auf Frameworks beschreiben [Joh97]. Im Kontext dieser Arbeit werden Frameworks auf Grundlage von [JF88, GHJV95, RJ96, FS97, Joh97] folgendermaßen definiert. Ein Framework ist ein wiederverwendbares abstraktes Design, das eine bewährte bzw. allgemeine Lösung für eine Menge von Problemen in einer bestimmten Domäne beschreibt. Ein Framework ist demnach selbst keine ausführbare Anwendung, sondern gibt die Architektur und die Eigenschaften von Anwendungen in der Domäne vor. Das Design eines Frameworks wird üblicherweise durch eine Menge abstrakter Klassen und der Interaktion ihrer Instanzen ausgedrückt. Diese Klassen müssen dann durch einen Anwendungsentwickler an definierten Stellen erweitert oder verfeinert werden, um das Framework-Verhalten zu spezialisieren und dadurch eine konkrete Anwendung zu erzeugen.

Durch den Aufbau und den Einsatzzweck von Frameworks ergeben sich einige grundlegende Eigenschaften und Vorteile. In der folgenden Auflistung werden die wichtigsten zusammengefasst, von denen einige auch im Kontext von Komponenten behandelt worden sind.

Modularität: Frameworks definieren stabile Schnittstellen, hinter denen die Details der Implementierung gemäß dem Geheimnisprinzip [Par72] verborgen sind. Dadurch haben Design- und Implementierungsänderungen nur lokale Auswirkungen und die Modularität einer Anwendung auf Grundlage eines Frameworks wird verbessert [FS97].

Wiederverwendbarkeit: Frameworks kapseln Wissen über eine Domäne in einer „halbfertigen“ Architektur [Pre95, FS97] und beschreiben dadurch das Verhalten und das Design einer Familie von Anwendungen. Über die Schnittstellen eines Frameworks können so verschiedene konkrete Anwendungen erzeugt werden, die sowohl das Domänenwissen als auch die Architektur wiederverwenden.

Erweiterbarkeit: Die Schnittstellen eines Frameworks bieten oft explizite Template- und Hook-Methoden [Pre95] an, über die Entwickler das spezifische Verhalten einer Anwendung bestimmen können [FS97]. Die Bereiche eines Frameworks, die über solche Mechanismen flexibel gehalten sind, werden auch als Hot Spots bezeichnet [Pre95].

Inversion of Control: Ein charakteristisches Merkmal von Frameworks ist, dass die anwendungsspezifischen Methoden vom Framework selbst aufgerufen werden. Dadurch gibt ein Framework nicht nur die Architektur vor, sondern steuert auch den Ablauf der Anwendung [JF88, FS97].

Die beschriebenen Eigenschaften und Vorteile von Frameworks gelten unabhängig davon, welche konkreten Techniken für die Erweiterungsmechanismen eingesetzt werden. Diese Techniken können jedoch für eine Klassifikation von Frameworks herangezogen werden. Bei sogenannten white-box Frameworks wird die anwendungsspezifische Funktionalität hauptsächlich durch Vererbung und dynamisches Binden ergänzt [FS97]. Spezifische Framework-Klassen werden dabei erweitert und deren Hook-Methoden unter Verwendung von Mustern wie Template Method [GHJV95] überschrieben. Diese Art von Frameworks werden als white-box Frameworks bezeichnet, da die Implementierung der Oberklassen einem Anwendungsentwickler bekannt sein muss. Im Gegensatz dazu basieren sogenannte black-box Frameworks nur auf der Definition von Schnittstellen und nicht auf der Verfügbarkeit des internen Aufbaus. Komponenten, die die Schnittstellen eines black-box Frameworks erfüllen, können in ein solches Framework beispielsweise mit Hilfe des Strategy-Musters [GHJV95] integriert werden [FS97].

Durch die unterschiedlichen Ansätze zur Erweiterung eines Frameworks entstehen einige Vor- und Nachteile. White-box Frameworks können flexiblere Erweiterungsmechanismen als black-box Frameworks anbieten, sind dadurch jedoch auch schwieriger zu verstehen und zu benutzen, da detaillierte Kenntnisse über den internen Aufbau des Frameworks vorhanden sein müssen [JF88]. Black-box Frameworks sind üblicherweise einfacher zu verstehen und zu benutzen, aber dafür schwieriger zu entwickeln [FS97]. Dadurch werden Frameworks in den meisten Fällen nicht direkt als black-box Frameworks entwickelt, sondern initial als white-box Frameworks konzipiert und inkrementell in black-box Frameworks überführt [RJ96]. Dementsprechend können Frameworks im Laufe ihres Lebenszyklus sowohl white-box-Anteile als auch black-box-Anteile enthalten, so dass eine strikte Unterscheidung in vielen Fällen nicht möglich ist.

Frameworks und Komponenten als kooperierende Technologien

Komponenten und Frameworks unterscheiden sich in einigen Aspekten. Komponenten erlauben keinen Einblick in ihren internen Aufbau und bieten eine abgeschlossene Menge an Operationen an, die ausschließlich über die definierten Schnittstellen aufgerufen werden können. Bei Frameworks sind gegebenenfalls Teile des internen Aufbaus bekannt und auch erweiterbar, so dass eine stärkere Modifikation der Funktionalität möglich ist. Frameworks bieten auch einen höheren Grad der Abstraktion und der Flexibilität, da sie nicht nur die Wiederverwendung von Code, sondern auch von domänenspezifischen Architekturen unterstützen [Joh97]. Dadurch haben Frameworks jedoch auch komplexere Schnittstellen, die den Aufwand bei deren Verwendung und die Kopplung zwischen einem Framework und dem anwendungsspezifischen Code erhöhen.

Insgesamt sind Komponenten und Frameworks daher unterschiedliche, gleichzeitig aber auch kooperierende Technologien. Komponenten machen bestimmte Annahmen über ihre Umgebung, so dass es schwierig sein kann, Komponenten mit unterschiedlichen Annahmen zu kombinieren [GAO95]. Durch ein black-box Framework kann der Wiederverwendungskontext von Komponenten definiert werden, so dass deren Komposition durch einheitliche Vorgaben vereinfacht

werden kann [Joh97, FS97, BW98]. Dadurch wird auch die Entwicklung neuer Komponenten erleichtert, da ein Framework die Spezifikation sowie Vorlagen für neue Komponenten bereitstellt [Joh97].

2.2.3 Plug-ins

Plug-in-Architekturen basieren auf den zuvor beschriebenen Konzepten von Komponenten und Frameworks und sind eine spezielle Art der Komponentenarchitektur [MV03]. Sie haben sich in den letzten Jahren zunehmend verbreitet, unter anderem auch durch den Erfolg von Anwendungen wie die Eclipse IDE [dRW04]. Die wichtigsten Elemente von Plug-in-Architekturen sind die Plug-ins, die sogenannte Host-Anwendung sowie der Plug-in-Vertrag. Die Eigenschaften dieser Elemente sowie deren Beziehungen zu Komponenten und Frameworks werden in diesem Abschnitt behandelt.

Plug-ins können zur Laufzeit einer Host-Anwendung geladen und ausgeführt werden. Durch eine wohldefinierte Schnittstelle – den Plug-in-Vertrag – gibt die Host-Anwendung den Kontext und die Anforderungen an die integrierbaren Plug-ins vor [Mar01]. Dadurch kann ein Plug-in die Host-Anwendung um bestimmte Funktionalität ergänzen, die von dieser selbst nicht bereitgestellt, jedoch über entsprechende Schnittstellen vorgesehen wird. Die Funktionalität wird dabei durch ein Plug-in so realisiert, dass die Host-Anwendung keine internen Details, sondern nur die Schnittstelle kennen muss [MV03], so dass das Prinzip des „information hiding“ [Par72] erfüllt wird. Insbesondere sind zur Kompilierzeit der Host-Anwendung keine konkreten Plug-ins bekannt, so dass keine Informationen über ein spezifisches Plug-in in der Host-Anwendung verfügbar sind [MMS03].

Der Plug-in-Vertrag ist ein zentrales Element von Plug-in-Architekturen [MV03]. Er enthält sowohl die Definition der Schnittstelle, die von Plug-ins erfüllt werden muss, als auch der Schnittstelle, die von der Host-Anwendung bereitgestellt wird. Erstere wird auch als „Plug-in Definition“ bezeichnet [Mar01] und gibt an, welche Operationen verpflichtend und welche optional durch ein Plug-in realisiert werden müssen. Letztere wird auch „Framework Interface“ genannt [Mar01] und spezifiziert die Dienste und Domänenklassen, die durch die Host-Anwendung zur Verfügung gestellt werden.

Bei Plug-in-Architekturen unterscheidet man zwischen klassischen und reinen Plug-in-Architekturen. In klassischen Plug-in-Architekturen erweitern die Plug-ins eine Host-Anwendung wie zuvor beschrieben um bestimmte Funktionalität über wohldefinierte Schnittstellen. In einer reinen Plug-in-Architektur wird die gesamte Funktionalität einer Anwendung durch Plug-ins realisiert. Die Host-Anwendung ist nur noch eine minimale Laufzeitumgebung ohne Endnutzerfunktionalität [Bir05, Rat11]. Abbildung 2.3 veranschaulicht den Unterschied beider Architekturvarianten.

In reinen Plug-in-Architekturen können Plug-ins nicht nur auf der Host-Anwendung aufbauen, sondern auch auf anderen Plug-ins [Bir05], die dann als Host-Plug-ins bezeichnet werden [Rat08]. Im Gegensatz dazu können in klassischen Plug-in-Architekturen Plug-ins ausschließlich die Host-Anwendung erweitern, so dass sie keine Informationen über andere Plug-ins besitzen und nicht aufeinander aufbauen können. Für reine Plug-in-Architekturen gilt diese Eigenschaft auf einer Integrationsebene [MV03].

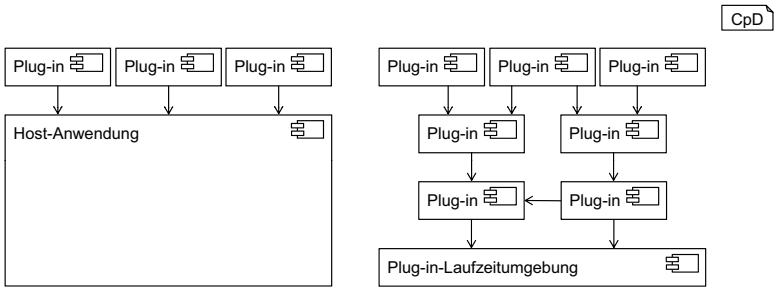


Abbildung 2.3: Schematische Darstellung einer klassischen und einer reinen Plug-in-Architektur (adaptiert aus [Bir05, Rat11])

Plug-in-Architekturen auf Grundlage von Komponenten und Frameworks

Plug-ins sind Softwarekomponenten gemäß der oben angegebenen Definition und werden in [Szy02] auch als feingranulare Komponenten bezeichnet. Sie bilden eine kompositionale Einheit, erfüllen einen Plug-in-Vertrag, der die Abhängigkeiten und die Schnittstellen beschreibt und können individuell ausgeliefert und unabhängig installiert werden [Rat08, Rat11]. Insbesondere reine Plug-in-Architekturen sind verwandt mit Komponentensystemen, da in diesem Fall Plug-ins das Zerlegungskonzept einer Anwendung realisieren und damit deren atomaren Bestandteile sind [Rat11].

Die Host-Anwendung definiert grundlegende Aspekte der Anwendungsdomäne, die über den Plug-in-Vertrag den Plug-ins verfügbar gemacht werden [Mar01, MV03]. Dementsprechend können Plug-ins nur für eine bestimmte Host-Anwendung geschrieben und in deren Kontext ausgeführt werden [MV03]. Dadurch ist die Host-Anwendung ein Framework für Plug-ins und wird daher auch „Framework-Providing Application“ genannt [Mar01]. Es handelt sich dabei um ein black-box Framework, da die Plug-ins keine Informationen über den internen Aufbau der Host-Anwendung erhalten [Mar01]. Neben den Framework-Bestandteilen enthält die Host-Anwendung auch die Logik, um den Lebenszyklus der Plug-ins zu kontrollieren [Rat08, Bir05].

Neben den Gemeinsamkeiten von Plug-in-Architekturen und Komponentensystemen lassen sich noch einige Unterschiede feststellen. Plug-ins besitzen durch die Framework-Bestandteile und die domänenspezifische Schnittstelle der Host-Anwendung eine engere Kopplung mit dieser. Komponenten sind im Vergleich dazu üblicherweise generischer, da sie eine kompaktere Schnittstelle besitzen und nicht so stark gekoppelt sind [MV03].

2.3 Zusammenfassung

In diesem Kapitel wurden einige Grundlagen zum Verständnis der vorliegenden Arbeit behandelt. Neben der Unterstützung von Software Engineering Projekten durch die Integration von Werkzeugen sowie deren servicebasierte Nutzung wurden auch technische Aspekte erweiterbarer Architekturen beschrieben. Aufbauend auf den hier eingeführten Begriffen und Konzepten

werden in den folgenden Kapiteln die Konzepte und die Architektur des SSELab-Frameworks diskutiert. Dabei wird bei Bedarf jeweils auf die hier behandelten Grundlagen Bezug genommen.

Kapitel 3

Nutzungsszenarien mit ihren Rollen und Anforderungen

In diesem Kapitel wird ein Überblick über die wichtigsten Anforderungen an das SSELab gegeben. Dazu wird zunächst in Abschnitt 3.1 dessen Verwendung in einigen Szenarien aus der Sicht verschiedener Nutzerrollen beschrieben. Danach werden in Abschnitt 3.2 alle Rollen definiert, die Personen bei der Interaktion mit dem SSELab annehmen können. Aufbauend auf diesen beiden Abschnitten werden in Abschnitt 3.3 die wichtigsten Anforderungen angegeben. Die Kategorisierung der Anforderungen erfolgt dabei sowohl auf Grundlage der definierten Rollen als auch der Eigenschaften des SSELabs als Framework und als verteilte webbasierte Anwendung.

3.1 Nutzungsszenarien

Szenarien werden im Requirements Engineering eingesetzt, um Anforderungen an ein System aus Sicht der Nutzer zu beschreiben. Ein Szenario ist immer exemplarisch und gibt die Verwendung des Systems durch einen oder mehrere Nutzer als eine Sequenz von Interaktionsschritten an [Gli00]. In den nachfolgenden Abschnitten wird zuerst der Kontext des jeweiligen Szenarios angegeben. Im Anschluss folgt die Beschreibung des Szenarios in natürlicher Sprache. Zur Strukturierung wird dabei die Interaktion der Nutzer mit dem SSELab explizit hervorgehoben. Aus den Szenarien werden in den nachfolgenden Abschnitten alle Rollen und die Anforderungen an das SSELab abgeleitet.

3.1.1 Softwareentwicklung in einem agilen verteilten Team

Im ersten Beispielszenario wird ein global verteiltes Software Engineering Projekt beschrieben. In diesem Projekt wird Scrum [SB02] als Vorgehensmodell eingesetzt. Insbesondere kommt eine Variante von Scrum zum Einsatz, bei der die Mitglieder der Entwicklerteams über mehrere Standorte verteilt sind [SVBP07]. Im Rahmen des Projekts wird eine Java-Enterprise-Anwendung unter Verwendung von Modellierungstechniken auf Grundlage der UML/P [Rum11, Rum12, Sch12] realisiert. Das Projekt verwendet das SSELab als Portal und nutzt die folgenden Services:

- Subversion [PCF08, Col02] als Versionsverwaltungssystem.
- Trac [Tra12] als Projektmanagementwerkzeug und als Tracker für die Product und Sprint Backlogs.

- Jenkins [Sma11] als Continuous Integration Server.
- Nexus [MOB+11, Nex12] als Verwaltungssystem für die Entwicklungsartefakte.
- Einen Storage-Service auf Basis von WebDAV [WG99].
- Mailing-Listen mit einem webbasierten Archiv.
- MontiCore [KRV07b, KRV08, KRV10, Kra10] als Framework für die Generierung und Verarbeitung domänenspezifischer Sprachen sowie die UML/P [Rum11, Rum12], die mit Hilfe von MontiCore in eine Werkzeuginfrastruktur umgesetzt worden ist [Sch12].

In dem Szenario läuft das Projekt bereits seit einiger Zeit und befindet sich momentan in einem Sprint. Im Folgenden werden einige Aspekte des Tagesablaufs eines Entwicklers beschrieben.

Der Tag des Entwicklers beginnt mit der Vorbereitung des Daily Scrum. Aufgrund der globalen Verteilung des Projekts und der daraus resultierenden Sprachbarrieren wurde von den Entwicklerteams entschieden, dass die drei Fragen (Was hat jeder einzelne seit dem letzten Daily Scrum erreicht? Was soll bis zum nächsten Daily Scrum erreicht werden? Welche Hindernisse gibt es?) vor dem Daily Scrum per Mail an alle Teammitglieder geschickt werden.

Entwickler (E): Der Entwickler schreibt eine E-Mail mit seinen Antworten auf die drei Fragen in seinem Mail-Programm und sendet sie an die Mailing-Liste des entsprechenden SSELab-Projekts.

SSELab (S): Nach dem Erhalt der E-Mail wird diese an alle Mitglieder des Projekts gesendet und zusätzlich in dem Web-Archiv der Mailing-Liste gespeichert.

Nach dem Ende des Daily Scrums, das per Telefonkonferenz durchgeführt wurde, beginnt der Entwickler die Arbeit an seiner heutigen Aufgabe.

E: Er öffnet Eclipse und loggt sich über Mylyn im Trac des SSELab-Projekts ein, in dem die Tickets mit den Entwicklungsaufgaben gespeichert sind. Der Entwickler weist sich selbst das Ticket mit seiner heutigen Aufgabe zu und startet zunächst mit der Anpassung der Templates, aus denen mit Hilfe von MontiCore und der UML/P aus Klassendiagrammen die Persistenzschicht der Anwendung generiert wird. Nachdem er das Template angepasst hat, verwendet er die Generierung über den in das SSELab integrierten UML/P- und MontiCore-Service. Anschließend nutzt er die neuen Funktionen des generierten Codes, um den manuell geschriebenen Code zu erweitern.

E: Vor der Mittagspause hat er seine Aufgabe fertig implementiert, so dass alle geschriebenen Tests auf seinem Entwicklersystem fehlerfrei durchlaufen. Daraufhin commitet er seine lokalen Änderungen aus Eclipse heraus in das zentrale Subversion-Repository des SSELabs.

S: Nach dem Commit versendet das SSELab automatisch Commit-Mails an die Projektmitglieder. Außerdem startet Jenkins kurz danach den automatischen Build, bei dem das gesamte System kompiliert, alle (Unit-)Testfälle ausgeführt werden sowie Qualitätsmetriken berechnet werden. Nach dem erfolgreich durchgelaufenen Build werden die Artefakte im Nexus des SSELab-Projekts aktualisiert, so dass die anderen Entwickler leicht Zugriff auf die aktuellsten Java-Archive haben. Darüber hinaus wird ebenfalls die Dokumentation aus den JavaDoc-Kommentaren des Codes generiert und auf dem Speicherplatz des Storage-Services des SSELab-Projekts abgelegt, so dass die API-Dokumentation des letzten erfolgreichen Builds direkt verfügbar ist. Dort werden zusätzlich auch die Ergebnisse der Testausführung und die berechneten Metriken als HTML-Dateien abgelegt.

E: Nach dem Mittagessen überprüft der Entwickler die Ergebnisse des Builds in Jenkins. Anschließend macht er sich an die Arbeit, um ein Problem zu lösen, das während der Mittagspause von den Entwicklern besprochen wurde. Während der Entwicklung muss er eine Bibliothek verwenden, mit der er bisher nur wenig Erfahrung hat. Da er weiß, dass sich auch keiner der Kollegen vor Ort mit dieser Bibliothek auskennt, öffnet er die Projektseite im SSELab und sucht dort innerhalb des Projekts nach Kollegen mit Erfahrungen im Umgang mit der Bibliothek.

S: Im SSELab haben die Entwickler in ihren jeweiligen Profilen einige ihrer persönlichen Daten gespeichert sowie insbesondere ihre technischen Kenntnisse und Fähigkeiten. Bei der Suche werden die Profile der Projektmitglieder miteinbezogen und eine Liste von passenden Ergebnissen angezeigt.

E: Ein Kollege an einem anderen Standort hat gemäß seinem Profil Erfahrungen mit der Bibliothek. Der Entwickler öffnet die Profilseite des Kollegen und startet mit Hilfe der dort angegebenen Instant Messaging Adresse einen Chat. Nach der Diskussion mit dem Kollegen arbeitet der Entwickler mit dem neu gewonnenen Wissen über die Bibliothek weiter an der Lösung des Problems.

3.1.2 Erstellung eines wissenschaftlichen Papiers

Im zweiten Beispielszenario wird ein relativ kurzlebiges Projekt betrachtet, in dem drei wissenschaftliche Mitarbeiter ein Papier schreiben und auf einer Konferenz einreichen wollen. Das Szenario ist aus Sicht der treibenden Mitarbeiterin geschrieben.

Bei einem initialen Treffen der drei Autoren im Besprechungsraum wird zunächst die Konferenz ausgewählt, auf der das Papier eingereicht werden soll. Die Mitarbeiterin schließt ihren Laptop an den Beamer an und öffnet die Webseite der Konferenz mit dem Call for Paper und den Terminen zur Einreichung von Beiträgen. Dann werden die ersten Inhalte des Papiers besprochen und ein Arbeitstitel festgelegt. Die Mitarbeiterin öffnet währenddessen die Webseite des SSELabs.

Mitarbeiterin (M): Sie navigiert von der Startseite auf die Seite zur Erzeugung eines neuen Projekts. Auf der Seite gibt sie den Namen des Projekts und eine passende Beschreibung

ein. Als Ablaufdatum des Projekts wählt sie einen Termin einige Wochen nach der Konferenz. Als Services für das Projekt wählt sie Subversion und MediaWiki [MW12] und erzeugt schließlich das Projekt, nachdem sie noch ihren SSELab-Benutzernamen und ihr Passwort angegeben hat.

SSELab (S): Das SSELab schickt nach der Erzeugung des Projekts und der Konfiguration der ausgewählten Services eine E-Mail an die Mitarbeiterin mit Links zu der Webseite des Projekts sowie zu den Services und der Dokumentation des SSELabs.

M: Nach dem Erhalt der E-Mail öffnet die Mitarbeiterin die Projektseite und lädt die beiden Mitautoren unter Angabe ihrer E-Mail-Adressen in das Projekt ein.

S: Das SSELab versendet jeweils eine E-Mail mit einer Einladung an die angegebenen E-Mail-Adressen, dem neu erstellten Projekt beizutreten. In der E-Mail ist eine Erklärung zu der Einladung sowie der Absender und eine Beschreibung des Projekts enthalten. Außerdem ist ein Link zu der Seite enthalten, auf der die Einladung angenommen werden kann.

M: Anschließend öffnet die Mitarbeiterin das Wiki, um Stichpunkte der nachfolgenden Diskussion festzuhalten.

Nach dem Ende des Treffens diskutieren zwei der Autoren einige weitere technische Details und die Verteilung der Arbeitspakete. Da das Papier in LaTeX geschrieben werden soll, erstellt die treibende Autorin sowohl aus der Vorlage des Lehrstuhls als auch aus der der Konferenz alle benötigten Dateien, um das Papier erstellen zu können.

M: Anschließend verwendet sie ihren installierten Subversion-Client und checkt alle Dateien in das Subversion-Repository des neu angelegten SSELab-Projekts ein.

S: Das SSELab sendet daraufhin Commit-Mails mit Informationen über die Änderungen am Repository an alle Mitglieder des Projekts.

Anschließend diskutieren die beiden Mitarbeiter noch über den Inhalt des Papiers und die technische Infrastruktur. Dabei entscheiden sie, dass die Abbildungen des Papiers in Powerpoint gezeichnet werden sollen, damit diese relativ einfach in den Vortragsfolien wiederverwendet werden können.

M: Für eine möglichst einfache Integration der Abbildungen in das LaTeX-Dokument aktiviert die Mitarbeiterin zusätzlich den SSELab-Service ppt2eps für das Projekt, mit dessen Hilfe Abbildungen aus Powerpoint-Dateien in eps- und pdf-Dateien umgewandelt werden können.

S: Das SSELab sendet eine Mail an die Mitglieder des Projekts, dass der neue Service im Projekt genutzt werden kann.

Dann beginnt die Schreibphase in der alle Autoren das Papier bearbeiten und ihre Änderungen mehrmals täglich in das Subversion-Repository commiten. Nachdem das Papier inhaltlich vollständig ist, wird es in einer internen Reviewphase mehrmals überarbeitet, um eine hohe Qualität sicherzustellen.

3.1.3 Integration eines Transformationswerkzeugs

In diesem Szenario wird die Integration des Transformationswerkzeugs xml2xsd [Her05] in das SSELab beschrieben. Mit Hilfe dieser Java-Anwendung kann aus einer Menge von XML-Dateien [BPM+08] eine XML Schema Definition [TBMM04, BM04] erzeugt werden, die das Format aller Eingabedateien beschreibt.

Service-Entwickler: Der Entwickler verbindet sich mit seiner virtuellen Maschine, auf der die Entwicklungs- und Laufzeitumgebung des SSELabs vorinstalliert sind. Unter Verwendung der APIs des SSELab-Frameworks für die Integration von Transformationswerkzeugen entwickelt er die Integrationskomponente. Zusätzlich zu den Unit-Tests führt er einen manuellen Integrationstest aus, indem er das Werkzeug und die Integrationskomponente in der SSELab-Instanz auf der Entwicklermaschine installiert. Nach dem erfolgreichen Test, stellt er einem Administrator der SSELab-Instanz unter `http://sselab.de` das Werkzeug sowie die Integrationskomponente zur Verfügung.

Administrator (A): Der Administrator loggt sich mit seinem Account unter der URL `http://sselab.de` in der SSELab-Instanz ein und lädt über die Web-Oberfläche das Werkzeug und die Integrationskomponente hoch.

SSELab: Das SSELab installiert das Werkzeug und die Integrationskomponente in der Laufzeitumgebung und prüft anschließend die wichtigsten Eigenschaften der Integrationskomponente und gibt das Resultat aus.

A: Nach erfolgreicher Installation aktiviert der Administrator den neuen Service, so dass er von allen Benutzern der SSELab-Instanz genutzt werden kann.

3.1.4 Erweiterung der Kapazität einer SSELab-Instanz

Dieses Beispielszenario beschreibt die Integration einer neuen virtuellen Maschine in die Serverinfrastruktur der SSELab-Instanz unter `http://sselab.de`. Die Integration erfolgt aus Kapazitätsgründen: es sollen neue Projekte unterstützt werden können, ohne dass die Leistungsfähigkeit der Services bereits vorhandener Projekte beeinträchtigt wird.

System-Administrator: Der System-Administrator erstellt die neue virtuelle Maschine für das SSELab aus einem gespeicherten Template und nimmt sie anschließend in Betrieb, indem er ihr eine IP-Adresse zuweist und sie startet. Anschließend loggt er sich auf der Maschine mit einem System-Account ein und führt die Skripte zur Konfiguration der Maschine als neue Backend-Instanz der SSELab-Instanz aus. Nach der erfolgreichen Konfiguration der Maschine übermittelt er die URL des auf der Maschine laufenden Servers an einen Administrator der SSELab-Instanz.

Administrator (A): Der Administrator meldet sich mit seinem Account unter der URL `http://sselab.de` in der SSELab-Instanz ein, trägt die URL der neuen Maschine in der Web-Oberfläche ein, so dass diese als neue Backend-Instanz registriert wird.

SSELab: Das SSELab prüft, ob unter der angegebenen URL eine entsprechende Backend-Instanz läuft. Nach erfolgreicher Prüfung werden alle Services auf der neuen Instanz installiert, so dass alle registrierten Backend-Instanzen in einem konsistenten Zustand sind.

A: Nach dem erfolgreichen Test der neuen Backend-Instanz gibt der Administrator diese zur allgemeinen Nutzung frei.

3.1.5 Interaktion externer Werkzeuge mit dem SSELab

In diesem Szenario wird die Interaktion von Werkzeugen, die nicht in das SSELab-Framework integriert sind, mit den Services einer SSELab-Instanz beschrieben. Als Beispiel wird hier ein externer Continuous Integration (CI) Server betrachtet, der durch einen Entwickler eines SSELab-Projekts betrieben wird.

Entwickler (E): Der Zugriff des externen Continuous Integration Servers auf die Services einer SSELab-Instanz wird nur nach einer erfolgreichen Authentifizierung und Autorisierung zugelassen. Daher erstellt der Entwickler im Kontext des relevanten Projekts zunächst einen sogenannten Bot-Account, der für die Interaktion zwischen externen Werkzeugen und den Services des SSELabs ausgelegt ist.

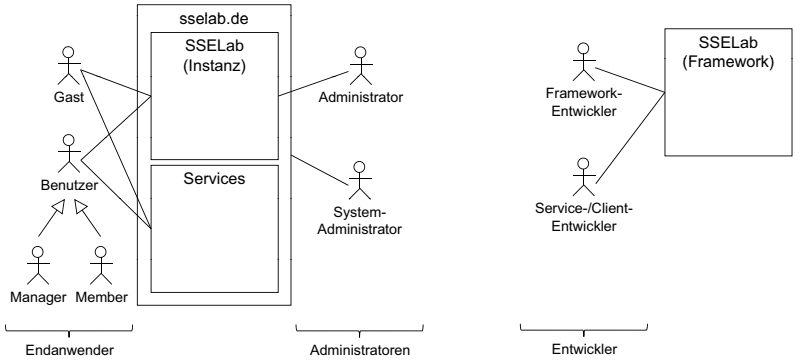
SSELab: Das SSELab erzeugt einen neuen Bot-Account, ordnet ihn dem Projekt zu und aktualisiert die Authentifizierungs- und Autorisierungsinformationen aller Services des Projekts, so dass der Zugriff des Bot-Accounts darauf ermöglicht wird.

E: Der Entwickler konfiguriert im Anschluss daran den externen CI Server. Er verwendet dabei sowohl die URL für das Subversion-Repository des SSELab-Projekts als auch den Kontonamen und das Passwort des neu angelegten Bot-Accounts, um den externen CI Server für die Überwachung des Repositories zu konfigurieren.

3.2 Rollen

In den Szenarien des vorigen Abschnitts wurden bereits einige Rollen eingeführt. In diesem Abschnitt werden alle Rollen definiert, die von Personen bei der Interaktion mit dem SSELab angenommen werden können. Grundlegend für die Differenzierung der Rollen sind die verschiedenen Sichtweisen auf das SSELab: Einerseits ist es ein Framework für webbasierte Projektportale und CDEs und andererseits ein verteiltes webbasiertes System, dass bereits ohne konkrete Erweiterungen in Form von Services oder Clients lauffähig ist. Diese beiden Sichtweisen und die zugehörigen Rollen werden durch Abbildung 3.1 veranschaulicht.

In Abbildung 3.1a ist die Instanz des SSELab-Frameworks dargestellt, die unter `http://sselab.de` läuft. Die Instanz selbst bietet den Benutzer unter anderem Funktionen zur Verwaltung von Konten und Projekten an. Zusätzlich sind über diese Instanz mehrere Werkzeuge als Services installiert. Die in der Abbildung gezeigten Rollen, die mit der SSELab-Instanz und den Services interagieren, werden im Folgenden definiert.



(a) SSELab-Instanz mit Rollen (b) SSELab-Framework mit Rollen

Abbildung 3.1: Rollen und deren Interaktion mit dem SSELab

Gast: Ein Gast ist nicht beim System authentifiziert, so dass er nur die öffentlichen Seiten der SSELab-Instanz nutzen kann. Dazu gehören einerseits die Seiten der Instanz selbst und andererseits die öffentlich zugänglichen Seiten von Services.

Benutzer: Ein Benutzer ist beim System authentifiziert, so dass er Zugang zu seinen Projekten und den zugehörigen Services besitzt sowie allen Verwaltungsmechanismen der SSELab-Instanz. Im Kontext von Projekten werden zusätzlich die beiden Rollen *Manager* und *Member* unterschieden. Die Manager eines Projekts können administrative Aufgaben wie die Aktivierung oder Deaktivierung von Services ausführen. Member können dagegen nur die Services des Projekts nutzen.

Administrator: Ein Administrator ist beim System authentifiziert und hat administrative Rechte in der gesamten Web-Oberfläche. Er kann alle Benutzer, Projekte sowie die angebotenen Services verwalten. Ein Administrator kann jedoch nicht Mitglied in Projekten sein, so dass er auch keinen Zugriff auf die Service-Instanzen einzelner Projekte besitzt.

System-Administrator: Ein System-Administrator hat Zugang zu den Servermaschinen der SSELab-Instanz auf Betriebssystemebene und ist verantwortlich für die Verwaltung und Konfiguration der gesamten Serverinfrastruktur.

Die Sicht auf das SSELab als Framework für webbasierte Projektportale ist in Abbildung 3.1b dargestellt. Die folgenden Rollen werden bei der Interaktion mit dem Framework unterschieden.

Framework-Entwickler: Ein Framework-Entwickler kennt die innere Architektur des SSELab-Frameworks und kann Funktionen innerhalb des Kerns des Systems realisieren.

Service-/Client-Entwickler: Ein Service- oder Client-Entwickler kennt die Schnittstellen des Frameworks und nutzt diese, um Werkzeuge als Services zu integrieren oder Clients für das Framework bzw. für Services zu entwickeln.

3.3 Anforderungen

In diesem Abschnitt werden die wichtigsten Anforderungen an das SSELab auf Grundlage der zuvor beschriebenen Sichtweisen und der zugehörigen Rollen angegeben. Ein Großteil der Anforderungen wurde von den Mitgliedern des Lehrstuhls für Software Engineering auf Grundlage der in dieser Arbeit behandelten Forschungsfrage selbst definiert. Sie basieren dementsprechend sowohl auf den Erfahrungen des Lehrstuhls bei der Durchführung diverser Forschungsprojekte mit industriellen Kooperationspartnern und vielfältiger Lehrveranstaltungen sowie der Entwicklung komplexer Systeme. Insgesamt sind die Mitglieder des Lehrstuhls daher die primären Kunden, die über die Erfüllung der Anforderungen entscheiden können, da durch sie auch alle zuvor definierten Rollen eingenommen werden. Darüber hinaus wurde eine Literaturrecherche im Bereich von Projektportalen und CDEs durchgeführt, deren Ergebnisse ebenfalls in die Definition der Anforderungen eingeflossen sind. Mit zunehmendem Einsatz der im Rahmen dieser Arbeit gebildeten SSELab-Instanzen wurden außerdem weitere Anforderungen durch externe Projektpartner und andere Lehrstühle definiert, die auch berücksichtigt worden sind.

Im folgenden Abschnitt 3.3.1 werden zunächst die Anforderungen an die Funktionen der SSELab-Instanz aus Sicht der Endanwender – also der Rollen Gast und Benutzer – aufgeführt. Anschließend werden in Abschnitt 3.3.2 die Anforderungen dieser beiden Rollen an die Services der Instanz beschrieben, die speziell für verteilte Software-Entwicklungsprojekte mit agilen Prozessen [BBvB+12] ausgelegt ist. Im Anschluss daran erfolgt in Abschnitt 3.3.3 die Darstellung der Anforderungen an die SSELab-Instanz aus Sicht der Administratoren. Abschließend werden in Abschnitt 3.3.4 die Anforderungen an das SSELab als Framework aufgeführt. Diese Anforderungen sind aus Sicht der Entwickler beschrieben, die das SSELab erweitern sollen, so dass sowohl Anforderungen an eine Methodik zur Nutzung und zur Erweiterung des Frameworks als auch Anforderungen an die Architektur angegeben werden.

Im Folgenden werden jeweils funktionale und nichtfunktionale Anforderungen definiert. Funktionale Anforderungen beschreiben, was das System aus Sicht der Nutzer leisten soll, wohingegen nichtfunktionale Anforderungen die Qualitätsattribute des Systems bestimmen [Bal09]. Jede Anforderung ist mit einem eindeutigen Schlüssel versehen, der in den folgenden Kapiteln bei der Diskussion von Konzepten und Architekturentscheidungen zum Referenzieren verwendet wird.

3.3.1 Anforderungen an die SSELab-Instanz aus Sicht der Endanwender

In diesem Abschnitt werden die Anforderungen an die Funktionen der SSELab-Instanz aus der Sicht der Endanwender aufgeführt. Gemäß der Definition aus Abschnitt 3.2 besitzt ein Endanwender die Rolle *Benutzer*, sobald er sich beim System authentifiziert hat. Andernfalls besitzt er die Rolle *Gast*.

Funktionale Anforderungen

FA1 *Nutzung über zentrale Web-Oberfläche*

Die Funktionen der SSELab-Instanz sollen über eine zentrale Web-Oberfläche per Browser genutzt werden können. Es soll nicht notwendig sein, dazu zusätzliche Software zu installieren.

FA2 *Zugriff als Gast*

Ein Gast soll einen Überblick über die SSELab-Instanz und die angebotenen Services bekommen können. Darüber hinaus soll er sich einen Account erzeugen und neue Projekte beantragen können. Außerdem soll es für einen Gast möglich sein, die öffentlich zugänglichen Services aller Projekte zu nutzen. Weiterhin soll ein Login entweder als Benutzer oder als Administrator und bei Bedarf die Anforderung eines neuen Passworts möglich sein.

FA3 *Verwaltung des eigenen Accounts*

Benutzer sollen ihren eigenen Account verwalten können, d. h. ihre Profilinformationen anpassen sowie ihr Passwort und ihre E-Mail-Adresse ändern können. Zu den Profilinformationen gehören unter anderem ein Profilbild, das Geschlecht, der Geburtstag, eine Kurzbeschreibung („about me“), Adressen, weitere E-Mail-Adressen und Fähigkeiten.

FA4 *Verwaltung der Sichtbarkeit von Profilinformationen*

Da in den Profilinformationen viele persönliche Daten gespeichert werden können, sollen die Benutzer die Sichtbarkeit ihrer Profilinformationen kontrollieren können. Es soll dementsprechend möglich sein, die Sichtbarkeit der Informationen innerhalb der SSELab-Instanz einzuschränken.

FA5 *Erzeugung neuer Projekte*

Ein Benutzer soll die Möglichkeit haben, neue Projekte zu erzeugen. Dazu soll er die benötigten Informationen wie Name und Beschreibung angeben und die gewünschten Services auswählen können. Der Benutzer soll automatisch Mitglied des neuen angelegten Projekts in der Rolle *Manager* sein.

FA6 *Nutzung von Projekten*

Innerhalb eines Projekts soll ein Benutzer entweder die Rolle *Manager* oder *Member* einnehmen. Sowohl als Manager als auch als Member soll ein Benutzer die Services eines Projekts konfigurieren, parametrisieren und nutzen können.

FA7 *Verwaltung vieler Projekte*

Die SSELab-Instanz soll darauf ausgelegt sein, dass die Benutzer viele Projekte organisieren können. Die Benutzer sollen schnell einen Überblick über ihre Projekte bekommen und diese effizient verwalten können.

FA8 *Administration von Projekten*

Projektweite Einstellungen sollen von den Managern eines Projekts vorgenommen werden können: Ein Manager soll andere Benutzer aus dem Projekt entfernen, die Rolle der Benutzer ändern oder Personen (mit oder ohne Benutzer-Account) in das Projekt einladen können. Darüber hinaus soll ein Manager Services für das Projekt aktivieren oder deaktivieren können.

FA9 *Nutzung von Services*

Ein Benutzer soll sich über alle verfügbaren Services in der SSELab-Instanz informieren können. Außerdem sollen alle Services, die für einen Benutzer bzw. dessen Projekte aktiviert sind, genutzt bzw. ausgeführt werden können.

FA10 *Kontext der Services*

In der SSELab-Instanz sollen geeignete Kategorien von Services verwendet und die Services in verschiedenen Kontexten angeboten werden. Beispielsweise sollen Services mit persistentem Zustand ausschließlich im Kontext von Projekten genutzt werden können. Services ohne persistenten Zustand sollen dagegen auch unabhängig von Projekten verwendbar sein.

FA11 *Konfiguration und Parametrisierung der Werkzeuge*

Bei der Integration von Werkzeugen als Services sollen die Konfigurations- bzw. Parametrisierungsmöglichkeiten der Werkzeuge erhalten bleiben, so dass deren voller Funktionsumfang auch nach der Integration für die Benutzer verfügbar ist.

FA12 *Bereitstellung von Clients (Framework-Clients)*

Die Nutzung der SSELab-Instanz und deren Funktionalität sowie der Services sollen nicht ausschließlich über einen Web-Browser, sondern auch über verschiedene Clients möglich sein. Diese Clients sollen sowohl in gängige IDEs integriert, als auch über die Kommandozeile bzw. aus Skripten heraus aufgerufen werden können. Ein Benutzer soll die im System verfügbaren Clients über die Web-Oberfläche herunterladen und ausführen können.

Zur Abgrenzung von den in Anforderung **FA24** beschriebenen Service-Clients werden diese Clients als Framework-Clients bezeichnet, da sie direkt mit einer SSELab-Instanz kommunizieren und ein Bestandteil des SSELab-Frameworks sind.

FA13 *Automatisierter Zugriff*

Im Kontext eines Projekts sollen sogenannte Bot-Accounts erzeugt und zu dem Projekt hinzugefügt werden können. Diese Accounts sollen ausschließlich für den automatisierten Zugriff auf die Services des Projekts genutzt werden können. Es soll nicht möglich sein, mit Hilfe eines solchen Accounts auf die Web-Oberfläche der SSELab-Instanz zuzugreifen.

Zusätzlich zur Nutzung von Bot-Accounts im Kontext eines Projekts sollen diese auch für Framework-Clients genutzt werden können. Ein Bot-Account soll bei Bedarf für einen Benutzer, der einen Client verwenden möchte, erzeugt werden, so dass sich bei der Verwendung des Clients der Benutzer nicht mehr selbst authentifizieren muss, sondern den Bot-Account dafür verwenden kann.

FA14 *Nutzung von Projekttemplates*

Projekttemplates fassen eine bestimmte Menge von Services sowie deren Konfigurationen und dadurch bestimmte Arten von Projekten zusammen. Sie sollen bei der

Erstellung eines Projekts von einem Benutzer verwendet werden können, so dass die einzelnen Services nicht manuell ausgewählt werden müssen.

FA15 *Zugriff auf Benutzerprofile anderer Benutzer*

Ein Benutzer soll entweder im Kontext eines Projekts oder projektübergreifend die Profile von anderen Benutzern auswählen und die sichtbaren Profilinformationen angezeigt bekommen können.

FA16 *Kontakt mit den Administratoren*

Ein Benutzer soll Nachrichten bzw. Support-Anfragen an die Administratoren über die Web-Oberfläche schicken und Antworten darüber empfangen können. Die Nachrichten sollen nicht nur im System angezeigt, sondern auch als E-Mails verschickt werden.

FA17 *Dokumentation*

Benutzer und Gäste sollen Zugriff auf Dokumentation des Systems, von Services und von Clients erhalten. Die Dokumentation soll jeweils im Kontext der aktuellen Seite angeboten werden.

FA18 *Suche*

Ein Benutzer soll nach anderen Benutzern, den eigenen Projekten sowie den verfügbaren Services suchen können.

Nichtfunktionale Anforderungen

NFA1 *Sicherer Zugriff von Benutzern*

Die Webseiten für Benutzer sollen nur nach erfolgreicher Authentifizierung mit einem gültigen Benutzerkonto erreichbar sein. Zur Authentifizierung werden ein Benutzername und ein Passwort verwendet.

NFA2 *Sichere Kommunikation*

Die Kommunikation zwischen allen Clients und der zentralen Web-Oberfläche sowie den integrierten Services soll verschlüsselt und unter Verwendung von Zertifikaten erfolgen.

NFA3 *Gängige Authentifizierungs- und Autorisierungsmechanismen*

Viele Werkzeuge, insbesondere Websysteme, verwenden eigene Authentifizierungs- und Berechtigungsmechanismen. Das SSELab soll daher einfache, flexible und auf Standards basierende Authentifizierungs- und Autorisierungsmechanismen verwenden.

NFA4 *Sichtbarkeit von Projekten und Benutzern*

Ein Benutzer soll nur Zugriff auf Projekte und damit deren Services haben, in denen er selbst Mitglied ist. Informationen über andere Projekte – wie beispielsweise deren Namen oder Mitglieder – sollen im System nicht sichtbar sein. Ein Ausnahme sind dabei explizit für ein Projekt freigeschaltete öffentliche Services, die Informationen

über ein Projekt preisgeben können. Darüber hinaus soll ein Benutzer nur im Kontext des Projekts andere Benutzerprofile sehen können. Die Anzeige und die Suche sollen dementsprechend nur Projekte und deren Mitglieder berücksichtigen, in denen der aktuelle Benutzer auch Mitglied ist.

NFA5 *Benutzerfreundlichkeit*

Das System sollte einfach zu benutzen sein. Dazu gehört, dass die Benutzeroberfläche komfortabel und intuitiv ist, so dass der Aufwand für die Einarbeitung in die Funktionsweise des Systems möglichst gering ist.

NFA6 *Performance bei der Nutzung*

Die Leistungsfähigkeit des System soll gut sein, so dass keine unnötigen Wartezeiten bei der Nutzung der SSELab-Instanz entstehen.

NFA7 *Verfügbarkeit der SSELab-Instanz*

Die Verfügbarkeit der Instanz und der installierten Services soll möglichst hoch sein.

3.3.2 Anforderungen an die Services aus Sicht der Endanwender

In diesem Abschnitt werden die Anforderungen an die Services der SSELab-Instanz für verteilte und agile Entwicklungsprojekte definiert. Die Anforderungen sind wie auch im vorigen Abschnitt aus Sicht der Endanwender geschrieben.

Funktionale Anforderungen

FA19 *Verschiedene Projektarten*

Die Instanz soll primär auf Software Engineering Projekte zugeschnitten sein. Darüber hinaus sollen aber noch weitere Projektarten durch eine geeignete Kombination von Services unterstützt werden. Ein Beispiel dafür sind Projekte für die Ausarbeitung von wissenschaftlichen Beiträgen, wie in Abschnitt 3.1 beschrieben.

FA20 *Agile Projekte*

Die Instanz des Frameworks soll auf Projekte mit agilen Vorgehensweisen zugeschnitten sein. Beispiele für agile Methoden sind Extreme Programming (XP) [BA04] oder Scrum [SB02].

FA21 *Integration existierender Werkzeuge*

Es sollten vorhandene Werkzeuge integriert werden, die gängig und weit verbreitet sind. Dabei können auch mehrere Werkzeuge mit gleicher oder überlappender Funktionalität zur Verfügung gestellt werden.

FA22 *Integration von Eigenentwicklungen*

Die Werkzeuge des Lehrstuhls sollen als Services integriert werden. Dazu gehören unter anderem MontiCore [Kra10] und MontiArc [HRR12].

FA23 *Interaktion der Werkzeuge*

Viele Werkzeuge bieten Mechanismen zur Kopplung bzw. Interaktion mit anderen Werkzeugen. Bei der Integration der Werkzeuge als Service sollen diese vorhandenen Integrationsmechanismen erhalten bleiben.

FA24 *Unterstützung existierender Clients für Werkzeuge (Service-Clients)*

Für viele Werkzeuge existieren bereits Clients für unterschiedliche Plattformen. Diese Clients sollen auch nach der Integration eines Werkzeugs als Service unverändert nutzbar sein. Diese Clients werden als Service-Clients bezeichnet, da sie direkt mit einem integrierten Werkzeug kommunizieren.

FA25 *Nutzung externer Werkzeuge*

Werkzeuge, die nicht in die SSELab-Instanz integriert worden sind, aber Schnittstellen zur Kopplung mit integrierten Werkzeugen bieten, sollen trotzdem in Kombination mit den integrierten Werkzeugen verwendet werden können.

FA26 *Datenintegration über ein Versionsverwaltungssystem*

Ein zentraler Service soll ein Versionsverwaltungssystem sein, über dass alle Artefakte eines Software-Entwicklungsprojekts versioniert abgelegt werden können. Das System soll über E-Mails Benachrichtigungen bei Änderungen versenden können und ein webbasiertes Frontend bereitstellen.

FA27 *Zentraler Speicherplatz zum Austausch unversionierter Dokumente*

Neben dem Versionsverwaltungssystem soll ein Werkzeug integriert werden, mit dessen Hilfe Dateien unversioniert zentral gespeichert werden können. Beispiele für solche Dateien sind große Binärdateien, die nicht oder nur selten verändert werden und daher nicht versioniert werden müssen.

FA28 *Bereitstellung eines Projektmanagementwerkzeugs*

Es soll ein Projektmanagementwerkzeug als Service zur Verfügung stehen über das Meilensteine und Aufgaben geplant und verwaltet werden können.

FA29 *Integration eines Bug-Trackers*

Es soll ein Bug-Tracker als Service integriert werden, um Fehler und Verbesserungsvorschläge in Projekten dokumentieren zu können.

FA30 *Interne Projektdokumentation über ein Wiki*

Für die interne Projektdokumentation soll ein Wiki als Service bereitgestellt werden.

FA31 *Webseiten von Projekten*

Für die Außendarstellung von Projekten soll es die Möglichkeit geben, statische Web-Inhalte bereitzustellen.

FA32 *Kommunikation*

Für die Kommunikation der Projektbeteiligten sollen Mailing-Listen als Service bereitgestellt werden.

FA33 *Single-Sign-On über Web-Browser*

Ein Benutzer soll sich nur einmalig an der SSELab-Instanz anmelden müssen und dann ohne erneute Authentifizierung sowohl auf die zentrale Web-Oberfläche als auch auf die integrierten Werkzeuge aller seiner Projekte zugreifen können.

FA34 *GUI-Integration*

Die integrierten webbasierten Werkzeuge sollen angepasst werden, so dass sie eine möglichst einheitliche Oberfläche besitzen. Die Anpassung soll jedoch nur erfolgen, wenn die Werkzeuge einen entsprechenden Mechanismus bereitstellen.

Nichtfunktionale Anforderungen

NFA8 *Performance der Services*

Die Performance der integrierten Werkzeuge soll gut sein.

3.3.3 Anforderungen an die SSELab-Instanz aus Sicht der Administratoren

In diesem Abschnitt werden die Anforderungen an die SSELab-Instanz aus Sicht der Administratoren und System-Administratoren strukturiert aufgeführt, die die wichtigsten Funktionen der zentralen Web-Oberfläche beschreiben.

Funktionale Anforderungen

FA35 *Zentrale Administration des Systems*

Die SSELab-Instanz mit allen registrierten Benutzern sowie die zur Verfügung gestellten Services und Clients und alle Projekte sollen über eine zentrale Web-Oberfläche per Browser administriert und konfiguriert werden können. Es soll nicht notwendig sein, dazu zusätzliche Software zu installieren.

FA36 *Automatisierte Konfiguration*

Die Konfiguration der Services soll automatisiert über die SSELab-Instanz erfolgen. Es soll möglichst keine manuelle Konfiguration einzelner Services durch einen System-Administrator notwendig sein.

FA37 *Verwaltung von Administrator-Accounts*

Ein Administrator soll sowohl alle relevanten Daten seines eigenen Accounts verwalten, als auch Administratorkonten anlegen und löschen können. Dabei muss beachtet werden, dass im System immer mindestens ein Administratorkonto vorhanden ist.

FA38 *Verwaltung von Benutzer-Accounts*

Ein Administrator soll Benutzer-Accounts anlegen, editieren und löschen können. Darüber hinaus soll es die Möglichkeit geben, mehrere redundante Accounts in einem einzigen Account zusammenzufassen. Dabei ist der resultierende Benutzer Mitglied in allen Projekten, in denen die ursprünglichen Accounts enthalten waren. Dies soll unter Berücksichtigung der Rolle eines Benutzers innerhalb der Projekte erfolgen.

FA39 *Verwaltung von Projekten*

Ein Administrator soll die Möglichkeit haben, alle Projekt zu verwalten. Dazu gehört, dass neue Projekte erzeugt und existierende Projekte aktiviert, deaktiviert oder gelöscht werden können. Außerdem sollen Benutzer zu beliebigen Projekten als Manager oder als Member hinzugefügt oder aus einem Projekt entfernt werden können. Zusätzlich soll ein Administrator die Rolle von Benutzern in einem Projekt ändern und Services für ein Projekt aktivieren oder deaktivieren können.

FA40 *Verwaltung von Services*

Ein Administrator soll Services zur Laufzeit des Systems installieren, aktualisieren und deinstallieren können. Es soll möglich sein, installierte Services global im System zu aktivieren, so dass diese in den Projekten durch die jeweiligen Manager aktiviert werden können. Installierte Services sollen darüber hinaus deaktiviert werden können, so dass diese nicht mehr durch Manager aktivierbar sind. Wenn ein deaktivierter Service in einem Projekt bereits aktiviert worden ist, soll er jedoch weiterhin verwendet werden können.

FA41 *Erstellung von Projekttemplates*

Ein Administrator soll Templates für Projekte erstellen können, die eine bestimmte Menge von Services zusammenfassen und dadurch bestimmte Arten von Projekten beschreiben.

FA42 *Verwaltung und Verwendung von Clients*

Ein Administrator soll Framework-Clients über die Web-Oberfläche hochladen, aktivieren sowie deaktivieren und entfernen können. Zu jedem Client soll eine Beschreibung angegeben werden können. Ein Administrator soll außerdem die im System verfügbaren Clients für Administratoren herunterladen und nutzen können.

FA43 *Kommunikationsmechanismen*

Ein Administrator soll Nachrichten an Benutzer, Benutzergruppen oder Administratoren senden können – beispielsweise zur Ankündigung von Wartungsarbeiten. Diese Nachrichten sollen auch auf den Webseiten des Systems angezeigt werden können.

FA44 *Monitoring*

Das System soll über Monitoring-Funktionen verfügen, so dass die Administratoren bzw. System-Administratoren eine Benachrichtigung erhalten, falls Teile des Systems ausfallen oder nicht mehr erreichbar sind.

FA45 *Suche*

Ein Administrator soll nach Benutzern, Projekten und Services suchen können.

Nichtfunktionale Anforderungen

NFA9 *Sicherer Zugriff von Administratoren*

Die administrativen Seiten der Web-Oberfläche sollen nur nach erfolgreicher Authentifizierung mit einem gültigen Administrator-Account erreichbar sein. Zur Authentifizierung werden ein Benutzername und ein Passwort benötigt.

NFA10 *Skalierbarkeit des Systems*

Die SSELab-Instanz soll eine möglichst große Anzahl von Services, Benutzern und Projekten ohne Beeinträchtigung der Nutzbarkeit der Instanz unterstützen.

3.3.4 Anforderungen an das SSELab-Framework

In diesem Abschnitt werden die wichtigsten Anforderungen an das SSELab als Framework aufgeführt, die sich teilweise aus der Diskussion der aktuellen Trends und Herausforderungen im Bereich von Entwicklungsumgebungen aus Kapitel 1 ergeben. Einige der Anforderungen sind aus Sicht der Service- und der Client-Entwickler geschrieben. Andere Anforderungen beschreiben grundlegende Eigenschaften des Frameworks. Alle Anforderungen dieses Abschnitts sind als Architekturtreiber [BCK03] zu verstehen, die zusätzlich zu den Anforderungen der vorigen Abschnitte das Design des Frameworks maßgeblich beeinflussen und durch die dessen Funktionalität und die Erweiterungsmechanismen definiert werden.

Funktionale Anforderungen

FA46 *Application-Framework für Projektportale und CDEs*

Das Framework soll als Anwendungsframework [FS97] für Projektportale und CDEs entwickelt werden.

FA47 *Entwicklung als webbasiertes Framework*

Das Framework soll als ein verteiltes webbasiertes System entwickelt werden. Es soll bereits lauffähig und nutzbar sein, ohne dass konkrete Erweiterungen in Form von Services oder Clients verfügbar sind. Die Entwicklung erfolgt dementsprechend in einer Mischung aus Anwendungs- und Framework-Entwicklung [Rat11].

FA48 *Unterstützung verschiedener Werkzeugarten*

Das Framework soll eine Integration unterschiedlicher Werkzeugarten unterstützen. Es soll unter anderem eine Nutzung von Legacy-Systemen, Standardsoftware und Eigenentwicklungen in Form von Web-, Web-Service-, Desktop- und Kommandozeilenanwendungen ermöglichen.

Da sich die verfügbaren Werkzeuge kontinuierlich weiterentwickeln und regelmäßig neue Werkzeuge entstehen, müssen die integrierten Werkzeuge leicht austauschbar und aktualisierbar sein. Daraus ergibt sich außerdem, dass die Werkzeuge nicht oder nur minimal für eine Integration angepasst werden sollten.

FA49 *Zentralisierung der integrierten Werkzeuge als Services (Hosted Services)*

Alle Werkzeuge, die über die Mechanismen des Frameworks integriert werden, sollen zentral als Service angeboten bzw. genutzt werden können. Die Nutzung eines Services soll nicht eine Installation von zusätzlicher Software beim Benutzer erfordern.

FA50 *Nutzung vorhandener Integrationsmechanismen der Werkzeuge*

Es existieren viele verschiedene Integrationsansätze [ADDK03, ADS02]. Daher soll das Framework vorhandene Integrationsmechanismen unterstützen, so dass diese auch nach der Integration von Werkzeugen in das Framework erhalten bleiben.

FA51 *Unterstützung eines mehrstufigen Integrationskonzepts*

Die Integration von Werkzeugen sollte in mehreren Schritten erfolgen. Dabei sollten auch optionale Schritte ausgelassen werden können – beispielsweise die Integration auf Ebene der Benutzeroberfläche. Dadurch wird eine inkrementelle Integration eines Werkzeugs möglich, so dass frühzeitig Feedback der Endanwender eingeholt werden kann.

FA52 *Entwicklung von Integrationskomponenten für Werkzeuge*

Die Entwicklung der Integrationskomponenten für Werkzeuge soll ohne Kenntnis des Aufbaus des Frameworks möglich sein. Dazu soll eine minimale Menge von Bibliotheken zur Verfügung gestellt werden. Diese Bibliotheken sollen einerseits eine einfache Realisierung der Integrationskomponenten ermöglichen, jedoch auch reichhaltige APIs für wiederkehrende Aufgaben bei der Entwicklung bereitstellen.

FA53 *Verwaltung von Services zur Laufzeit*

Da das Framework als verteiltes webbasiertes System entwickelt wird, sollen die Instanzen des Frameworks im Dauerbetrieb laufen. Daher muss das Framework so realisiert sein, dass Services zur Laufzeit einer Instanz hinzugefügt, aktualisiert und wieder deinstalliert werden können.

FA54 *Unabhängige Entwicklung von Framework-Clients*

Die Entwicklung von Framework-Clients soll von der Entwicklung des Frameworks entkoppelt und ohne Kenntnis dessen Aufbaus möglich sein. Dazu soll das Framework Web-Service-Schnittstellen und darauf aufbauen APIs anbieten, die eine einfache Entwicklung von Clients unterstützen.

FA55 *Bereitstellung von Benutzerinformationen zur Authentifizierung*

Das Framework soll über eine zentrale Verwaltung der Benutzer-Accounts verfügen.

Darüber hinaus soll es den integrierten Werkzeugen möglich sein, auf die Account-Informationen über einen LDAP-Dienst zuzugreifen und darüber die Authentifizierung zu realisieren.

Nichtfunktionale Anforderungen

NFA11 *Flexibles Deployment*

Das Framework soll ein flexibles Deployment aller Komponenten ermöglichen, so dass ein Deployment auf einem oder mehreren realen oder virtuellen Servern oder auch auf Cloud-Infrastrukturen möglich ist.

NFA12 *Sichere Kommunikation innerhalb des Frameworks*

Da die Komponenten einer Instanz des Frameworks über mehrere Servermaschinen verteilt werden können, soll die Kommunikation zwischen allen Komponenten verschlüsselt und unter Verwendung von Zertifikaten erfolgen.

3.4 Zusammenfassung

In diesem Kapitel wurden die wichtigsten Anforderungen an das SSELab angegeben. Dazu wurden zunächst anhand fiktiver, aber typischer Projekte einige Szenarien beschrieben und im Anschluss daran die Rollen definiert, die Personen bei der Interaktion mit dem SSELab annehmen können. Wesentlich für die Diskussion der Anforderungen in diesem Kapitel sind die unterschiedlichen Sichtweisen auf das SSELab. Die beiden Sichtweisen und die zugehörigen Anforderungen führen zu einer Mischung aus Anwendungs- und Framework-Konzeption, bei der gleichzeitig ein lauffähiges webbasiertes System und ein Framework entwickelt werden. In den folgenden Kapiteln wird auf die hier beschriebenen Anforderungen jeweils Bezug genommen und deren Auswirkungen auf Design-Entscheidungen diskutiert.

Kapitel 4

Architektur eines Frameworks für webbasierte Projektportale und CDEs

Aufbauend auf den in Kapitel 1 diskutierten Trends und Herausforderungen im Kontext von Entwicklungsumgebungen wurden im vorigen Kapitel 3 die wichtigsten Anforderungen und Architekturtreiber [BCK03] definiert, auf deren Grundlage das SSELab entworfen und umgesetzt worden ist. Daran anknüpfend gibt dieses Kapitel eine detaillierte Übersicht über die Konzepte und die Architektur des SSELabs als Framework. Dazu wird im folgenden Abschnitt zunächst der Teil des Domänenmodells beschrieben, der die konzeptionelle Basis für alle weiterführenden Betrachtungen bildet. Danach folgt eine kompakte Darstellung der Framework-Architektur, auf deren Grundlage die Details aller Komponenten und deren Interaktionen in den weiteren Abschnitten diskutiert werden.

In diesem Kapitel stehen die innere Architektur und die zugrunde liegenden Konzepte des Frameworks im Vordergrund. Im folgenden Kapitel 5 werden darauf aufbauend die Methodik und die Mechanismen zur Erweiterung des Frameworks diskutiert.

4.1 Modellierung der Domäne von Projektportalen

Frameworks sind abstrakte Lösungen für wiederkehrende Probleme in bestimmten Anwendungsbereichen bzw. Domänen in Form von wiederverwendbaren Architekturen. Dementsprechend enthält der Kern jedes Frameworks ein Modell der Domäne, das im Rahmen einer Domänenanalyse identifiziert worden ist [RJ96]. In diesem Abschnitt werden die grundlegenden Konzepte des SSELab-Frameworks anhand des Domänenmodells beschrieben. In den nachfolgenden Abschnitten werden dann weitere Details des Domänenmodells im Kontext der Architekturbeschreibung erläutert.

In einem webbasierten Projektportal arbeiten mehrere Personen im Rahmen von Projekten zusammen und verwenden zur Bearbeitung ihrer Aufgaben verschiedene Werkzeuge. Da das SSELab gemäß Anforderung **FA46** als ein Framework für solche Projektportale konzipiert worden ist, enthält der Kern des Domänenmodells Klassen zur Beschreibung von *Werkzeugen*, *Personen* und *Projekten*. Im SSELab werden die Werkzeuge zentral als Hosted Services bereitgestellt (vgl. Anforderung **FA49**), so dass sie im Folgenden auch als *Services* bezeichnet werden. In diesem Abschnitt werden diese Klassen des Domänenmodells eingeführt, da sie die Grundlage für alle weiteren Betrachtungen dieses Kapitels bilden. Die Begriffe Service, Person und Projekt werden in vielen Kontexten unterschiedlich verwendet, so dass zunächst jeweils eine kurze Definition sowie einige Beispiele angegeben werden, um ein einheitliches Verständnis zu ermöglichen.

4.1.1 Services

Als *Service* wird in diesem Kontext ein Werkzeug bzw. eine Anwendung oder auch eine einzelne Funktion einer Anwendung bezeichnet, die in das Framework integriert und darüber genutzt werden soll. Beispiele für Services, die im Rahmen von Software Engineering Projekten eingesetzt werden, sind Versionsmanagement-Systeme [Lou06b] wie Subversion [PCF08], Continuous Integration Server [Fow06] wie Jenkins [Sma11], Wikis [Lou06a] wie MediaWiki [MW12], Codegeneratoren wie MontiCore [Kra10] oder aber auch Funktionen, die von sozialen Netzwerken wie LinkedIn [Lin12] angeboten werden.

Der Service-Begriff ist hier bewusst weit gefasst, da das SSELab als erweiterbares Framework konzipiert ist, das gemäß Anforderung **FA48** ein große Anzahl und Vielfalt von Werkzeugen unterstützen soll. Eine solch abstrakte Definition des Service-Begriffs im Rahmen der Domänenmodellierung schafft erst die Grundlage für die Unterstützung vielfältiger Werkzeuge. Da jedoch diese Werkzeuge völlig unterschiedliche Eigenschaften besitzen können, wurden im Rahmen dieser Arbeit mehrere Service-Kategorien identifiziert, die die charakteristischen Eigenschaften verwandter Werkzeuge repräsentieren und dadurch deren effektive Integration ermöglichen. Abbildung 4.1 zeigt drei der Kategorien in einem Klassendiagramm.

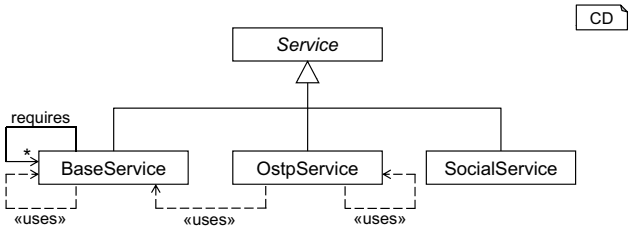


Abbildung 4.1: Service-Kategorien des SSELab-Frameworks

Die charakteristischen Eigenschaften von Werkzeugen der drei Service-Kategorien werden im Folgenden beschrieben. Dabei ergeben sich jeweils auch Anforderungen an das Framework bezüglich der Integration von Werkzeugen jeder Kategorie.

- In der Kategorie *Base-Service* werden server- bzw. webbasierte Anwendungen zusammengefasst, die demzufolge klassische Hosted Services sind. Diese Anwendungen werden in [BB03] als Kerninfrastruktur von Software Engineering Projekten bezeichnet, so dass sie grundlegend für die Durchführung von Projekten sind und daher als eigene Kategorie im Domänenmodell repräsentiert werden. Einige Beispiele für Werkzeuge in dieser Kategorie sind Subversion, Jenkins und MediaWiki.

  Charakteristisch für diese Art von Anwendungen ist, dass sie eine Serverinfrastruktur zur Ausführung benötigen. Für die Verwendung der oben genannten Werkzeuge werden beispielsweise mindestens ein Web-, ein Application- und ein Datenbank-Server benötigt. Darüber hinaus verfügen diese Anwendungen in vielen Fällen über eine eigene Benutzeroberfläche, ein eigenes Rechtemanagement für die Authentifizierung und Autorisierung von Zugriffen und eine Komponente zur Verwaltung ihre persistenten Daten. Anwendungen

dieser Kategorie können außerdem aufeinander aufbauen, indem sie bestimmte andere Anwendungen voraussetzen oder optional verwenden können.

- Dateiverarbeitende Transformationswerkzeuge werden als *Ostp-Services* [Her06] bezeichnet. Beispiele dafür sind Konvertierungswerkzeuge oder Codegeneratoren wie MontiCore, die eine Menge von Eingabedateien verarbeiten und eine Menge von Ausgabedateien produzieren. Um das unterschiedlichen Ein- und Ausgabeverhalten von solchen Werkzeugen zu unterstützen, werden Ostp-Services noch in weitere Subkategorien aufgegliedert.

  Charakteristisch für diese Werkzeuge ist, dass sie üblicherweise nicht als Services, sondern als Desktop- oder Kommandozeilenanwendungen genutzt werden. Die Motivation für die Definition dieser Kategorie ist daher, auch solche Werkzeuge servicebasiert über das Framework anbieten zu können. Weitere charakteristische Eigenschaften dieser Werkzeuge sind, dass sie ebenfalls auf anderen Werkzeugen aufbauen können und im Gegensatz zu Base-Services neben den Ausgabedateien keine persistenten Daten verwalten.

- In der Kategorie *Social-Service* werden Funktionen sozialer Netzwerke oder anderer webbasierter Projektportale zusammengefasst, die genutzt werden können, um beispielsweise Profilinformationen von Benutzern zu importieren und zu verarbeiten. Die Motivation für diese Kategorie ist, dass solche Informationen in verteilten Projekten zum Aufbau von Vertrauen zwischen Entwicklern genutzt werden können [PVEP10, ACGL08].

  Charakteristisch für diese Kategorie ist, dass üblicherweise vor der Nutzung solcher Funktionen zunächst eine Authentifizierung des Benutzers bei dem entsprechenden Portal erfolgen muss, damit anschließend die gewünschten Informationen über eine dediziert Schnittstelle abgerufen werden können.

Durch diese drei Service-Kategorien kann bereits eine Vielzahl von Anwendungen und Werkzeugen repräsentiert werden. Darüber hinaus wurden im Rahmen dieser Arbeit noch weitere sinnvolle Service-Kategorien identifiziert, die durch die Erweiterungsmechanismen des Frameworks unterstützt werden könnten. Ein Beispiel dafür ist eine Kategorie für webbasierte IDEs in Anlehnung an [vDMC+10], durch die Werkzeuge wie Compiler, Generatoren, Debugger und Testumgebungen repräsentiert sowie deren interaktive servicebasierte Nutzung beschrieben werden kann. Anhand dieses Beispiels wird auch deutlich, dass Werkzeuge durchaus mehreren Kategorien zugeordnet werden können. Beispielsweise könnte der MontiCore-Generator sowohl als Ostp-Service als auch als Service einer webbasierten IDE genutzt werden.

Insgesamt sollte bei der Definition einer neuen Service-Kategorie darauf geachtet werden, dass die Eigenschaften möglichst vieler Werkzeuge durch die Kategorie zusammengefasst werden. Sinnvoll ist es daher, von einer ausreichend großen Menge repräsentativer Werkzeuge zu abstrahieren. Nach der Definition einer Kategorie kann dann das Framework erweitert werden. Die entsprechenden Konzepte und die zugehörige Methodik werden ausführlich in Kapitel 5 beschrieben. Die Verwaltung von Services der hier vorgestellten Kategorien wird in diesem Kapitel in den Abschnitten 4.3 und 4.4.2 diskutiert.

4.1.2 Personen

Im Kontext von Projekten arbeiten mehrere *Personen* zusammen. Diese Personen werden im Domänenmodell des Frameworks durch einen entsprechenden Kontotyp repräsentiert. Einen weiteren Kontotyp bilden die sogenannten *Bot-Accounts* für den automatisierten Zugriff auf Services und Funktionen einer Framework-Instanz. Abbildung 4.2 zeigt diese beiden Kontotypen in einem Klassendiagramm.

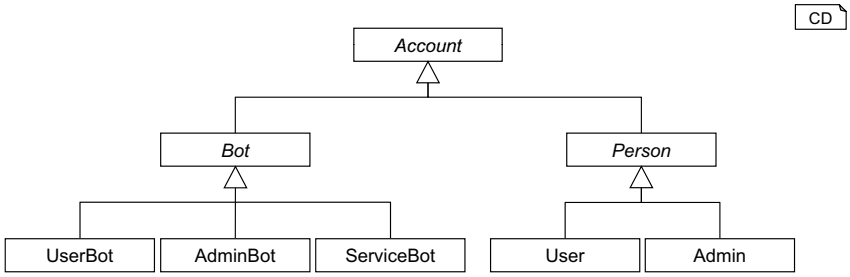


Abbildung 4.2: Kontotypen des SSELab-Frameworks

Personengebundene Konten erlauben einen interaktiven Zugriff auf die Funktionen einer Framework-Instanz. Dabei werden durch die Klassen *User* und *Admin* die beiden in Abschnitt 3.2 definierten Rollen Benutzer und Administrator unterschieden. Benutzer können gemäß den definierten Anforderungen unter anderem Projekte erzeugen und die verfügbaren Services nutzen. Administratoren können alle Benutzer, Services und Projekte verwalten. Weitere Rollen, die beispielsweise Projekte mit ihren konkreten Prozessen, Methoden und Verantwortlichkeiten berücksichtigen, wurden hier bewusst nicht definiert, da das Domänenmodell eine prozessagnostische Sicht auf Projekte beschreibt. Die Unterstützung konkreter Prozesse und der zugehörigen Rollen innerhalb von Projekten soll durch die Integration geeigneter Werkzeuge oder die Verwendung von Werkzeugen gemäß den Methoden und Prozessen erfolgen. Daher ist das Domänenmodell auch in Bezug auf die Beschreibung von Personen abstrakt gehalten.

Für den automatisierten Zugriff auf eine SSELab-Instanz werden verschiedene Arten von *Bot-Accounts* verwendet (vgl. Anforderung **FA13**). Ein Bot-Account kann entweder stellvertretend für einen Administrator, einen Benutzer oder einen Service eingesetzt werden. Dementsprechend werden drei Arten von Bot-Accounts unterschieden: Admin-Bots, User-Bots und Service-Bots. Diese Konten werden entweder bei Bedarf automatisch bereitgestellt oder können von Benutzern bzw. Administratoren manuell erzeugt werden. Der Einsatz von Bot-Accounts sowie die Verwaltung personengebundener Konten werden in Abschnitt 4.4.3 genauer behandelt.

4.1.3 Projekte

Im Domänenmodell wird durch ein *Projekt* eine Menge von Konten und Services zusammengefasst. Die wichtigsten Beziehungen zwischen Projekten und den zuvor vorgestellten Klassen des Domänenmodells sind in Abbildung 4.3 dargestellt.

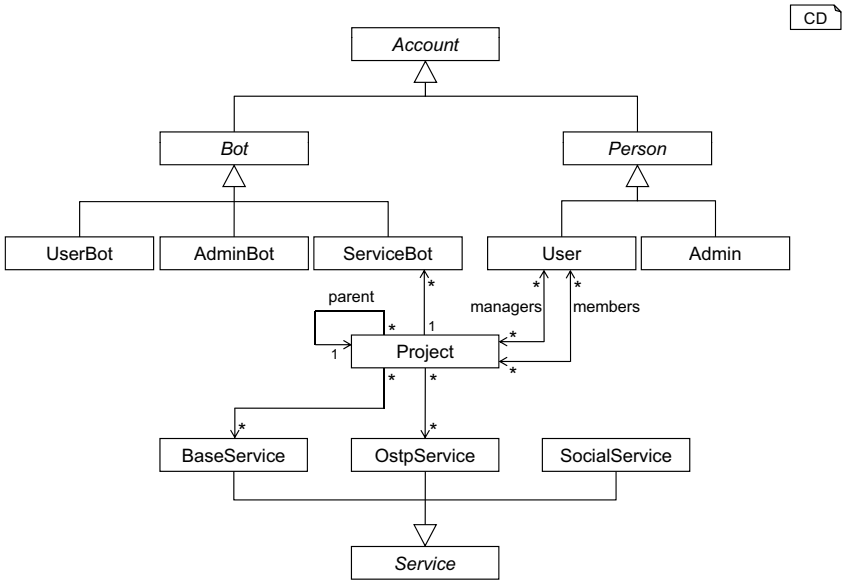


Abbildung 4.3: Projekte und deren Beziehungen zu Konten und Services

Ein Benutzer kann Teilnehmer in beliebig vielen Projekten sein. Innerhalb eines Projekts besitzt er entweder die Rolle *Manager* oder *Member*. Diese beiden Rollen entsprechen den in Abschnitt 3.2 definierten Rollen. Manager können administrative Aufgaben innerhalb eines Projekts übernehmen, wie beispielsweise die Verwaltung der anderen Benutzer im Projekt oder die Aktivierung von Services für das Projekt. Member können dagegen keine administrativen Tätigkeiten durchführen, sondern nur die Services des Projekts nutzen. Aus den im vorigen Abschnitt beschriebenen Gründen wurden auch in diesem Fall über die administrativen Rollen hinaus keine weiteren Rollen definiert, so dass Projekte unabhängig von deren konkreten Vorgehensweisen und spezifischen Rollen durch das Domänenmodell repräsentiert werden können.

Zusätzlich zu den personengebundenen Konten können auch noch Bot-Accounts einem Projekt zugeordnet werden. Dabei kommen Service-Bots zum Einsatz, die einen automatisierten Zugriff auf die Services des Projekts erlauben. Jeder dieser Service-Bots ist eindeutig einem konkreten Projekt zugeordnet und kann ausschließlich für die Service-Instanzen dieses Projekts verwendet werden.

Die Services der zuvor beschriebenen Kategorien werden in verschiedenen Kontexten verwendet. Base-Services werden ausschließlich innerhalb von Projekten genutzt. Ostp-Services können dagegen im Rahmen von Projekten, aber auch unabhängig davon aufgerufen werden, so dass für deren Nutzung nicht unbedingt ein Projekt benötigt wird. Social-Services werden nur unabhängig von Projekten von jedem Benutzer individuell verwendet, so dass es keine direkte Beziehung zwischen diesen Services und den Projekten gibt.

Eine weitere Eigenschaft von Projekten ist die hierarchische Schachtelung. Jedes Projekt kann über ein übergeordnetes Projekt verfügen, das wiederum selbst eine Menge von Benutzern und Services zusammenfasst. Dadurch können die Beziehungen zwischen Teilprojekten durch das Domänenmodell repräsentiert werden. Die Verwaltungskonzepte von Projekten, die auf den hier eingeführten Eigenschaften des Domänenmodells aufbauen, werden in Abschnitt 4.4.4 ausführlich diskutiert.

4.2 Übersicht über die Architektur

Dieser Abschnitt gibt eine Übersicht über die Komponenten des Frameworks und beschreibt deren Funktionen und Interaktionen innerhalb der Gesamtarchitektur. Darüber hinaus werden die Konzepte zur Erweiterung der Komponenten sowie zu deren Verteilungsmöglichkeiten einführend behandelt.

4.2.1 Komponenten des SSELab-Frameworks

Das SSELab ist als Framework konzipiert, das sich aus mehreren Komponenten zusammensetzt. Es besteht aus einem zentralen Frontend, mehreren Backends für die unterschiedlichen Service-Kategorien und Client-APIs für verschiedene Benutzergruppen. Abbildung 4.4 zeigt diese grundlegenden Komponenten und deren Beziehungen untereinander auf hoher Abstraktionsebene in einem Komponentendiagramm. Die wesentlichen Funktionen der einzelnen Komponenten werden in der folgenden Auflistung zusammengefasst.

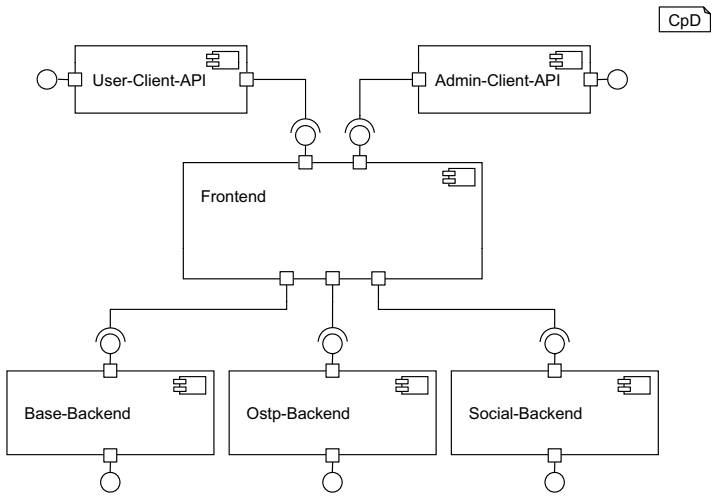


Abbildung 4.4: Komponentendiagramm der Architektur des SSELab-Frameworks

- Die *Backends* sind für die Konfiguration bzw. für die Ausführung der Services verantwortlich und bieten jeweils eine dedizierte Schnittstelle für die Kommunikation mit dem Frontend an. Im Rahmen dieser Arbeit wurden drei unterschiedliche Backend-Arten realisiert, die jeweils eine der in Abschnitt 4.1.1 beschriebenen Service-Kategorien unterstützen: Das Base-Backend ermöglicht die Instanziierung und Konfiguration der Base-Services für Projekte. Das Ostp-Backend bietet einen einheitlichen Zugriff auf Ostp-Services und das Social-Backend ermöglicht die Nutzung von Funktionen sozialer Netzwerke über Social-Services.

  Alle Backends basieren auf den Konzepten einer reinen Plug-in-Architektur [Bir05, Rat11] und ermöglichen die Verwaltung von Plug-ins zur Laufzeit. Durch jedes Plug-in wird jeweils ein spezifisches Werkzeug als Service in das Framework integriert, so dass sie auch als *Service-Plug-ins* bezeichnet werden. Die Entwicklung der Plug-ins erfolgt dabei auf Grundlage spezifischer APIs, die auf die Services jeder Backend-Art zugeschnitten sind. Abschnitt 4.3 erläutert die Konzepte und Eigenschaften aller Backend-Arten sowie den Einsatz von Plug-in-Systemen und darauf aufbauend die Mechanismen zur Integration, Konfiguration und Ausführung der Services.

- Das *Frontend* ist die zentrale webbasierte Administrationskomponente, in der alle relevanten Daten der Services, Benutzer und Projekte sowie aller Backends und der Clients verwaltet werden. Es ermöglicht einen direkten Zugriff per Browser sowohl auf die administrativen Funktionen als auch auf die verfügbaren Services. Darüber hinaus bietet es mehrere Schnittstellen an, über die Clients mit dem Frontend kommunizieren können. Über diese Schnittstellen werden jeweils Teilmengen der Funktionen des Frontends für bestimmte Benutzergruppen zusammengefasst und angeboten. Beispiele für solche Funktionen sind die Installation oder Aktualisierung von Service-Plug-ins durch Administratoren oder die Nutzung von Services durch die Benutzer.

  In Abschnitt 4.4 wird zunächst die Architektur des Frontends genauer beschrieben. Darauf aufbauend werden die wichtigsten Verwaltungsmechanismen diskutiert.

- Die *Client-APIs* ermöglichen den Zugriff auf die Schnittstellen zur Kommunikation mit dem Frontend. Im Rahmen dieser Arbeit wurden mehrere Framework-Clients auf Grundlage der APIs realisiert, die in unterschiedlichen Szenarien und Kontexten eingesetzt werden können. Beispielsweise ermöglichen sie Benutzern die Verwendung von Services per Kommandozeile oder aus einer IDE heraus. Darüber hinaus wurden Clients zur Unterstützung administrativer Aufgaben entwickelt. In Abschnitt 4.5 werden die Konzepte der Client-APIs sowie die realisierten Framework-Clients genauer beschrieben.

Das Ziel des Frameworks ist, ein möglichst breites Spektrum und eine große Anzahl von Services zu unterstützen, so dass die Architektur entsprechend auf die dafür benötigte Flexibilität und Skalierbarkeit ausgerichtet wurde. Die Entkopplung der Backends und Clients von dem Frontend ist eine grundlegende Architekturentscheidung, die diese Eigenschaften unterstützt. Die darauf aufbauenden Konzepte zur Erweiterbarkeit und Verteilung werden nachfolgend eingeführt.

4.2.2 Erweiterbarkeit der Komponenten

Die Erweiterbarkeit des Frameworks ist ein wesentlicher Architekturtreiber, der sich in dem Design aller Komponenten widerspiegelt. Zwei grundlegende Konzepte, die die Erweiterbarkeit ermöglichen, sind die strikte Trennung und die daraus resultierende Entkopplung des Frontends von den Clients und von den unterschiedlichen Backends sowie der Einsatz einer zur Laufzeit erweiterbaren Architektur in allen Backend-Arten.

Die Bereitstellung der Client-APIs und deren Entkopplung vom Frontend ermöglichen eine unabhängige Entwicklung von Clients. Darüber hinaus besteht durch die Entkopplung der Verwaltungsmechanismen im Frontend von der Konfiguration und Ausführung der Services in den Backends die Möglichkeit, beliebig viele Instanzen jeder Backend-Art zur Laufzeit des Frontends zu unterstützen, so dass eine skalierbare Architektur entsteht. Andererseits erlaubt die Entkopplung, neue Backend-Arten zu realisieren und dadurch neue Service-Kategorien zu unterstützen. Im Rahmen dieser Arbeit wurden beispielsweise zunächst das Base-Backend und das Ostp-Backend umgesetzt. Später wurde das Social-Backend hinzugefügt. Im Hinblick auf die Unterstützung der Erweiterbarkeit des Frameworks um neue Backend-Arten musste jedoch ein Kompromiss gefunden werden. Dies resultiert daraus, dass sich der Funktionsumfang einer Backend-Art stark auf die graphische Benutzeroberfläche sowie auf die Clients auswirkt. Da jedoch die Benutzbarkeit der Oberflächen wichtig ist und unter Berücksichtigung der Tatsache, dass neue Backend-Arten nur relativ selten hinzugefügt werden müssen, wurde im Rahmen der Arbeit darauf verzichtet, diesen Erweiterungspunkt des Frameworks stärker auszubauen. Dadurch ist ein tieferes Verständnis der Framework-Architektur zur Erweiterung notwendig.

Die Erweiterbarkeit des Systems zur Laufzeit ist ein wichtiges Konzept, das eine Integration von Services ermöglicht, ohne dass der Betrieb aktiver Projekte und bereits vorhandener Services beeinträchtigt wird. Dieses Konzept wird im Framework durch den Einsatz von Plug-in-Systemen in den Backends realisiert. Andere Mechanismen, die eine Erweiterung zur Laufzeit erlauben, wären hier jedoch auch nutzbar.

In Kapitel 5 werden die hier nur grob skizzierten Erweiterungsmechanismen im Detail beschrieben. Der Fokus liegt dabei auf den zugrunde liegenden Konzepten und der methodischen Verwendung der Erweiterungsmechanismen. Darauf aufbauend werden in Kapitel 6 einige der realisierten Services und Clients vorgestellt.

4.2.3 Verteilung der Komponenten

Durch ein Deployment der Komponenten des Frameworks auf einer Serverinfrastruktur entsteht eine Instanz, über die konkrete Benutzer, Services und Projekte verwaltet werden können. Gemäß Anforderung **NFA11** (Flexibles Deployment) und **FA47** (Entwicklung als webbasiertes Framework) wurde das SSELab-Framework so konzipiert, dass es verschiedene Verteilungsvarianten unterstützt. Beispielsweise ist es möglich, die Frontend- und die Backend-Instanzen auf einer einzelnen oder auf mehreren Servermaschinen auszuführen. Die Architektur des Frameworks erlaubt dabei sowohl die Nutzung herkömmlicher Serverumgebungen (reale oder virtuelle Maschinen) als auch von Cloud-Infrastrukturen [AFG+10]. Darüber hinaus ermöglicht die Entkopplung des Frontends von den Backends und den Clients ein unabhängiges Deployment aller Komponenten, so dass auch heterogene Serverumgebungen unterstützt werden können.

Durch die Verteilungsvarianten der Komponenten des Frameworks ist es beispielsweise möglich, das Hosting der Backends und damit der Projektdaten vom Hosting der Administrationsoberfläche zu trennen. Darüber hinaus kann eine Instanz für mehrere oder für einzelne Kunden betrieben werden. Durch die Kombination dieser Möglichkeiten ergeben sich verschiedene Betriebsarten. Außerdem ist es möglich, bereits laufende Werkzeuge durch eine a posteriori Integration von einer SSELab-Instanz verwalten zu lassen.

Die Methodik zur Bildung von Instanzen des Frameworks wird in Kapitel 7 diskutiert. Dabei werden auf Grundlage der Verteilungsvarianten sowohl die hier eingeführten Betriebsarten als auch die a posteriori Integration von Werkzeugen sowie die Laufzeitumgebung der Framework-Komponenten und deren Deployment beschrieben.

4.3 Verwaltung von Services in den Backends

In Kapitel 3 wurden die Anforderungen an das SSELab als Framework für webbasierte Projektportale und CDEs sowie als verteiltes webbasiertes System aus Sicht der Benutzer und Administratoren definiert. Diese Anforderungen werden durch unterschiedliche Konzepte in der Architektur des Frameworks erfüllt. Wie im vorigen Abschnitt 4.2.2 diskutiert, ist die Trennung des Frontends von den unterschiedlichen Backends, auf denen die eigentlichen Services konfiguriert und ausgeführt werden, ein zentrales Konzept, durch das einige wichtige Anforderungen umgesetzt werden können. Im Folgenden werden noch einmal die Anforderungen aufgelistet, die wesentliche Architekturtreiber für die Backends des Frameworks darstellen.

- **FA48** *Unterstützung verschiedener Werkzeugarten*
- **FA49** *Zentralisierung der integrierten Werkzeuge als Services (Hosted Services)*
- **FA50** *Nutzung vorhandener Integrationsmechanismen der Werkzeuge*
- **FA52** *Entwicklung von Integrationskomponenten für Werkzeuge*
- **FA53** *Verwaltung von Services zur Laufzeit*
- **FA11** *Konfiguration und Parametrisierung der Werkzeuge*
- **NFA11** *Flexibles Deployment*
- **NFA12** *Sichere Kommunikation innerhalb des Frameworks*

Aus dieser Liste der Anforderungen wurden sowohl die Architektur als auch die Funktionalität der Backends abgeleitet, die in den folgenden beiden Abschnitten diskutiert werden.

4.3.1 Architektur der Backends

Alle Backend-Arten wurden auf Grundlage der zuvor aufgelisteten Anforderungen und dementsprechend auf Grundlage gleicher Konzepte entwickelt und haben eine einheitliche Architektur. Sie unterscheiden sich jedoch in einigen Aspekten, wie beispielsweise den angebotenen APIs

zur Integration von Services, die auf die jeweilige Service-Kategorie zugeschnitten sind, so dass Anforderung **FA48** (Unterstützung verschiedener Werkzeugarten) erfüllt werden kann. In diesem Abschnitt werden die Gemeinsamkeiten der Architekturen aller Backend-Arten anhand des Base-Backends beschrieben. In Abbildung 4.5 ist dazu in einem Komponentendiagramm eine exemplarische Base-Backend-Instanz mit zwei installierten Plug-ins für die beiden Werkzeuge Subversion [PCF08, Col02] und Trac [Tra12] dargestellt. Das Subversion-Plug-in ist für die Konfiguration von Subversion-Repositories und das Trac-Plug-in entsprechend für die Verwaltung von Trac-Umgebungen verantwortlich. Auf die Integration von Subversion und Trac wird in Kapitel 6 genauer eingegangen, so dass diese Komponenten im Folgenden nicht weiter behandelt werden. Im Fokus steht anstatt dessen die innere Architektur des in Abbildung 4.5 dargestellten Base-Backends, die repräsentativ für alle Backend-Arten ist.

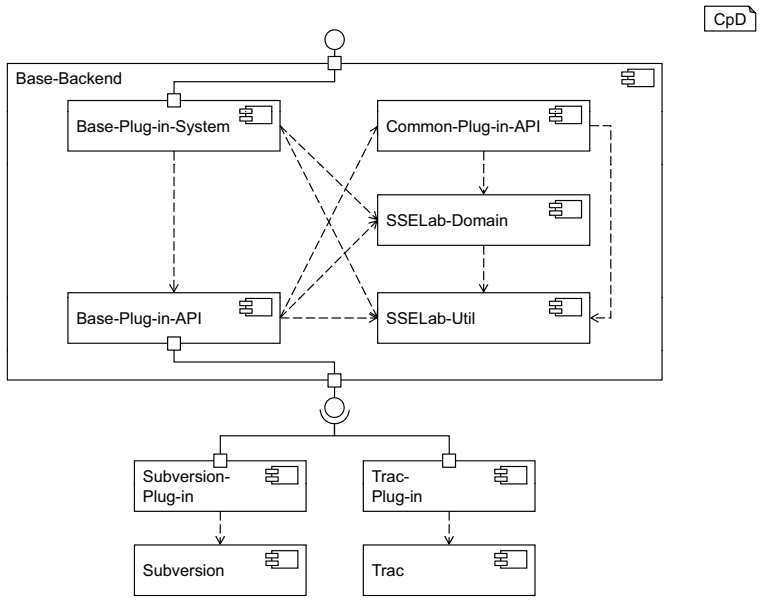


Abbildung 4.5: Base-Backend-Instanz mit zwei installierten Service-Plug-ins und den zugehörigen Services

Um ein möglichst flexible Architektur zu erreichen, wird in den Backends eine domänenspezifische Anpassung einer reinen Plug-in-Architektur verwendet [Rat08]. Das bedeutet, dass die gesamte Funktionalität eines Backends durch Plug-ins zur Verfügung gestellt wird, jedoch nicht alle zur Erweiterung genutzt werden können, sondern nur einige zentrale Plug-ins Erweiterungspunkte definieren. Das Zusammenspiel der einzelnen Komponenten bzw. Plug-ins im Base-Backend wird im Folgenden beschrieben.

SSELab-Util ist eine Komponente mit Hilfsfunktionalität, die sowohl in den Backends, dem

Frontend als auch in den Clients genutzt wird. Darin ist unter anderem Funktionalität zur Manipulation von Dateien, zur Ausführung von Betriebssystemprozessen und zur Nutzung von kryptografischen Algorithmen enthalten. Diese Komponente ist nur der Vollständigkeit halber hier aufgeführt und wird nicht genauer behandelt.

SSELab-Domain enthält eine Realisierung des Domänenmodells des SSELabs. Diese Komponente wird, wie auch SSELab-Util, in allen Teilen des Frameworks wiederverwendet. Aufbauend auf der Beschreibung in Abschnitt 4.1 werden weitere wesentliche Konzepte des Domänenmodells in Abschnitt 4.4 behandelt.

Common-Plug-in-API stellt die grundlegende API zur Entwicklung von Plug-ins für alle Backend-Arten zur Verfügung. Diese Komponente wird in allen Backends genutzt und erweitert, um backendspezifische Plug-in-APIs zu definieren. Der Aufbau und die Nutzung dieser Komponente werden in Kapitel 5 beschrieben.

Base-Plug-in-API fasst die Schnittstellen zusammen, die bei der Realisierung von Base-Service-Plug-ins genutzt werden müssen und definiert so den zentralen Erweiterungspunkt von Base-Backends. Die Definition der Schnittstellen erfolgt durch eine Erweiterung der Schnittstellen aus der Komponente Common-Plug-in-API. Der Aufbau und die Methodik zur Nutzung dieser sowie der APIs der anderen Backend-Arten werden ebenfalls ausführlich in Kapitel 5 beschrieben.

Base-Plug-in-System enthält die Kernfunktionalität des Base-Backends unter Verwendung der übrigen Komponenten und bietet diese dem Frontend über eine geeignete Schnittstelle an. Die grundlegenden Konzepte und die Funktionalität dieser Komponente werden im folgenden Abschnitt 4.3.2 genauer erläutert.

Gemäß den Definitionen aus [Mar01] realisieren die beiden Komponenten Base-Plug-in-API und Common-Plug-in-API die *Plug-in Definition*, enthalten also die von Plug-ins zu implementierenden Schnittstellen. Die beiden Komponenten SSELab-Domain und SSELab-Util bilden entsprechend das *Framework-Interface* für Plug-ins, stellen also Funktionalität bereit, die bei der Realisierung von Plug-ins genutzt werden kann. Insgesamt ergibt sich durch die beschriebene Dekomposition des Base-Backends eine Entkopplung der API-Komponenten von der Komponente Base-Plug-in-System. Dadurch kann die Entwicklung von Plug-ins unabhängig von einer konkreten Backend-Instanz und die Installation und Aktualisierung von Plug-ins zur Laufzeit erfolgen, so dass durch diese Architektur die Anforderungen **FA52** (Entwicklung von Integrationskomponenten für Werkzeuge) und **FA53** (Verwaltung von Services zur Laufzeit) erfüllt werden können.

Die Architektur der Backends basiert auf der in [Bir05, Rat11] beschriebenen reinen Plug-in-Architektur. Der Einsatz dieser Architektur auf der Serverseite wird in der Literatur auch in einigen verwandten Ansätzen behandelt. Einer davon wird in [vDMC+10] diskutiert. Die Autoren behandeln Adinda, eine webbasierte IDE, die auf dem in [Rya07] vorgestellten Prototyp aufbaut und die Eclipse Java IDE auf der Serverseite einsetzt. Adinda erlaubt es, Java-Projekte, Pakete und Klassen über einen Browser anzulegen und die Klassen dann unter anderem zu editieren und zu kompilieren. Im Fokus steht dabei auch die Unterstützung der Kollaboration

durch die Services von Adinda. Insgesamt sind die Architektur und der Ansatz von Adinda vergleichbar mit den Backends des SSELab-Frameworks. Daher wäre auch eine Umsetzung der Konzepte von Adinda durch das Framework möglich. Eine entsprechende Service-Kategorie für webbasierte IDEs wurde bereits in Abschnitt 4.1.1 skizziert und wird in den weiteren Kapiteln dieser Arbeit wieder aufgegriffen.

Die Autoren von [WDPM06, WP07, Wol08] beschreiben das Plug-in-Framework Plux.NET für die .NET-Plattform. Das Framework basiert auf ähnlichen Konzepten wie die Eclipse Rich Client Platform (Eclipse RCP) [MLA10] und erlaubt die Komposition von Anwendungen aus Plug-in-Komponenten. Ursprünglich war Plux.NET auf Desktop-Anwendungen für einzelne Benutzer ausgelegt und ermöglichte deren dynamische Rekonfiguration. In [JWM10] und [JWLM13] beschreiben die Autoren eine Modifikation von Plux.NET, die Web-Anwendungen für mehrere Benutzer unterstützt und deren Funktionalität durch Plug-ins auf der Client- und auf der Serverseite erweitert werden kann. Im Gegensatz zu den Backends des SSELab-Frameworks ist Plux.NET jedoch nicht auf die Integration existierender Werkzeuge ausgelegt. Dieser Ansatz steht daher im Gegensatz zu dem Ansatz dieser Arbeit, bei dem existierende Werkzeuge integriert und dabei möglichst wenig angepasst werden sollen.

4.3.2 Funktionalität der Backends

In diesem Abschnitt wird die Kernfunktionalität aller Backend-Arten behandelt. Dies geschieht anhand der Komponente Base-Plug-in-System. Die Beschreibung ist jedoch allgemein gehalten und kann dementsprechend auf die anderen Backend-Arten übertragen werden. Abbildung 4.6 zeigt die Architektur der Komponente in Form eines Klassendiagramms. Anhand der gezeigten Klassen werden im Folgenden die wichtigsten Aspekte der Funktionalität der Backends beschrieben.

Erweiterbarkeit auf Grundlage von Plug-in-Systemen

Zur Umsetzung des Konzepts der Erweiterbarkeit zur Laufzeit werden in den Backends Plug-in-Systeme eingesetzt. In der Realisierung der Backends wird OSGi als konkrete Technologie verwendet. Die OSGi-Spezifikation [OSG09a] definiert ein Komponentenmodell für die Programmiersprache Java, das konform zu der in Abschnitt 2.2.1 angegebenen Definition aus [Szy02] ist. Die Komponenten werden in diesem Kontext als *Bundles* bezeichnet. Ein Bundle entspricht einem Java-Archiv, das neben den Java-Klassen eine Manifestdatei und optional weitere Artefakte enthält. In dem Manifest werden über OSGi-spezifische Einträge die Inhalte des Bundles beschrieben. Insbesondere werden hier die Abhängigkeiten zu anderen Bundles und die zur Verfügung gestellt Schnittstellen explizit angegeben. Auf Grundlage dieser Informationen können Bundles zur Laufzeit eines OSGi-Containers zu einem Gesamtsystem komponiert werden, so dass diese Technologie zur Umsetzung des Erweiterbarkeitskonzepts geeignet ist.

In den Backends werden OSGi-Container als Laufzeitumgebung eingesetzt. Daraus ergibt sich, dass jedes Service-Plug-in und auch die Komponenten der Backends des SSELab-Frameworks in Form eines OSGi-Bundles realisiert sind. In der Implementierung der Komponente Base-Plug-in-System werden dabei ausschließlich Klassen verwendet, die durch die OSGi-Spezifikation vorgegeben werden, so dass keine Abhängigkeit zu einem konkreten OSGi-Container entsteht

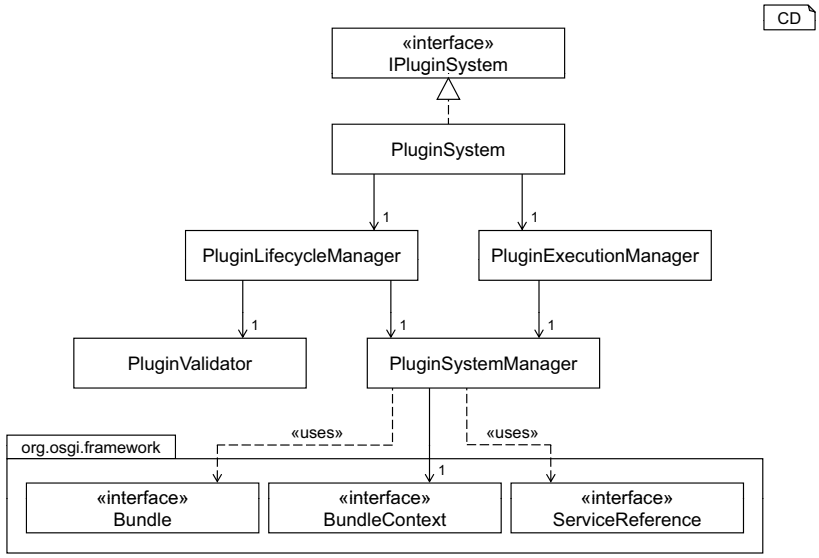


Abbildung 4.6: Grundlegende Architektur aller Backend-Arten

und dieser ohne Änderungen an der Implementierung ausgetauscht werden kann. Aktuell wird für das Deployment von Backend-Instanzen Apache Felix [Géd10] eingesetzt. Im Laufe dieser Arbeit wurde jedoch auch Eclipse Equinox [WHKL08] verwendet, so dass die Unabhängigkeit von einem konkreten Container gezeigt werden konnte.

Im Klassendiagramm in Abbildung 4.6 sind einige wesentliche Interfaces der OSGi-API dargestellt, die in der Implementierung der Komponente verwendet werden. Durch das `Bundle`-Interface kann unter anderem der Lebenszyklus eines Bundles kontrolliert werden. Das Interface `BundleContext` ermöglicht den Zugriff auf Funktionen der OSGi-Laufzeitumgebung. Durch `ServiceReference` werden Services im OSGi-Container repräsentiert. Im Kontext von OSGi wird dabei ein Objekt einer Bundle-Klasse, das in der Service-Registry des OSGi-Containers angemeldet ist, als Service bezeichnet.

Auf Grundlage dieser und einiger weiterer Klassen der OSGi-API bietet die Klasse `PluginSystemManager` eine Schnittstelle zur Interaktion mit dem OSGi-Container für die restlichen Klassen an. Diese Schnittstelle verwendet die Klassen des Domänenmodells und abstrahiert von den Details der OSGi-API, so dass eine Entkopplung der übrigen Klassen von dieser API erreicht werden kann. Die restliche Architektur der Backends ist dadurch unabhängig von einer konkreten Technologie, so dass die im Folgenden beschrieben Konzepte ebenfalls auf Grundlage anderer Plug-in-Systeme realisiert werden könnten.

Verwaltung von Plug-ins zur Laufzeit

Die Klasse `PluginLifecycleManager` verwaltet unter Verwendung der zuvor beschriebenen Schnittstelle den Lebenszyklus von Service-Plug-ins und ermöglicht unter anderem die Installation und die Deinstallation sowie die Aktualisierung von Plug-ins. Neben der Installation und Deinstallation ist auch eine Möglichkeit zur Aktualisierung erforderlich. Dies ergibt sich unter anderem aus den Eigenschaften von Base-Services. Diese werden im Kontext eines Projekts zur Instanziierung und Konfiguration einer web- oder serverbasierten Anwendung genutzt. Nach der Aktivierung eines Base-Services für ein Projekt muss dessen Konfiguration über die gesamte Lebensdauer des Projekts angepasst werden können, so dass die Deinstallation eines einmal genutzten Base-Service-Plug-ins nicht mehr zugelassen werden darf. Dementsprechend muss die Aktualisierung von Plug-ins möglich sein, damit in einer laufenden Framework-Instanz sowohl neue Funktionalität verfügbar gemacht als auch Fehler in der Realisierung von Plug-ins korrigiert werden können.

Im Folgenden werden die Mechanismen zur Installation, Aktualisierung und Deinstallation von Plug-ins genauer erläutert. Zunächst wird auf die Installation und Aktualisierung eingegangen. Abbildung 4.7 zeigt dazu in einem Aktivitätsdiagramm die Schritte, die dabei ausgeführt werden.

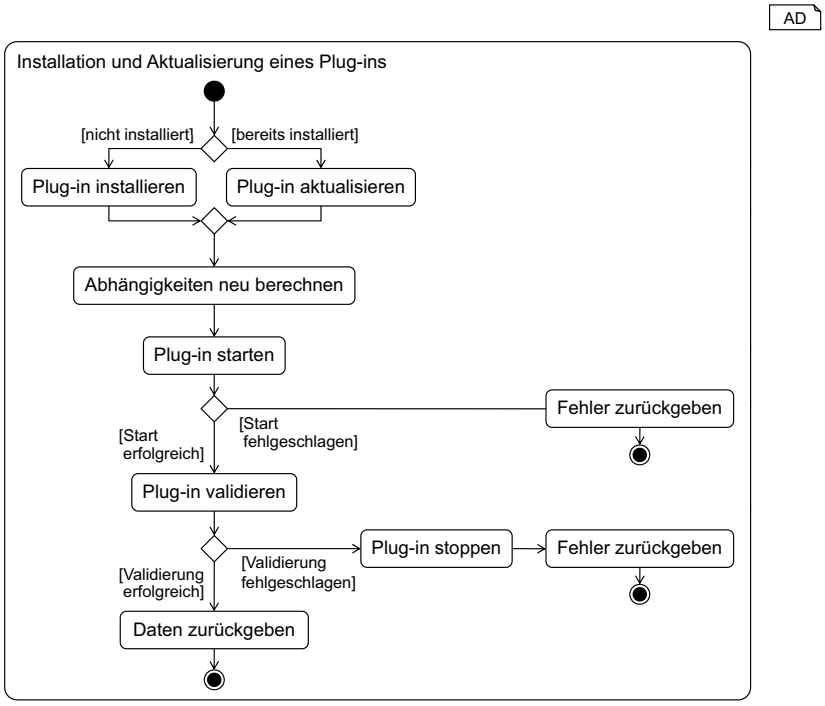


Abbildung 4.7: Installation und Aktualisierung eines Plug-ins in einer Backend-Instanz

Die Backends ermöglichen sowohl die Installation und Aktualisierung einzelner als auch mehrerer Plug-ins auf einmal. Anhand Abbildung 4.7 wird hier die Verarbeitung eines einzelnen Plug-ins beschrieben.

Nachdem ein Plug-in von dem Frontend zu einer Backend-Instanz übertragen worden ist, wird es verarbeitet. Dabei wird zwischen der Installation eines neuen Plug-ins und der Aktualisierung eines bereits installierten Plug-ins unterschieden. Ob eine Installation oder eine Aktualisierung notwendig ist, ist abhängig davon, ob das Plug-in bereits in der Backend-Instanz enthalten ist oder nicht. Um dies zu ermitteln, kann entweder der Name des Plug-ins selbst oder aber dessen Inhalt analysiert werden. In der Realisierung der Backends wird aktuell die Namensgleicheit der Plug-ins überprüft. Eine Konsequenz davon ist, dass bei der Aktualisierung eines Plug-ins darauf geachtet werden muss, dass der Dateiname beibehalten wird, der auch bei der ursprünglichen Installation verwendet wurde, da sonst eine mehrfache Installation erfolgen kann. Bei einer Analyse der Inhalte eines Plug-ins wäre im Gegensatz dazu mehr Flexibilität bei der Namensgebung möglich.

Nach der Aktualisierung eines Service-Plug-ins in einem Backend muss darauf geachtet werden, dass auch die neueste Version genutzt wird, so dass die neuen Funktionen eines aktualisierten Plug-ins unmittelbar verwendet werden können. Die OSGi-Spezifikation schreibt beispielsweise vor, dass auch nach der Aktualisierung eines Plug-ins die zuvor installierte Version bzw. die zuvor exportierten Schnittstellen noch innerhalb des Containers verfügbar sein müssen [OSG09a]. Daher besitzen OSGi-Container entsprechende Caching-Mechanismen, über die alle Versionen vorgehalten werden können. Da jedoch diese Verhalten im Kontext der Backends nicht zielführend ist, werden nach einer erfolgreichen Aktualisierung die Abhängigkeiten der enthaltenen Plug-ins neu berechnet. Dadurch wird sichergestellt, dass ein konsistenter Zustand in Bezug auf die zuletzt aktualisierten Plug-ins einer Backend-Instanz hergestellt wird. Bei einer Realisierung der Backends auf Grundlage anderer Plug-in-Systeme ohne entsprechende Caching-Mechanismen könnte dieser Schritt entfallen.

Anschließend wird das Plug-in gestartet. Bei Verwendung eines OSGi-Containers werden erst zu diesem Zeitpunkt die Informationen in der Manifestdatei eines neu installierten Plug-ins analysiert. Nach der Analyse steht unter anderem fest, ob die Abhängigkeiten des Plug-ins innerhalb des laufenden Containers erfüllt werden können. Ist das der Fall, wird es im OSGi-Container aktiviert (d. h. es besitzt den Zustand `ACTIVE` im Container [OSG09a]) und kann über die im Manifest definierten Schnittstellen aufgerufen werden. Ist dies nicht der Fall, wird vom Container eine Fehler erzeugt, der zum Abbruch der Installation oder Aktualisierung des Plug-ins im Backend führt.

Im nachfolgenden Schritt erfolgt die Validierung des Plug-ins mit Hilfe der in Abbildung 4.6 gezeigten Klasse `PluginValidator`. Dies geschieht auf Grundlage der Plug-in-API des jeweiligen Backends. Bei der Validierung können die wesentlichen Daten auf Vollständigkeit und Konsistenz mit den Vorgaben der jeweiligen Plug-in-API überprüft werden. Außerdem ist auch eine Analyse des Plug-ins auf fehlerhaften oder bösartigen Code zu diesem Zeitpunkt möglich. Der Umfang der Validierung ist dabei auch von dem Verhältnis der Plug-in-Entwickler zu den Betreibern einer Framework-Instanz abhängig. Im Rahmen dieser Arbeit existierte ein Vertrauensverhältnis zwischen Entwicklern und Betreibern, so dass in der Realisierung keine speziellen Code-Analysen durchgeführt werden. Unabhängig vom Umfang führen Fehler bei

der Validierung zu einem Stopp des Plug-ins. Alle gefundenen Fehler werden dann in einer Fehlermeldung zusammengefasst und für die weitere Verarbeitung im Frontend zurückgegeben. Falls die Validierung erfolgreich ist, werden grundlegende Daten aus dem Plug-in ausgelesen und zurückgegeben.

Die Deinstallation eines Plug-ins kann genau wie die Installation zur Laufzeit einer Backend-Instanz erfolgen. Die Schritte, die dabei durchgeführt werden, sind in Abbildung 4.8 in einem Aktivitätsdiagramm dargestellt.

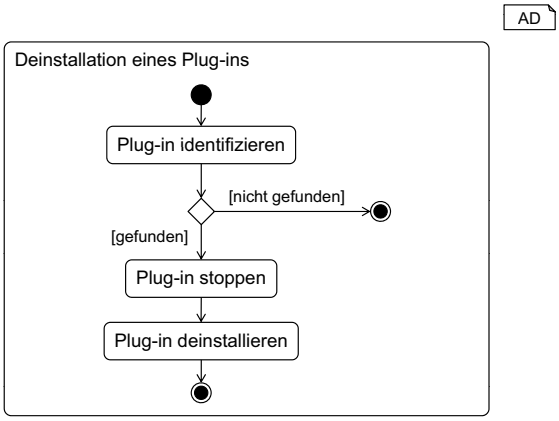


Abbildung 4.8: Deinstallation von Plug-ins in einer Backend-Instanz

Zur Deinstallation eines Plug-ins wird die im Frontend gespeicherte ID an das Backend übergeben. Der Aufbau der ID ist dabei abhängig von der jeweiligen Backend-Art. Im Fall des Base-Backends besteht die ID beispielsweise aus dem symbolischen Namen des Plug-ins, der bei der Installation oder Aktualisierung ausgelesen wird. Zur Deinstallation wird zunächst das passende Plug-in identifiziert. Falls ein entsprechendes Plug-in installiert und aktiviert ist, wird es zunächst gestoppt und anschließend deinstalliert. Andernfalls wird keine weitere Aktion ausgeführt.

Insgesamt ermöglicht die Klasse `PluginLifecycleManager` die Verwaltung von Service-Plug-in zur Laufzeit einer Backend-Instanz. Wichtig dabei ist, dass alle Operationen die den Lebenszyklus von Plug-ins beeinflussen so realisiert werden, dass bei konsekutiven Aufrufen mit identischen Parametern der Zustand des Plug-in-Systems gleich bleibt und auch keine Fehler produziert werden. Dieses Design ist essentiell für die Unterstützung und Konsistenzsicherung mehrerer Instanzen einer Backend-Art. In Abschnitt 4.4.2 werden die auf diesen Eigenschaften aufbauenden Mechanismen genauer beschrieben.

Ausführung von Plug-ins

Die Klasse `PluginExecutionManager` ermöglicht die Ausführung von Plug-ins auf Grundlage der Plug-in-API der jeweiligen Backend-Art. Gemäß den unterschiedlichen Eigenschaften und Anforderungen der Service-Kategorien ist die Funktionalität dieser Klasse grundlegend

verschieden für jede Backend-Art. Alle Backend-spezifischen Plug-in-APIs bauen jedoch auf der Komponente Common-Plug-in-API auf und basieren daher auf einheitlichen Konzepten, so dass es auch Gemeinsamkeiten gibt. Diese Gemeinsamkeiten werden im Folgenden beschrieben. In Abschnitt 5.2 werden dann weitere Backend-spezifische Konzepte anhand der jeweiligen Plug-in-API diskutiert.

Ein Plug-in-System ist ein dynamische Umgebung, in der zu einem beliebigen Zeitpunkt neue Plug-ins installiert, deinstalliert, gestartet oder gestoppt werden können. Dieses dynamische Verhalten erfordert Mechanismen, die eine robuste Interaktion mit dem Plug-in-System und dessen Plug-ins ermöglichen. Die Klasse `PluginExecutionManager` baut dementsprechend auf mehreren Prinzipien auf. Ein Prinzip ist, dass vor jedem Aufruf einer Methode eines bestimmten Service-Plug-ins dessen Verfügbarkeit überprüft werden muss. Dazu wird – wie auch bei der zuvor beschriebenen Deinstallation – anhand der ID das Plug-in identifiziert und dessen Zustand geprüft. Nur wenn es aktiviert ist, wird ein Aufruf zugelassen. Andernfalls wird durch einen entsprechenden Fehler der Zugriff auf ein nicht verfügbares Plug-in angezeigt.

Zwischen der Überprüfung, ob ein Plug-in überhaupt verfügbar ist, und dem Aufruf des Plug-ins kann sich der Zustand eines Plug-in-Systems jedoch ebenfalls ändern, so dass der Aufruf zu einem Fehler führen kann. Darüber hinaus kann ein Plug-in auch fehlerhaften oder bösartigen Code enthalten, so dass es zu unerwartetem Verhalten kommen kann. Auch wenn entsprechende Validierungsmechanismen genutzt werden, muss ein Backend auf diese und andere Fehlfunktionen robust reagieren können, damit es nicht zum Absturz gebracht werden kann. Daher müssen bei dem Aufruf eines Plug-ins auch alle unerwartete Fehlerfälle behandelt werden. Dies geschieht in der Realisierung der `PluginExecutionManager`-Klasse konkret durch das Abfangen aller Exceptions. Dies ist zwar im Allgemeinen nicht ratsam, da sonst alle Fehlerfälle auf die gleiche Weise behandelt werden, in den Backends jedoch notwendig, um alle unerwarteten Fehler abzufangen und dadurch kontrolliert auf Fehlverhalten von Plug-ins reagieren zu können. Die unerwarteten Exceptions werden dann über Exception Chaining [Blo08] weiterverarbeitet.

Zusätzlich zu den beschriebenen Prinzipien für eine robuste Interaktion mit dem Plug-in-System und den Plug-ins werden in den `PluginExecutionManager`-Klassen der jeweiligen Backend-Arten die Parameter und die Rückgabewerte für Methodenaufrufe der Plug-in-API aufbereitet. Dazu gehört beispielsweise das Erstellen temporärer Verzeichnisse oder das Entpacken von Eingabedateien. Auf diese technischen Details wird hier jedoch nicht weiter eingegangen.

Einheitlicher Zugriff auf Backend-Instanzen

Alle Instanzen einer Backend-Art basieren auf dem Konzept einer identischen Schnittstelle, damit sie vom Frontend in einer einheitlichen Art und Weise verwaltet und aufgerufen werden können. Dadurch ist es möglich, Instanzen einer Backend-Art zur Laufzeit des Frontends in Betrieb zu nehmen, so dass eine skalierbare Architektur entsteht und ein flexibles Deployment möglich wird (vgl. Anforderung **NFA11**). Zusätzlich wird durch das Frontend ein konsistenter Zustand aller Backend-Instanzen einer Art sichergestellt. Auf die dabei eingesetzten Synchronisationsmechanismen wird in Abschnitt 4.4.2 genauer eingegangen. Im Folgenden werden einige Eigenschaften der einheitlichen Backend-Schnittstellen beschrieben.

In jedem Backend bilden das Interface `IPluginSystem` und die Klasse `PluginSystem` eine Fassade [GHJV95] für den Zugriff auf die Backend-Instanz. Über diese Fassade wird die

Funktionalität eines Backends für das Frontend in einer einheitlichen Schnittstelle angeboten. Über die Schnittstelle kann ein Backend und dessen Plug-ins verwaltet sowie die Plug-ins ausgeführt werden. Dabei realisiert die Klasse `PluginSystem` keine der angebotenen Funktionen selbst, sondern delegiert an die zuvor beschriebenen Klassen `PluginLifecycleManager` und `PluginExecutionManager`.

In der im Rahmen der Arbeit entwickelten Realisierung des Frameworks erfolgt die Kommunikation zwischen der Frontend- und einer Backend-Instanz über Web-Service [FF03]. Dazu wird das Interface `IPluginSystem` mit Hilfe eines Web-Service-Stacks und eines HTTP-Servers beim Deployment im OSGi-Container als Web-Service publiziert. Alle Methoden und Parameter- und Rückgabetypen werden dementsprechend in einer WSDL-Datei [CCMW01] deklariert. Damit jedoch die publizierte WSDL-Schnittstelle einer Backend-Instanz nicht allgemein zugänglich ist, wird eine TLS-verschlüsselte Verbindung mit beidseitigem Handshake unter Verwendung von Client-Zertifikaten zwischen Frontends und Backends verwendet, so dass Anforderung **NFA12** (Sichere Kommunikation innerhalb des Frameworks) erfüllt wird. Der Einsatz anderer geeigneter Technologien ist hier jedoch ebenfalls möglich.

4.4 Verwaltungsmechanismen im Frontend

In einem Framework für Projektportale und CDEs werden Mechanismen für die Verwaltung von Benutzern, Projekten und Services benötigt. Das Frontend enthält solche Mechanismen und ist die zentrale webbasierte Administrationskomponente, die die Interaktion aller anderen Komponenten steuert. Es verwaltet Benutzer, Projekte, die Backend-Instanzen und deren Services sowie die Clients. Darüber hinaus ermöglicht es den Zugriff auf alle Funktionen einer SSELab-Instanz, die in den Anforderungen in Kapitel 3 beschrieben sind. Im Folgenden sind die Anforderungen noch einmal aufgelistet, die die Architektur des Frontends maßgeblich beeinflusst haben.

- **FA46** *Application-Framework für Projektportale und CDEs*
- **FA47** *Entwicklung als webbasiertes Framework*
- **FA53** *Verwaltung von Services zur Laufzeit*
- **FA54** *Unabhängige Entwicklung von Framework-Clients*
- **FA55** *Bereitstellung von Benutzerinformationen zur Authentifizierung*
- **NFA11** *Flexibles Deployment*
- **NFA12** *Sichere Kommunikation innerhalb des Frameworks*

Diese Anforderungen beschreiben primär die Framework-Sicht auf das Frontend. Darüber hinaus beeinflussen die Anforderungen aus den Abschnitten 3.3.1 und 3.3.3 maßgeblich die Verwaltungsmechanismen des Frontends. In diesem Abschnitt wird zunächst die Architektur des Frontends diskutiert und im Anschluss werden die wichtigsten Verwaltungsmechanismen auf Grundlage des Domänenmodells des SSELab-Frameworks genauer betrachtet.

4.4.1 Architektur des Frontends

In Abschnitt 4.2 wurde die Framework-Architektur und die Schnittstellen des Frontends zur Kommunikation mit den Backends und den Clients auf hoher Abstraktionsebene eingeführt. In diesem Abschnitt wird darauf aufbauend und auf Grundlage der zuvor aufgelisteten Architekturtreiber der interne Aufbau des Frontends mit den wichtigsten Subkomponenten und deren Abhängigkeiten untereinander diskutiert. Abbildung 4.9 zeigt dazu ein Komponentendiagramm mit den relevanten Subkomponenten des Frontends. Die beiden Komponenten *SSELab-Util* und *SSELab-Domain* werden im Frontend genau wie in den Backends unverändert genutzt und sind hier der Vollständigkeit halber aufgeführt. Die weiteren Subkomponenten werden im Folgenden behandelt.

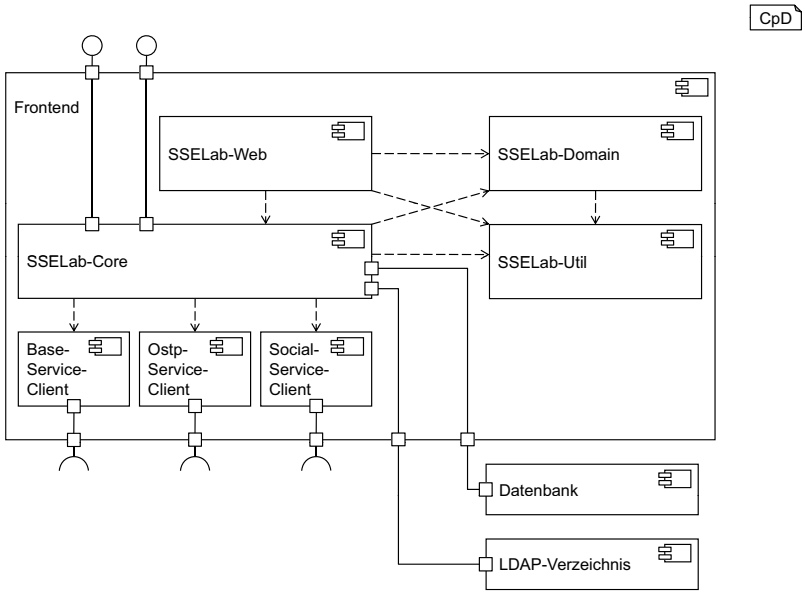


Abbildung 4.9: Komponentendiagramm der Frontend-Architektur

- Im vorangegangenen Abschnitt 4.3.2 wurde der einheitliche Zugriff auf die Instanzen einer Backend-Art über deren Schnittstellen beschrieben. Die Kommunikation zwischen dem Frontend und den Backend-Instanzen erfolgt unter Verwendung der drei Subkomponenten *Base-Service-Client*, *Ostp-Service-Client* und *Social-Service-Client*. Durch diese Komponenten wird der Zugriff auf die jeweiligen Backend-Instanzen gekapselt und in wiederverwendbarer Form angeboten.
- Die Komponente *SSELab-Core* ist die Hauptkomponente des Frontends, in der die Geschäftslogik auf Grundlage des Domänenmodells enthalten ist. Diese Komponente steuert

alle Backend-Instanzen über die Service-Client-Komponenten, verwaltet die persistenten Daten und ermöglicht den Zugriff auf das System über mehrere Fassaden [GHJV95]. Die in den folgenden Abschnitten beschriebenen Verwaltungsmechanismen sind primärer Bestandteil dieser Komponente.

- Die Daten einer SSELab-Instanz werden in einer *Datenbank* sowie in einem *LDAP-Verzeichnis* verwaltet. In der Datenbank werden alle Daten gespeichert, die durch das Domänenmodell definiert werden. Dazu gehören dementsprechend Informationen über die Benutzer, Services und Projekte der Instanz. Einige der Daten werden zusätzlich auch über einen LDAP-Server angeboten. Dazu gehören unter anderem Benutzerinformationen zur Authentifizierung, so dass Anforderung **FA55** erfüllt wird.

- Die Komponente *SSELab-Web* enthält die zentrale Web-Anwendung, über die alle Funktionen per Browser genutzt werden können. SSELab-Web verwendet die in SSELab-Core enthaltene Geschäftslogik und ermöglicht so gemäß den Anforderungen **FA1** (Nutzung über zentrale Web-Oberfläche) und **FA35** (Zentrale Administration des Systems) den Zugriff von Gästen, Benutzern und Administratoren über Web-Browser. Auf Details dieser Komponente wird im Folgenden nicht genauer eingegangen.

Insgesamt basiert die Architektur des Frontends auf einer klassischen Schichtenarchitektur [BMR+96], wie sie im Bereich von Enterprise-Anwendungen eingesetzt wird. Dabei sind die Framework-Anteile innerhalb des Frontends vergleichsweise schwach ausgeprägt. Dies ergibt sich aus den Anforderungen, durch die einerseits ein erweiterbares Framework und andererseits ein verteiltes System, das bereits ohne konkrete Services und Clients lauffähig ist, gefordert werden. Die Realisierung der Konzepte des Frontends erfolgt daher immer in einer Mischung aus Framework- und Anwendungsentwicklung.

4.4.2 Service- und Backend-Verwaltung

Zur Laufzeit einer Frontend-Instanz können sowohl neue Instanzen aller verfügbaren Backend-Arten als auch neue Services hinzugefügt werden (vgl. Anforderung **FA53**). Die dafür notwendigen Mechanismen innerhalb des Frontends basieren sowohl auf den in Abschnitt 4.3.2 beschriebenen Konzepten zur Verwaltung und Ausführung von Plug-ins als auch auf den einheitlichen Schnittstellen aller Instanzen einer Backend-Art. In diesem Abschnitt werden zunächst die Konzepte zur Verwaltung von Services und den zugehörigen Plug-ins innerhalb des Frontends genauer beschrieben. Anschließend werden die Mechanismen zur Verwaltung von Backend-Instanzen diskutiert. Die Erläuterungen erfolgen dabei auf Grundlage des in Abbildung 4.10 gezeigten Deployment-Szenarios mit einer Frontend-Instanz auf einem Server mit dem Namen `lab10` und drei Base-Backend-Instanzen auf den Servern `lab2`, `lab9` und `lab14`. Die Beschreibung ist auch hier wieder allgemein gehalten und kann analog auf weitere Deployment-Szenarien mit Instanzen anderer Backend-Arten übertragen werden.

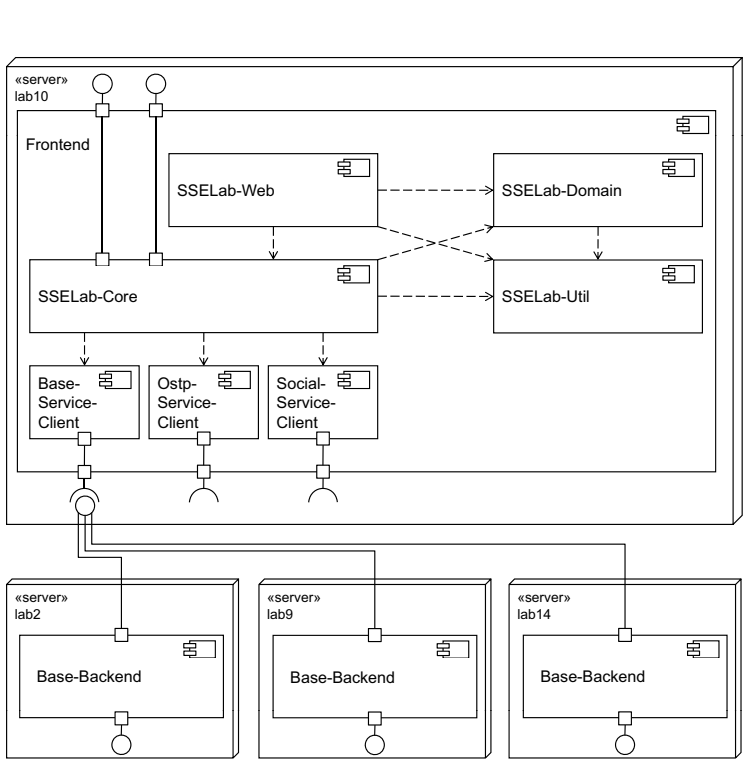


Abbildung 4.10: Verteilungsdiagramm mit einer Frontend- und drei Base-Backend-Instanzen

Repräsentation und Verwaltung von Services und Plug-ins

Wie in Abschnitt 4.1.1 diskutiert, werden Werkzeuge und Anwendungen, die in das Framework integriert worden sind, als Services bezeichnet und durch entsprechende Klassen des Domänenmodells repräsentiert. Die Services der verschiedenen Kategorien werden dabei anhand von Subklassen unterschieden. Abbildung 4.11 verfeinert das Klassendiagramm aus Abschnitt 4.1.1 und zeigt neben den Beziehungen dieser Klassen zusätzlich einige exemplarische Eigenschaften.

Die Klasse `Service` beschreibt die gemeinsamen Aspekte von Services aller Kategorien. Jeder Service verfügt unter anderem über einen symbolischen Namen zur Identifikation innerhalb des Frameworks, einen Namen und eine Beschreibung. Zusätzlich zu diesen allgemeinen Informationen werden in den Subklassen von `Service` jeweils Eigenschaften definiert, die spezifisch für jede Service-Kategorie sind. In Abbildung 4.11 sind einige Beispiele dafür angegeben. Ostp-Services besitzen beispielsweise eine Versionsnummer, die zusätzlich zu dem symbolischen Namen zur Identifikation verwendet wird.

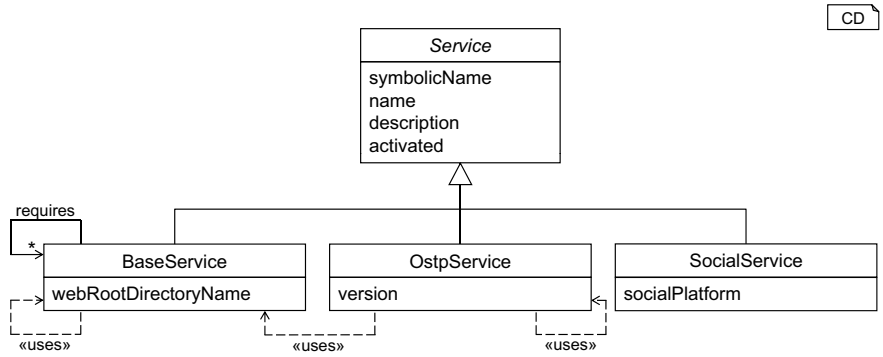


Abbildung 4.11: Eigenschaften der Service-Kategorien

Die Informationen über einen Service stammen teilweise aus den zugehörigen Service-Plug-ins. Bei der Installation oder Aktualisierung eines Plug-ins werden diese auf Grundlage der Plug-in-APIs ausgelesen (vgl. Abschnitt 4.3.2). Andere Informationen werden durch das Frontend bei der Verwaltung von Services verwendet. Als Beispiel ist in Abbildung 4.11 das Attribut `activated` gezeigt, dass beschriebt welchen Aktivierungszustand ein Service besitzt. Dieser Zustand ändert sich, wenn beispielsweise ein Administrator einen Service zur allgemeinen Verwendung freigibt bzw. sperrt.

Insgesamt können mit Hilfe der Klassen alle relevanten Informationen über einen installierten Service und dessen aktuellen Zustand repräsentiert werden. Diese Informationen werden vom Frontend unter anderem für die Zuordnung von Services zu Projekten genutzt. Neben den Informationen über die Services verwaltet das Frontend auch die eigentlichen Service-Plug-ins, die in den Backend-Instanzen installiert sind. Die Klassen des Domänenmodells, die dazu genutzt werden, sind in Abbildung 4.12 gezeigt.

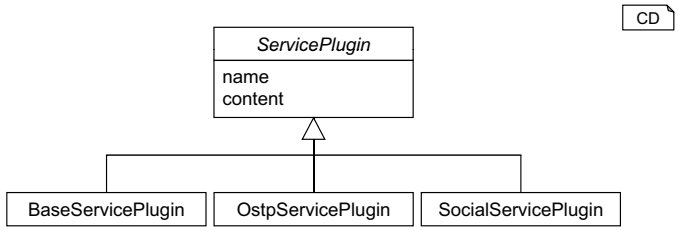


Abbildung 4.12: Klassen zur Verwaltung von Service-Plug-ins

Bei der Installation oder Aktualisierung von Service-Plug-ins werden die zugehörigen Archive entweder über einen der Web-Services des Frontends oder über dessen Web-Oberfläche durch einen Administrator hochgeladen. Im Frontend werden die Plug-ins nach der erfolgreichen Durchführung der Installation oder Aktualisierung gespeichert. Dies ist notwendig, um einer-

seits die Konsistenzsicherung vorhandener Backend-Instanzen und andererseits die Registrierung und Initialisierung neuer Backend-Instanzen unterstützen zu können. Beide Aspekte werden im Folgenden genauer behandelt.

Identifizierung von Backend-Instanzen durch Server-Mappings

Die in dem Deployment-Szenario in Abbildung 4.10 auf Seite 63 dargestellte Base-Service-Client-Komponente kapselt den Zugriff auf die einheitliche Schnittstelle der drei Base-Backend-Instanzen `lab2`, `lab9` und `lab14`. Damit das Frontend mit einer der drei Instanzen kommunizieren kann, muss lediglich beim Aufruf der Komponente die entsprechende Server-URL angegeben werden. Die Server-URLs und alle weiteren Informationen über die registrierten Backend-Instanzen werden innerhalb des Frontends durch sogenannte Server-Mappings verwaltet. Die Klassen des Domänenmodells zur Repräsentation von Server-Mappings sind in Abbildung 4.13 dargestellt.

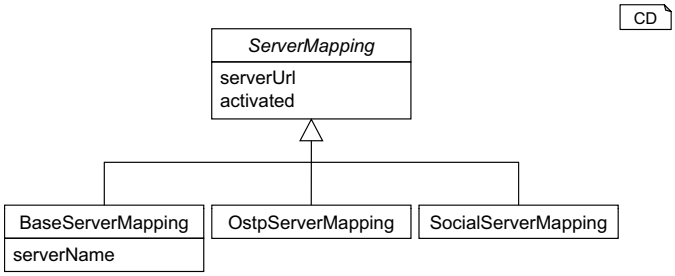


Abbildung 4.13: Domänenklassen zur Repräsentation und Identifizierung von Backend-Instanzen

Für jede Backend-Art existiert ein eigener Server-Mapping-Typ. Bei der Registrierung einer Backend-Instanz durch einen Administrator wird vom Frontend ein Server-Mapping des entsprechenden Typs verwendet. Jedes Server-Mapping enthält dabei die URL, unter der die zugehörige Backend-Instanz erreichbar ist. In einem Base-Server-Mapping wird zusätzlich noch der Name des Servers gespeichert, der unter anderem für den Aufbau eindeutiger URLs für den Zugriff auf Base-Service-Instanzen verwendet wird. Für die Nutzung von Ostp- und Social-Backend-Instanzen werden dagegen keine zusätzlichen Informationen benötigt. Nach der Registrierung einer Backend-Instanz kann deren Funktionalität im Probebetrieb durch einen Administrator getestet werden. Nach dem erfolgreichen Test kann sie aktiviert und damit für die Konfiguration und Ausführung von Services durch das Frontend genutzt werden.

Durch die charakteristischen Eigenschaften der Services jeder Backend-Art (vgl. Abschnitt 4.1.1) unterscheiden sich die Nutzung der Backends und dementsprechend auch die Verwaltungsmechanismen der Server-Mappings im Frontend voneinander. Die daraus resultierenden Beziehungen der Server-Mappings zu anderen Klassen des Domänenmodells werden im Folgenden kurz beschrieben.

Base-Server-Mappings: Durch Base-Services werden server- bzw. webbasierte Anwendungen in das Framework integriert, die ausschließlich im Kontext von Projekten genutzt werden können. Damit die Konfiguration und Handhabung aller Base-Services eines Projekts vereinfacht wird, ist es sinnvoll, dass diese auf dem gleichen Backend-Server konfiguriert und ausgeführt werden. Dementsprechend gibt es eine feste Zuordnung eines jeden Projekts zu einer konkreten Base-Backend-Instanz über ein Base-Server-Mapping. Aus dieser festen Zuordnung folgt, dass eine einmal genutzte Backend-Instanz und demzufolge das zugehörige Base-Server-Mapping nicht mehr gelöscht werden darf, da sonst das Frontend nicht mehr zur Konfiguration der Services auf die Backend-Instanz zugreifen könnte.

Die Auswahl einer Backend-Instanz für ein Projekt und damit die Zuordnung eines Server-Mappings erfolgt bei dessen Aktivierung. Dabei sollte die Instanz gewählt werden, die am geringsten ausgelastet ist, um eine möglichst effektive Lastverteilung und damit eine optimale Skalierung zu erreichen. In der Realisierung des Frontends wird momentan die aktuelle Anzahl von Projekten der Backend-Instanzen als Grundlage für die Zuordnung gewählt. Komplexere Verfahren zur Ermittlung der Auslastung könnten jedoch auch genutzt werden.

Ostp-Server-Mappings: Ostp-Services sind zustandslose Transformationswerkzeuge, die keine Daten zwischen konsekutiven Aufrufen speichern. Daraus folgt, dass es nicht notwendig ist, einem Projekt oder einem Benutzer eine bestimmte Ostp-Backend-Instanz dauerhaft zuzuweisen. Prinzipiell könnte also für jeden Aufruf eine geeignete Backend-Instanz ausgewählt und genutzt werden. Allerdings besitzen die Ostp-Backends die Möglichkeit, die übertragenen Dateien bei einem Aufruf eines Services in einem serverseitigen Cache zu speichern, so dass bei einem nachfolgenden Aufruf des selben Benutzers diese Dateien nicht neu übertragen werden müssen, sondern aus dem Cache geladen werden können. Daher wird einem Benutzer beim ersten Aufruf eines Ostp-Services eine konkrete Backend-Instanz zugewiesen, die bei nachfolgenden Aufrufen auch genutzt wird. Die Auswahl erfolgt auch hier in Abhängigkeit von der Auslastung der Ostp-Backend-Instanzen. Vor jedem Aufruf wird zusätzlich die Verfügbarkeit der zugewiesenen Backend-Instanz geprüft und bei Bedarf eine neue Instanz dynamisch ausgewählt.

Social-Server-Mappings: Social-Services sind genau wie Ostp-Services zustandslos. Mit ihrer Hilfe können Informationen eines Benutzers von einem sozialen Netzwerk oder anderen Projektportalen abgerufen und in dessen Profil gespeichert werden. Dementsprechend müsste auch hier keine feste Zuweisung eines Social-Server-Mappings zu einem Benutzer erfolgen. Aus Gründen der Konsistenz werden jedoch Social-Server-Mappings analog zu Ostp-Server-Mappings behandelt und jedem Benutzer eine spezifische Backend-Instanz über ein Server-Mapping zugewiesen.

Aufbauend auf der Verwendung einheitlicher Schnittstellen ermöglicht die hier beschriebene Parametrisierung der Service-Client-Komponenten mit Server-Mappings einen unabhängigen Betrieb von Frontend- und Backend-Instanzen. Dadurch ist das Deployment eines Frontends vom Deployment der Backends entkoppelt und neue Backend-Instanzen können zur Laufzeit des Frontends in Betrieb genommen werden, so dass die gewünschte Skalierbarkeit erreicht wird.

Konsistenzsicherung von Backend-Instanzen

Die Verwaltung der Backend-Instanzen zur Laufzeit des Frontends sowie das dynamische Verhalten der Plug-in-Systeme (vgl. Abschnitt 4.3.2) kann zu inkonsistenten Zuständen in Bezug auf die installierten Service-Plug-ins der Backends führen. Beispielsweise könnte bei der Installation eines Service-Plug-ins in einer Backend-Instanz ein Fehler auftreten, der in einer anderen nicht auftritt, so dass diese beiden Backend-Instanzen einen inkonsistenten Zustand besitzen. Dies erfordert Mechanismen innerhalb des Frontends, die einerseits eine robuste Interaktion mit allen Backend-Instanzen sicherstellen und andererseits die Wiederherstellung eines konsistenten Zustands ermöglichen. Das Ziel dabei ist, dass die registrierten Backend-Instanzen einer Art die gleichen Services anbieten und das Frontend eine beliebige Instanz für die Konfiguration oder Ausführung von Services auswählen kann. Die Konzepte und Mechanismen zur Konsistenzsicherung der Backend-Instanzen werden im Folgenden beschrieben.

Das Frontend muss bestimmte Operationen, wie beispielsweise die Installation eines neuen Services, auf allen registrierten Backend-Instanzen ausführen. Es verwendet dazu alle gespeicherten Server-Mappings und ruft die Backend-Instanzen der Reihe nach auf. In Abbildung 4.14 ist in einem Sequenzdiagramm die Installation eines neuen Services auf den drei Backend-Instanzen `lab2`, `lab9` und `lab14` zur Verdeutlichung dieses Vorgehens dargestellt.

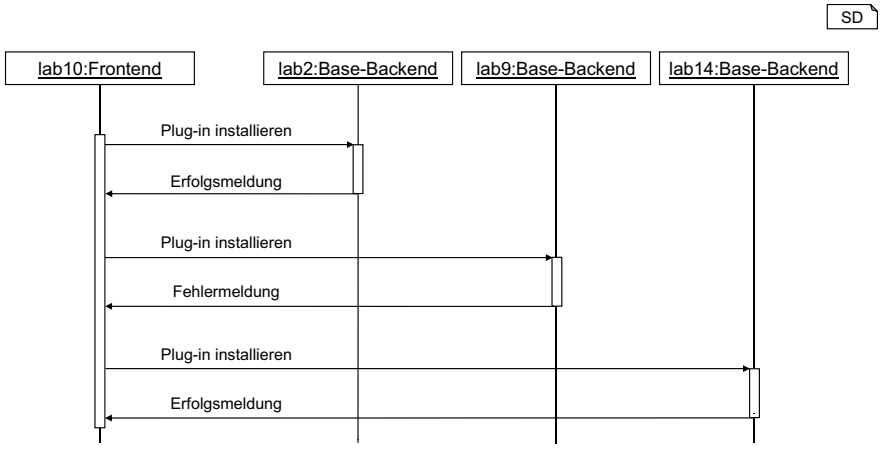


Abbildung 4.14: Sequenzdiagramm zur Installation eines neuen Services auf drei Backend-Instanzen

In diesem Beispiel ruft das Frontend zunächst die Backend-Instanz `lab2` auf und übergibt das zu installierende Service-Plug-in. Die Installation ist in diesem Fall erfolgreich, so dass danach `lab9` aufgerufen wird. Bei der Installation des Service-Plug-ins auf dieser Backend-Instanz tritt ein Fehler auf, so dass eine entsprechende Fehlermeldung von der Backend-Instanz zum Frontend zurückgegeben wird. Trotz dieses Fehlers wird die Installation des Services auch auf der dritten Backend-Instanz `lab14` durchgeführt. Diese Strategie zum Umgang mit Fehlern bei

der Interaktion mit den Backend-Instanzen ergibt sich aus der folgenden Kategorisierung der Fehlerfälle, die bei der Interaktion mit den Backend-Instanzen auftreten können.

- Wenn von einem konsistenten Zustand aller Backend-Instanzen ausgegangen wird, treten Fehler, die durch einzelne Service-Plug-ins ausgelöst werden, in allen Backend-Instanzen gleichermaßen auf.
- Andere Fehler treten erfahrungsgemäß nur isoliert, d. h. in einzelnen Backend-Instanzen auf. Beispielsweise durch das Auftreten temporärer Verbindungsprobleme oder Ressourcenknappheit.

Daraus ergibt sich, dass es bei einem Fehler bei der Interaktion mit einer einzelnen Backend-Instanz sinnvoll ist, die Operation dennoch auf allen anderen Backend-Instanzen durchzuführen, damit möglichst viele Backend-Instanzen in einem korrekten und konsistenten Zustand und damit nutzbar sind. Durch die zuvor beschriebene Eigenschaft, dass konsekutive Aufrufe mit gleichen Parametern keine Zustandsänderung der Backend-Instanzen zur Folge haben, kann dann nach der Behebung des Fehlers durch eine erneute Durchführung der Operation ein konsistenter Zustand aller Backend-Instanzen hergestellt werden.

Zusätzlich zu dieser Strategie zum Umgang mit Fehlern bei der Interaktion mit einzelnen Backend-Instanzen wird vom Frontend kontinuierlich die Verfügbarkeit aller registrierten Instanzen geprüft (vgl. Anforderung **FA44**). Dazu werden regelmäßig Verbindungen zu den Instanzen aufgebaut und deren Zustand geprüft. Falls dieser Test nicht erfolgreich durchgeführt werden kann, werden die Administratoren vom Frontend per E-Mail benachrichtigt. Die Administratoren können diesen Test darüber hinaus auch manuell über die Administrationsoberfläche des Frontends ausführen.

Da das Frontend alle installierten Service-Plug-ins verwaltet, kann es jederzeit einen konsistenten Zustand aller Backend-Instanzen durch eine Synchronisation der in den Backend-Instanzen installierten Plug-ins mit den gespeicherten Plug-ins herstellen. Eine Synchronisation ist dabei in einem der beiden folgenden Szenarien notwendig.

- Bei der Registrierung und Initialisierung einer neuen Backend-Instanz durch einen Administrator müssen alle vom Frontend gespeicherten Plug-ins in der Instanz installiert werden.
- Falls Fehler bei der Interaktion mit oder der Konfiguration von einer Backend-Instanz aufgetreten sind, können diese behoben werden und schließlich durch eine Synchronisation alle Instanzen in einen konsistenten Zustand gebracht werden.

Ein Beispiel für das erste Szenario ist im Sequenzdiagramm in Abbildung 4.15 gezeigt. In diesem Beispiel ist die erfolgreiche Synchronisation der Base-Backend-Instanz `lab14` nach deren Registrierung durch einen Administrator dargestellt.

Bei der Registrierung einer neuen Base-Backend-Instanz gibt ein Administrator über die Web-Oberfläche des Frontends die Server-URL an. Dann wird zunächst geprüft, ob eine Backend-Instanz unter dieser URL erreichbar ist. Wenn der Test erfolgreich war, werden alle gespeicherten Service-Plug-ins geladen und vom Frontend zu der neuen Backend-Instanz übertragen und dort

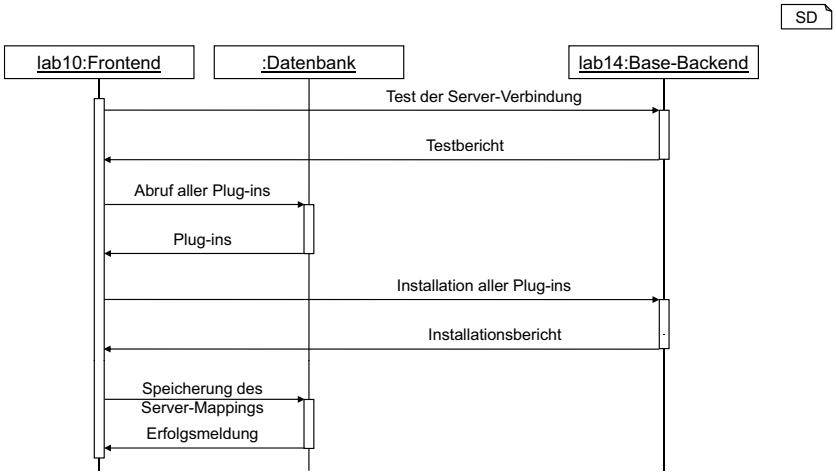


Abbildung 4.15: Sequenzdiagramm zur Synchronisation von Services der Backend-Instanzen

gemäß der Beschreibung aus Abschnitt 4.3.2 installiert. Nachdem die Backend-Instanz einen Bericht mit den Ergebnissen der Installation an das Frontend zurückgegeben hat, wird schließlich ein entsprechendes Base-Server-Mapping erzeugt und gespeichert, so dass die neue Backend-Instanz wie zuvor beschrieben durch einen Administrator getestet werden kann.

Die Synchronisation wird nicht nur automatisch bei der Registrierung einer neuen Backend-Instanz durchgeführt, sondern kann auch manuell durch einen Administrator angestoßen werden. Dazu gibt es in den administrativen Seiten des Frontends die Möglichkeit, für jede Backend-Art die Konsistenz aller installierten Service-Plug-ins der Backends gegen die vom Frontend gespeicherten zu prüfen. Dabei wird überprüft, ob alle gespeicherten Plug-ins in jeder Backend-Instanz installiert sind und zusätzlich ob in den Backend-Instanzen noch weitere nicht im Frontend gespeicherte Plug-ins installiert sind. Falls Inkonsistenzen bestehen, kann ein Administrator entsprechend die Synchronisation aller Plug-ins anstoßen, so dass auch das zweite Szenario unterstützt wird. Hier wäre auch eine weitere Automatisierung denkbar, so dass in regelmäßigen Abständen ebenfalls die Konsistenz aller Backends geprüft und bei Inkonsistenzen automatisch eine Synchronisation angestoßen wird.

Ein anderer Ansatz zu der beschriebenen Bereitstellung der Service-Plug-ins und der Konsistenzsicherung wird in [AAE+11] beschrieben. Die Autoren präsentieren die Konzepte und die Architektur von *Arvue*, einer Browser-basierten Anwendung zur Entwicklung und Publikation von Web-Anwendungen. Die Web-Anwendungen werden mit Hilfe eines GUI-Designers und des webbasierten Editors CoRED [LNK+12] erstellt und dann durch ein Deployment in einer Cloud-Umgebung verfügbar gemacht. In der Umgebung werden ebenfalls Plug-in-Systeme zur Ausführung der erstellten Anwendungen verwendet. Im Gegensatz zu dem hier vorgestellten Ansatz werden die Plug-in-Systeme jedoch erst bei Bedarf gestartet. Sie laden sich dann die benötigten Plug-ins selbständig aus einem zentralen Repository. Wenn eine Anwendung nicht

mehr aktiv ist, wird sie nach einer gewissen Zeit wieder gestoppt, so dass die belegten Ressourcen wieder freigegeben werden können. Im SSELab-Framework wurde davon abgesehen, die Backend-Instanzen erst bei Bedarf zu starten, da einerseits die Skalierung auf Ebene der Betriebssystem- bzw. Plattformressourcen erfolgen soll und andererseits bei diesem Ansatz gegebenenfalls Fehler erst beim Aufruf der Services durch die Benutzer auftreten und nicht bereits bei der Installation durch einen Administrator. Außerdem wäre die Nutzung der Services bei der in [AAE+11] beschriebenen dynamischen Bereitstellung langsamer als bei der für das Framework gewählten Strategie. Eine voll automatisierte Bereitstellung und Konfiguration neuer Backend-Instanzen durch das Frontend wäre jedoch ein sinnvolles Konzept und könnte auf Grundlage der vorhandenen Mechanismen des Frameworks effizient umgesetzt werden.

4.4.3 Kontenverwaltung

In Abschnitt 4.1.2 wurden die Kontotypen des Domänenmodells auf Grundlage der in Abschnitt 3.2 definierten Rollen eingeführt. In diesem Abschnitt werden die wichtigsten Verwaltungsmechanismen aller Kontotypen im Frontend beschrieben. Abbildung 4.16 zeigt dazu in einem verfeinerten Klassendiagramm die Beziehungen der relevanten Klassen des Domänenmodells. In der gezeigten Hierarchie wird zwischen personengebundenen Konten durch die Klasse `Person` und Konten für den automatisierten Zugriff durch die Klasse `Bot` unterschieden. Die Gemeinsamkeiten aller Konten werden durch die Klasse `Account` beschrieben, durch die auch Authentifizierung ermöglicht wird. Dementsprechend enthält sie Attribute zur Verwaltung des Benutzernamens bzw. der Kennung und dem zugehörigen Passwort eines Kontos.

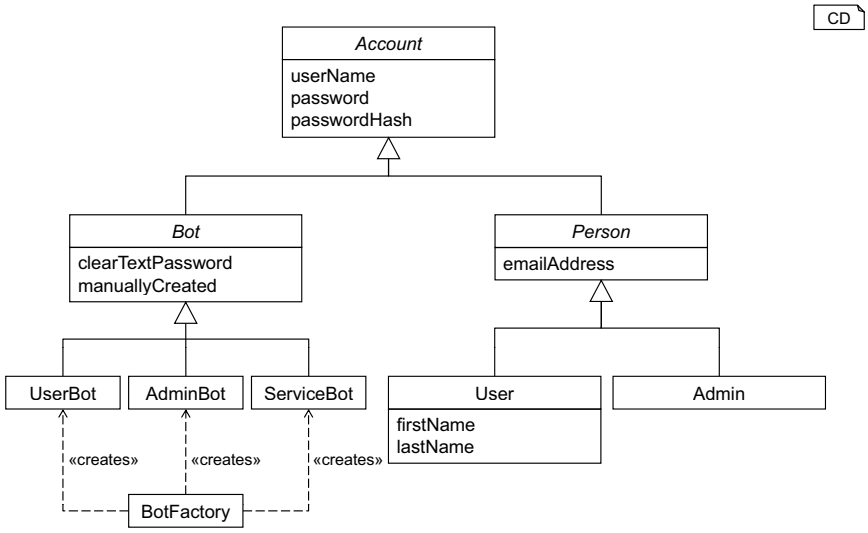


Abbildung 4.16: Details der Kontotypen im SSELab-Framework

Personengebundener Konten und Profile

Personengebundene Konten ermöglichen einen interaktiven Zugriff von Benutzern und Administratoren auf die Funktionen des Frontends. Die Konten für diese beiden Rollen werden dementsprechend durch die beiden Domänenklassen `User` und `Admin` repräsentiert, die jeweils eine Subklasse von `Person` sind. In diesen Klassen werden zusätzlich zu den Informationen der `Account`-Klasse noch einige grundlegende Informationen verwaltet. Alle personengebundenen Konten verfügen über eine primäre E-Mail-Adresse, die unter anderem für Benachrichtigungsmails vom Frontend genutzt wird. Von Benutzern werden darüber hinaus noch der Vor- und der Nachname gespeichert. Alle weiteren Informationen über einen Benutzer werden in einem zugehörigen Profil abgelegt.

Die Domänenklassen zur Verwaltung von Profilinformationen wurden im Rahmen der Erweiterung des Frameworks um Social-Services erstellt [Man12]. Der Fokus bei dem Entwurf des Domänenmodells lag dabei auf der Beschreibung von Informationen, die in sozialen Netzwerken gespeichert sind und über deren öffentlich zugänglichen APIs abgerufen werden können. Die Spezifikation von OpenSocial bezüglich solcher Informationen [OS11b] wurde dabei als Ausgangspunkt des Entwurfs verwendet. Abbildung 4.17 zeigt einen Ausschnitt aus dem Domänenmodell zur Speicherung von Profilinformationen.

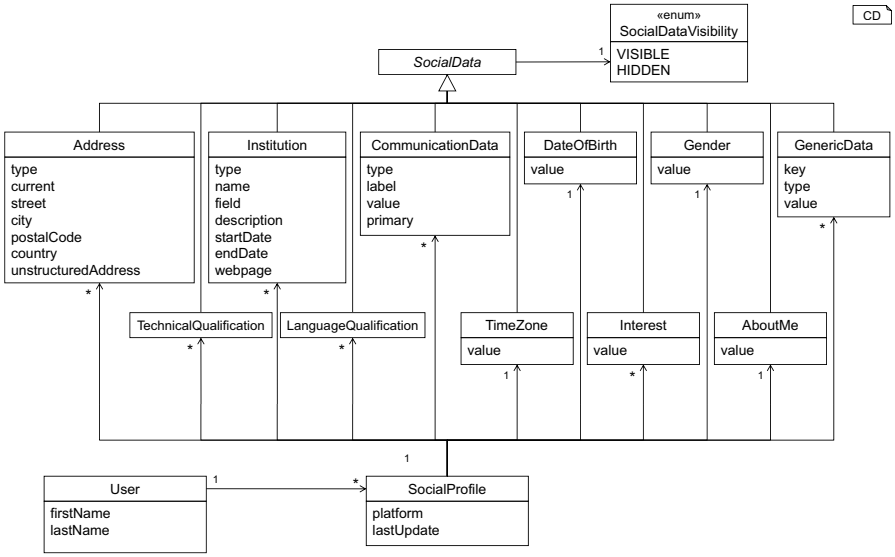


Abbildung 4.17: Domänenmodell zur Verwaltung von Profilinformationen der Benutzer

Die Klasse `SocialData` fasst die Gemeinsamkeiten aller Klassen des Domänenmodells zusammen, in denen Profilinformationen gespeichert werden. Diese Klassen werden in der folgenden Auflistung nur kurz beschrieben. Weitere Details sind in [Man12] enthalten.

- Die Adressen eines Benutzers werden mit Hilfe der Klasse `Address` verwaltet.
- Mit Hilfe der `Institution`-Klasse können Informationen über die Ausbildung und der berufliche Laufbahn von Benutzern gespeichert werden.
- Die `CommunicationData`-Klasse fasst Informationen über Kommunikationsmedien zusammen, über die ein Benutzer erreichbar ist.
- Die Klassen `DateOfBirth` und `Gender` geben das Geburtsdatum und das Geschlecht eines Benutzers an.
- Die beiden Klassen `TechnicalQualification` und `LanguageQualification` ermöglichen die Angabe von technischen Fähigkeiten und den gesprochenen Sprachen eines Benutzers.
- Durch die Klasse `TimeZone` wird die Zeitzone gespeichert, in der sich ein Benutzer aufhält.
- Die Klasse `Interest` beschreibt die Interessen und Hobbys von Benutzern.
- In der `AboutMe`-Klasse werden generelle Informationen über einen Benutzer abgelegt.
- Die Klasse `GenericData` bietet die Möglichkeit, weitere Informationen, die nicht durch die restlichen Klassen adäquat modelliert sind, nicht typisiert zu speichern. Dies ermöglicht eine Verwaltung beliebiger Informationen zur Laufzeit.

Die Klasse `SocialProfile` aggregiert alle Profilinformationen und ermöglicht die Entkopplung der `User`-Klasse von den Details der Profilklassen sowie die Speicherung beliebig vieler Profile für einen Benutzer. Ein spezielles Profil ist dabei das SSELab-Standardprofil [Man12]. Die Informationen, die in diesem Profil gespeichert werden, können vom Benutzer entweder manuell eingetragen oder aber aus den anderen Profilen übertragen werden. Die anderen Profile können ausschließlich durch den Import der Informationen aus sozialen Netzwerken über Social-Services erstellt werden. Nur das Standardprofil ist für andere Benutzer sichtbar und wird beispielsweise bei einer Suche nach Benutzern mit bestimmten Fähigkeiten berücksichtigt. Darüber hinaus kann gemäß Anforderung **FA4** (Verwaltung der Sichtbarkeit von Profilinformationen) die Sichtbarkeit einzelner Informationen des Standardprofils durch den Benutzer festgelegt werden. Im Domänenmodell ist die Klasse `SocialDataVisibility` dafür zuständig.

Der Einfluss und Nutzen von Social Software in Entwicklungsprojekten – insbesondere in global verteilten Projekten – ist ein aktuelles Forschungsgebiet mit vielen unbeantworteten Fragestellungen [STvDC10, Chi08]. Die Verarbeitung von Profilinformationen aus sozialen Netzwerken im Kontext von Software Engineering Projekten wurde in [Man12] auf Grundlage der Social-Services des Frameworks initial betrachtet. Darauf aufbauend könnten weitere Konzepte erarbeitet sowie das Framework beispielsweise um die Verarbeitung sozialer Graphen erweitert und dadurch offene Fragestellungen im Bereich Social Software in Entwicklungsprojekten behandelt werden.

Verwendung von Bot-Accounts

Neben den personengebundenen Konten für einen interaktiven Zugriff werden auch sogenannte Bot-Accounts für automatisierte Zugriffe auf das Frontend und die integrierten Services durch das Framework unterstützt (vgl. Anforderung **FA13**). Um eine Differenzierung dieser Konten von personengebundenen Konten zu ermöglichen, werden Bot-Accounts durch die Klasse `Bot` repräsentiert. Da ein Bot-Account entweder stellvertretend für einen Administrator, einen Benutzer oder einen Service agieren kann, gibt es jeweils einen entsprechenden Subtypen von `Bot`. In Abbildung 4.16 auf Seite 70 ist diese Hierarchie dargestellt.

Die Interaktion von Bot-Accounts mit dem Frontend oder den Services erfolgt in drei verschiedenen Varianten. Diese Interaktionsvarianten sind in Abbildung 4.18 schematisch dargestellt und werden im Folgenden beschrieben.

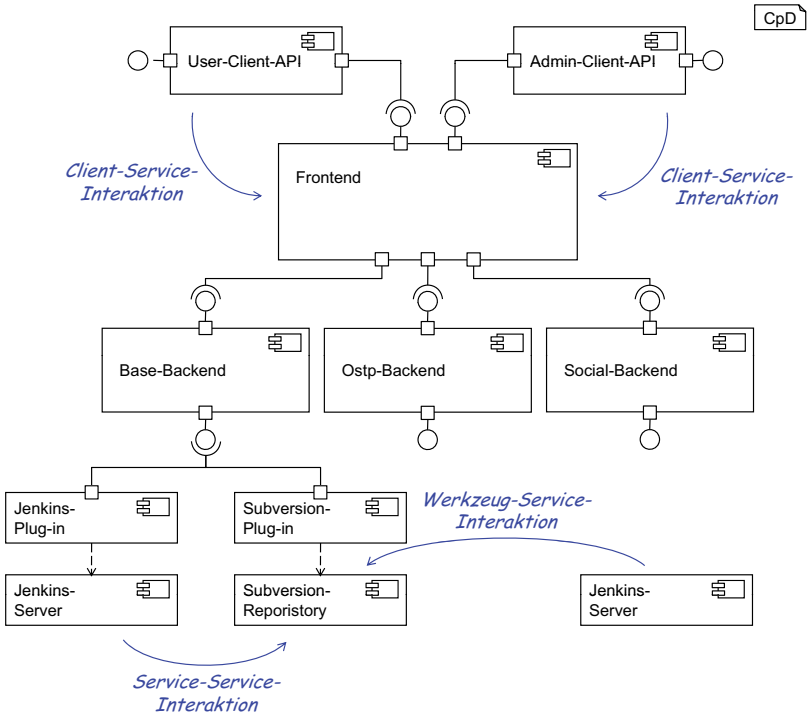


Abbildung 4.18: Interaktionsvarianten bei denen Bot-Accounts zum Einsatz kommen

Service-Service-Interaktion: Durch Base-Services werden server- und webbasierte Anwendungen in das Framework integriert. Der Zugriff auf einen Base-Service durch einen anderen Service wird als Service-Service-Interaktion bezeichnet. In Abbildung 4.18 ist als Bei-

spiel für diese Variante der Zugriff des integrierten Jenkins-Servers auf ein Subversion-Repository gezeigt. Jenkins überwacht das Repository und startet bei Änderungen einen konfigurierten Build-Prozess. Der Zugriff auf das Repository darf nur durch einen authentifizierten und autorisierten Benutzer erfolgen, so dass hier ein Bot-Account zum Einsatz kommen kann.

Bei dieser Variante werden die benötigten Bot-Accounts automatisch durch das Frontend bereitgestellt und zentral verwaltet. Die Konfiguration, ob ein Base-Service den Zugriff durch Bot-Accounts überhaupt unterstützt und auf welche Base-Services ein Service zugreifen möchte, erfolgt dabei dezentral in den zugehörigen Plug-ins. Die dabei verwendeten Mechanismen der Plug-in-API werden in Kapitel 5 behandelt. Die wichtigsten Aspekte der Verwaltung der benötigten Bot-Accounts im Frontend werden hier skizziert. Weitere Details dazu sind in [Web11] enthalten.

Bei der Aktivierung eines Services für ein Projekt wird vom Frontend auf Grundlage der Plug-in-API geprüft, ob dieser auf andere Base-Services zugreifen möchte. Ist das der Fall, wird noch kontrolliert, ob der Ziel-Service überhaupt einen Zugriff durch Bot-Accounts unterstützt. Im positiven Fall erstellt das Frontend einen Service-Bot und fügt diesen dem entsprechenden Projekt hinzu.

Werkzeug-Service-Interaktion: Gemäß Anforderung **FA25** sollen externe Werkzeuge, die nicht in das Framework integriert worden sind, auf Services einer Framework-Instanz zugreifen können. Für diese Art der Interaktion werden ebenfalls Service-Bots verwendet. In diesem Fall ist eine automatische Bereitstellung der Bot-Accounts jedoch nicht möglich, da eine Framework-Instanz keine Informationen über externe Werkzeuge besitzt. Daher können die benötigten Bot-Accounts für die Werkzeug-Service-Interaktion von Benutzern oder Administratoren manuell erstellt werden. Dazu bietet die Web-Oberfläche die Möglichkeit, Bot-Accounts im Kontext eines Projekts anzulegen und zu verwalten. Nach der Erstellung eines Service-Bots kann dieser zur Konfiguration des externen Werkzeugs verwendet werden.

Client-Service-Interaktion: Die Framework-Clients ermöglichen es den Benutzern und Administratoren, nicht nur über Web-Browser, sondern auch über die Kommandozeile oder aus einer IDE heraus, Funktionen oder Services einer Framework-Instanz zu nutzen. Die verfügbaren Clients werden im Frontend verwaltet und können darüber heruntergeladen werden. Im folgenden Abschnitt 4.5 werden sowohl die Architektur und die Eigenschaften der Clients beschrieben.

Bei der Verwendung eines Clients muss sich ein Benutzer oder Administrator mit Hilfe einer Kennung und dem zugehörigen Passwort authentifizieren. Alternativ kann ein Client unter Verwendung von Bot-Accounts personalisiert werden. Dabei wird beim Herunterladen eines Clients in Abhängigkeit von der Rolle der angemeldeten Person ein Bot-Account erstellt und der Person zugeordnet. Der Client wird dann mit Hilfe der Kennung und des Passworts des Bot-Accounts personalisiert. In Abschnitt 4.4.5 werden diese Verwaltungsmechanismen von Framework-Clients genauer beschrieben.

Persistierung von Konten

In einem Konto werden die Kennung und das Passwort gespeichert, die sowohl zur Authentifizierung als auch zur Autorisierung verwendet werden. Alle Konten werden durch das Frontend in der Datenbank verwaltet. In der Realisierung des Frameworks wird bei der Persistierung eines Kontos das Passwort nicht im Klartext, sondern ausschließlich als Hashwert in der Datenbank abgelegt. Dabei wird aktuell der SHA-1-Algorithmus [EH11] in Kombination mit einer Base64-Kodierung [Jos06] verwendet. Diese Verfahren wurden gewählt, da sie von vielen Werkzeugen und Servern unterstützt werden, so dass Anforderung **NFA3** erfüllt wird. Bei Bedarf können jedoch auch andere Hash-Algorithmen und Kodierungen zur Verwaltung der Passwörter unterstützt werden.

Die Konten werden vom Frontend nicht nur in der Datenbank abgelegt, sondern auch über ein LDAP-Verzeichnis bereitgestellt. Mit Hilfe dieses Verzeichnis können Anwendungen auf die Informationen zur Authentifizierung und Autorisierung zugreifen, so dass Anforderung **FA55** umgesetzt werden kann.

4.4.4 Projektverwaltung

Projekte fassen Benutzer und Services zusammen und ermöglichen die Kollaboration der Benutzer unter Verwendung der Services. Einige der Eigenschaften von Projekten wurden bereits in Abschnitt 4.1.3 beschrieben. Darauf aufbauend werden in diesem Abschnitt die wesentlichen Aspekte der Projektverwaltung im Frontend behandelt. Abbildung 4.19 zeigt dazu einen Ausschnitt des Domänenmodells mit den Beziehungen der `Project`-Klasse zu einigen anderen Klassen. Im Folgenden werden anhand dieses und weiter verfeinerter Klassendiagramme sowohl die Eigenschaften als auch die Verwaltung von Projekten diskutiert.

Lebenszyklus von Projekten

Gemäß den Anforderungen **FA2** (Zugriff als Gast), **FA5** (Erzeugung neuer Projekte) und **FA39** (Verwaltung von Projekten) können Projekte sowohl von Gästen, Benutzern und Administratoren angelegt werden. Beim Erstellen eines Projekts muss ein eindeutiger Name und eine Beschreibung des Projekts angegeben werden. Optional kann ein Ablaufdatum bestimmt werden, an dem das Projekt voraussichtlich abgeschlossen ist. Wenn das Datum erreicht ist, wird eine Benachrichtigung an die Mitglieder des Projekts gesendet. Die Projektverantwortlichen können dann entscheiden, ob das Projekt deaktiviert werden kann oder weitergeführt werden soll. Bei der Erstellung eines Projekts können darüber hinaus noch die Services ausgewählt werden, die genutzt werden sollen. Resultierend aus Anforderung **FA10** (Kontext der Services) und den Eigenschaften der Service-Kategorien können nur Base- und Ostp-Services im Kontext eines Projekts verwendet werden.

Wenn ein neues Projekt von einem Gast oder einem Benutzer angelegt wird, muss dabei ein gültiger Benutzer-Account angegeben werden, so dass einem neu angelegten Projekt immer ein Benutzer zugeordnet ist. Bei der Registrierung eines neuen Projekts als Gast kann dementsprechend ein neuer Account erstellt oder ein existierender Account genutzt werden. Bei der Erstellung eines Projekts als Benutzer wird im Gegensatz dazu automatisch der aktuell authen-

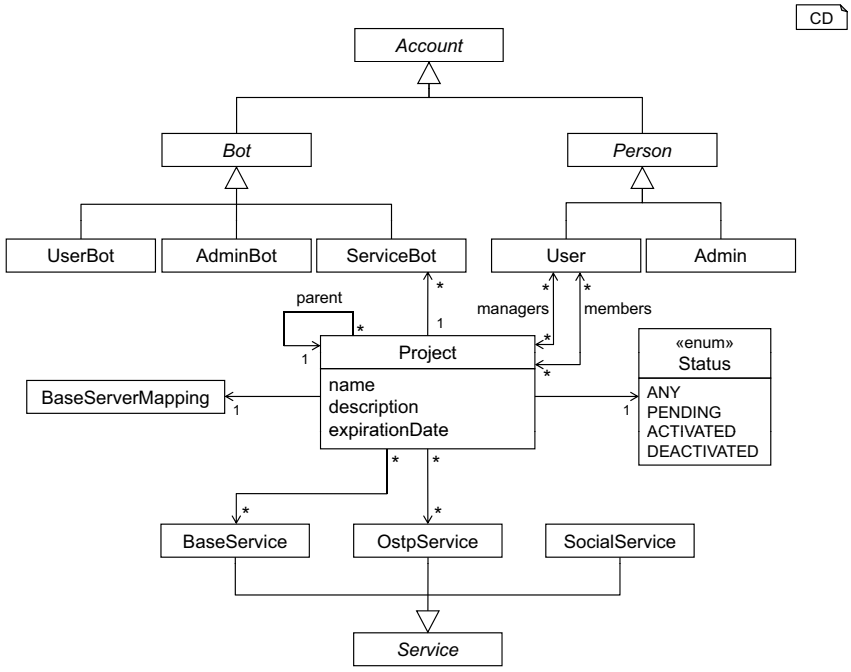


Abbildung 4.19: Projekte und deren Beziehungen zu weiteren Klassen des Domänenmodells

tifizierte Account verwendet. In jedem Fall wird der entsprechende Account dem Projekt in der Rolle Manager hinzugefügt.

Nach der Erstellung eines Projekts befindet es sich initial in dem Zustand `PENDING`. Das bedeutet, dass es noch nicht aktiviert ist, die Services noch nicht konfiguriert sind und der Zugriff durch die Benutzer noch nicht möglich ist. Ein Projekt in diesem Zustand muss erst durch einen Administrator freigeschaltet und damit aktiviert werden, so dass es in den Zustand `ACTIVATED` überführt wird. Bei der Aktivierung muss der Administrator den Server mit einer der verfügbaren Base-Backend-Instanzen auswählen auf dem die Base-Services des Projekts konfiguriert werden sollen. Dann werden dem Projekt ein Base-Server-Mapping zugeordnet und sowohl alle zugeordneten Services konfiguriert und die Manager und Member des Projekts per E-Mail über die Aktivierung und die verfügbaren Services informiert.

Nachdem ein Projekt aktiviert worden ist, kann es durch einen Administrator wieder deaktiviert werden. Ein wesentliches Konzept dabei ist, dass bei der Deaktivierung eines Projekts keine persistenten Daten eines Services gelöscht werden, sondern nur der Zugriff auf alle Service-Instanzen des Projekts unterbunden wird. Dies verhindert einen ungewollten Datenverlust, da nach einer erneuten Aktivierung eines deaktivierten Projekts alle zuvor gespeicherten Daten noch immer verfügbar sind.

Verwaltung zugriffsberechtigter Konten von Projekten

Gemäß Abbildung 4.19 können Benutzer – entweder in der Rolle Manager oder Member – und Service-Bots Teil eines Projekts sein. Die Verwaltung der Konten in einem Projekt kann entweder durch die Administratoren (vgl. Anforderung **FA39**) oder durch die Manager des Projekts (vgl. Anforderung **FA8**) erfolgen. Administratoren können alle Aspekte eines Projekts über die Web-Oberfläche des Frontends konfigurieren, aber nicht auf die Services zugreifen. Auf die einzelnen Funktionen für Administratoren wird hier nicht weiter eingegangen. Anstatt dessen werden die Verwaltungsmechanismen aus Sicht der Benutzer anhand des Domänenmodells beschrieben.

Entsprechend Anforderung **FA8** ist die eigenständige Verwaltung von Projekten durch Manager und ohne Intervention durch Administratoren ein wichtiges Konzept. Manager können sowohl die Bot-Accounts als auch die anderen Manager und Member eines Projekts verwalten. Neue Benutzer können mit Hilfe von Einladungen in das Projekt aufgenommen werden. Abbildung 4.20 zeigt die zugehörige Klasse `Invitation` aus dem Domänenmodell.

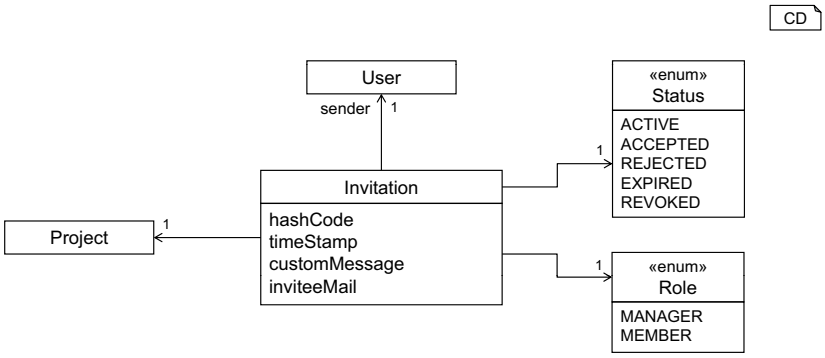


Abbildung 4.20: Domänenmodell zur Repräsentation von Einladungen zu Projekten

Eine Einladung wird immer im Kontext eines Projekts von einem Manager erstellt und versandt, so dass die `Invitation`-Klasse entsprechende Assoziationen zu den Klassen `Project` und `User` besitzt. Um Einladungen identifizieren zu können, wird bei deren Erstellung vom Frontend ein eindeutiger Hash-Code generiert und der Einladung zugewiesen. Dieser Hash-Code wird dann bei der Konstruktion der URL verwendet, über die die Seite erreichbar ist, auf der die Einladung akzeptiert oder abgelehnt werden kann. Weiterhin wird in einer Einladung noch der Zeitpunkt der Erstellung, ein optionaler Einladungstext vom Manager und die Rolle, die der Eingeladene im Projekt haben soll, gespeichert.

Der Einladungsmechanismus ist so konzipiert, dass eine Einladung unabhängig davon erstellt wird, ob der Empfänger bereits einen Account besitzt oder nicht. Daher wird in einer Einladung nur die E-Mail-Adresse des Empfängers gespeichert. Bei der Annahme der Einladung kann der Empfänger dann entweder einen existierenden Account verwenden oder einen neuen Account erzeugen. Falls bei der Erstellung einer Einladung existierende Accounts zur Auswahl angezeigt werden würden, wäre Anforderung **NFA4** verletzt, die fordert, dass ein Benutzer nur dann Infor-

mationen über andere Benutzer erhalten kann, wenn diese beiden mindestens ein gemeinsames Projekt besitzen.

Der Lebenszyklus einer Einladung wird über deren Status gesteuert. Nach der Erstellung einer Einladung besitzt diese zunächst den Zustand `ACTIVE`. Nachdem der Empfänger die Einladung angenommen oder abgelehnt hat, besitzt sie den Zustand `ACCEPTED` bzw. `REJECTED`. Wird die Einladung vom Empfänger nicht innerhalb einer bestimmten Zeit angenommen, verfällt sie und besitzt dann den Zustand `EXPIRED`. Zusätzlich kann eine Einladung vor Ablauf der Gültigkeitsdauer durch einen Manager wieder zurückgezogen werden. Ihr Zustand ist dann `REVOKED`.

Verwaltung der Services von Projekten

Gemäß Anforderung **FA6** (Nutzung von Projekten) sollen die Manager und Member eines Projekts die Services konfigurieren, parametrisieren und verwenden können. Im Kontext von Projekten werden primär Base-Services genutzt. Bei der Aktivierung eines Base-Services für ein Projekt wird dieser instanziiert und für die Verwendung durch die Manager und Member des Projekts konfiguriert. Ostp-Services können auch einem Projekt zugeordnet sein, werden jedoch weder für ein Projekt instanziiert noch mit projektspezifischen Informationen aufgerufen, so dass der Projektkontext optional ist. Dieser Sachverhalt wird auch durch die Beziehungen zwischen einem Projekt und diesen beiden Service-Kategorien im Domänenmodell repräsentiert. Abbildung 4.21 zeigt die zugehörigen Klassen.

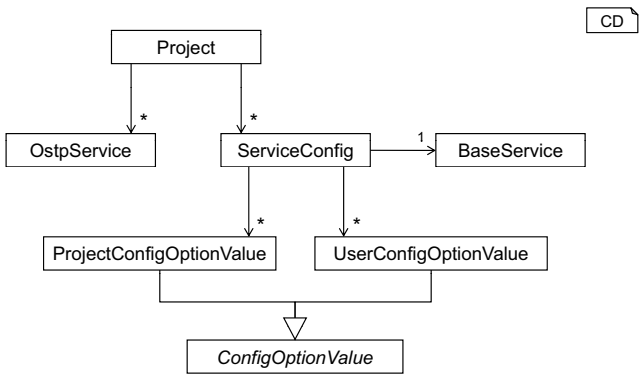


Abbildung 4.21: Klassen zur Beschreibung der Beziehungen zwischen Services und Projekten

Jede `Service`-Klasse repräsentiert gemäß Abschnitt 4.4.2 einen verfügbaren Service bzw. das zugehörigen Werkzeug. Die Instanziierung von Base-Services für ein Projekt wird durch die indirekte Assoziation über die Klasse `ServiceConfig` repräsentiert. Ostp-Services werden dagegen nicht für ein Projekt instanziiert, so dass diese direkt assoziiert sind. Die Klasse `ServiceConfig` beschreibt die Zuordnung der Konfiguration eines Base-Service im Kontext eines Projekts. Mit Hilfe der Klasse `ConfigOptionValue` werden die konkreten Konfigurationen

beschrieben. Dabei werden projektweite und benutzerspezifische Konfigurationsmöglichkeiten unterschieden. Die zugehörigen Konfigurationsoptionen auf Grundlage der Base-Plug-in-API werden in Kapitel 5 genauer erläutert.

Wie zuvor beschrieben, können Services bei der Erzeugung eines Projekts ausgewählt werden. Bei der Aktivierung eines Projekts erfolgt dann die Instanziierung und Konfiguration der ausgewählten Base-Services. Die Administratoren und die Manager eines Projekts können jedoch auch zu einem beliebigen Zeitpunkt neue Services für ein Projekt aktivieren oder deaktivieren.

4.4.5 Client-Verwaltung

Die Framework-Clients können über das Frontend zentral verwaltet werden. Die Motivation dafür ist einerseits, dass den Benutzern alle Funktionen über das Frontend angeboten werden sollen, wozu auch die Bereitstellung der Clients gehört. Andererseits können die Clients vor der Verwendung durch einen Benutzer oder Administrator durch das Frontend mit einem Bot-Account konfiguriert werden (vgl. Abschnitt 4.4.3). In diesem Abschnitt werden diese Verwaltungsmechanismen für Framework-Clients beschrieben. Abbildung 4.22 zeigt dazu zunächst ein Klassendiagramm mit dem relevanten Ausschnitt des Domänenmodells.

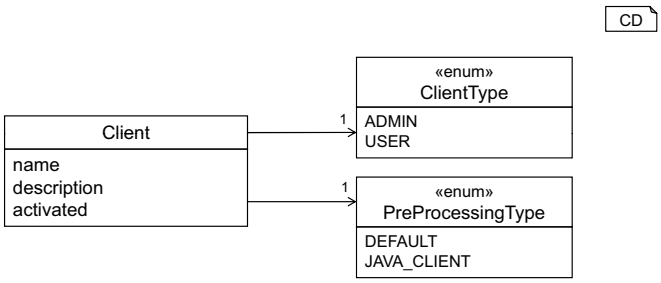


Abbildung 4.22: Domänenmodell zur Verwaltung von Clients

Durch die Framework-Clients können einzelne Funktionen des Frontends oder Services in verschiedenen Kontexten – beispielsweise aus einer IDE heraus – aufgerufen werden. Über die Web-Oberfläche des Frontends können diese Clients von Administratoren verwaltet werden, so dass sie von den Benutzern heruntergeladen und in ihren Projekten genutzt werden können. Die Verwaltung der Clients im Frontend erfolgt auf Grundlage der in Abbildung 4.22 gezeigten Klassen. Jeder Client besitzt einen Namen und eine Beschreibung, die durch den Administrator bestimmt werden können. Da sowohl Clients mit Funktionalität für Administratoren als auch für Benutzer im Framework vorgesehen sind, kann die Zielgruppe ebenfalls festgelegt werden.

Wie zuvor beschrieben, können Clients vor dem herunterladen mit einem Bot-Account konfiguriert und damit personalisiert werden. Diese Eigenschaft wird mit Hilfe der Klasse `PreProcessingType` beschrieben. In der Abbildung sind exemplarisch zwei Vorverarbeitungsarten angegeben, die in der Realisierung des Frameworks unterstützt werden. Die Client-Verwaltung ist jedoch so konzipiert, dass weitere Vorverarbeitungsarten leicht ergänzt werden können [Web11].

`DEFAULT`: Bei dieser Art der Vorverarbeitung wird die Client-Datei vor dem Herunterladen durch einen Benutzer oder Administrator nicht weiter manipuliert, sondern unverändert ausgeliefert.

`JAVA_CLIENT`: Durch diese Art der Vorverarbeitung werden ausschließlich Java-Archive als Clients unterstützt. Vor dem Herunterladen eines solchen Clients durch einen Benutzer oder Administrator wird ein entsprechender Bot-Account vom Frontend erzeugt und dessen Benutzername und Passwort zur Konfiguration des Archivs verwendet, so dass die Informationen zur Laufzeit durch den Client zur Authentifizierung genutzt werden können.

Die Vorverarbeitung eines Clients findet in einem personalisierten Download-Verzeichnis des Servers statt. Nach erfolgreicher Vorverarbeitung wird eine URL auf die Client-Datei in der Web-Oberfläche des Frontends angezeigt. Der Zugriff auf diese URL ist geschützt und nur für den Benutzer möglich, der den Client angefordert hat. Die Client-Dateien werden nach Ablauf einer konfigurierbaren Zeitspanne aus dem Download-Verzeichnis gelöscht und müssen bei einer erneuten Anforderung wieder erzeugt werden. Dabei wird jedoch der bereits erzeugte Bot-Account wiederverwendet. Insgesamt ist durch diese Mechanismen sichergestellt, dass mit Bot-Accounts personalisierte Clients innerhalb des Frontends nur von autorisierten Benutzern verwendet werden können. Nach dem Download eines solchen Clients muss ein Benutzer selbst darauf achten, dass der Client nicht durch Dritte verwendet wird, um einen unberechtigten Zugriff auf die Services eines Projekts zu verhindern.

4.5 Architektur und Funktionsumfang der Clients

Die Kopplung und Integration webbasierter und desktopbasierter Entwicklungsumgebungen ist gemäß Abschnitt 1.2 ein wichtiger Aspekt. Beim Entwurf des Frameworks wurde daher auf die Unterstützung vielfältiger Clients Wert gelegt, die in verschiedenen Kontexten eine effektive Nutzung der Funktionen einer Framework-Instanz und der integrierten Services ermöglichen. Die folgende Auflistung gibt noch einmal einen Überblick über die wichtigsten Anforderungen, die in Kapitel 3 in Bezug auf diese Clients definiert worden sind.

- **FA1** *Nutzung über zentrale Web-Oberfläche*
- **FA12** *Bereitstellung von Clients (Framework-Clients)*
- **FA13** *Automatisierter Zugriff*
- **FA24** *Unterstützung existierender Clients für Werkzeuge (Service-Clients)*
- **FA35** *Zentrale Administration des Systems*
- **FA42** *Verwaltung und Verwendung von Clients*
- **FA47** *Entwicklung als webbasiertes Framework*

- **FA54** *Unabhängige Entwicklung von Framework-Clients*
- **NFA2** *Sichere Kommunikation*
- **NFA12** *Sichere Kommunikation innerhalb des Frameworks*

Aus diesen Anforderungen lassen sich insgesamt drei Kategorien von Clients ableiten, die durch das Framework unterstützt werden.

Browser-Clients: Ein Großteil der Funktionalität einer Framework-Instanz soll gemäß den beiden Anforderungen **FA1** und **FA35** per Web-Browser genutzt werden können. Zur Umsetzung dieser Anforderungen stellt das Frontend eine Web-Anwendung zur Verfügung, die dies ermöglicht.

Service-Clients: Um Anforderung **FA24** umzusetzen, wurde bei der Integration von Werkzeugen als Service immer auf den Einsatz verbreiteter Technologien und Server geachtet, so dass existierenden Clients unverändert genutzt werden können. In den Anforderungen wurden diese Clients als Service-Clients bezeichnet, um sie von den Clients zu unterscheiden, die im Rahmen dieser Arbeit speziell für das Framework entworfen worden sind.

Framework-Clients: Die restlichen funktionalen Anforderungen der vorigen Auflistung haben Auswirkungen auf die Architektur und die Eigenschaften der Framework-Clients. Im Folgenden werden auf Grundlage dieser Anforderungen die wichtigsten Konzepte dieser Clients diskutiert.

Die im Rahmen dieser Arbeit konzipierten Framework-Clients unterstützen einerseits administrative Tätigkeiten – beispielsweise die Installation und Aktualisierung von Service-Plug-ins – und andererseits die Nutzung von Services. Die Clients für die Durchführung administrativer Aufgaben sind primär dazu gedacht, diese Aufgaben zu automatisieren. Ein Beispiel dafür ist die Installation oder Aktualisierung eines Services im Rahmen eines Build- oder Release-Prozesses. Die Clients zur Nutzung von Services sind als Alternative zum Web-Browser gedacht, um die Verwendung von Services auch über die Kommandozeile oder aus IDEs wie Eclipse heraus zu ermöglichen und dadurch die Kopplung von IDEs und CDEs zu unterstützen. Durch diese Clients werden jedoch nicht alle Service-Kategorien des Frameworks unterstützt. Die Gründe dafür werden im Folgenden kurz diskutiert.

- *Base-Services* sind server- oder webbasierte Anwendungen. Die meisten dieser Werkzeuge können entweder direkt über einen Web-Browser effektiv verwendet werden (z.B. MediaWiki, Jenkins) oder sind zusätzlich über existierende Service-Clients aus IDEs wie Eclipse heraus nutzbar (z.B. Subversion, Trac). Da das Framework sowohl die Nutzung über Web-Browser als auch von Service-Clients unterstützt und darauf ausgelegt ist, bestehende Integrationsmechanismen zu erhalten, wurde im Rahmen dieser Arbeit darauf verzichtet, eigenständige Clients für Base-Services zu realisieren.
- *Ostp-Services* sind dateiverarbeitende Werkzeuge, die üblicherweise lokal und nicht in einer Serverumgebung ausgeführt werden. Bei der Integration solcher Werkzeuge in das

Framework werden sie jedoch als Hosted Services betrieben. Die Nutzung dieser Werkzeuge über einen Browser ist in vielen Fällen nicht effizient möglich, so dass im Rahmen dieser Arbeit mehrere Framework-Clients speziell für die Nutzung von Ostp-Services konzipiert worden sind.

- *Social-Services* werden im Framework zur Verarbeitung von Informationen im Kontext von Benutzerprofilen eingesetzt. Die Verwendung von Web-Browsern ist in diesem Fall gängig und effizient, so dass keine spezifischen Clients für die Nutzung von Social-Services vorgesehen sind.

Alle Framework-Clients basieren auf einer von zwei Client-APIs, die durch das Framework angeboten werden. In dem folgenden Abschnitt wird zunächst die Architektur dieser APIs beschrieben und anschließend der Funktionsumfang der realisierten Framework-Clients behandelt. Darauf aufbauend erfolgt dann in Kapitel 5 die Diskussion der Erweiterungsmechanismen der APIs und der realisierten Clients.

4.5.1 Architektur der Client-APIs

Beim Entwurf der Architektur der Framework-Clients wurde auf den Einsatz einheitlicher Konzepte geachtet. Daher basieren alle Framework-Clients auf einer von zwei APIs, die für den Zugriff von Benutzern bzw. von Administratoren auf Funktionen des Frontends ausgelegt sind. Der Aufbau beider APIs ist in Abbildung 4.23 gezeigt. Die Komponenten *SSELab-Domain* und *SSELab-Util* werden in beiden APIs und damit in den Framework-Clients unverändert genutzt. Die Verwendung dieser Komponenten ist nicht zwingend erforderlich, ermöglicht aber eine effizientere Realisierung der Clients, da existierende Konzepte und Funktionalität wiederverwendet werden können. Die restlichen Komponenten der beiden Client-APIs bauen auf diesen Komponenten auf.

- Das Frontend einer SSELab-Instanz bietet Schnittstellen für den Zugriff von Benutzern und von Administratoren an. Die Komponenten *User-Service-Client* und *Admin-Service-Client* kapseln jeweils die Kommunikationsmechanismen über die entsprechende Schnittstelle und bieten den übrigen Komponenten den Zugriff darauf an. Ein Aufruf der Schnittstelle ist nur nach erfolgreicher Authentifizierung und Autorisierung eines Benutzers oder Administrators bzw. eines Bot-Accounts möglich. Dadurch wird eine sichere Kommunikation zwischen den Clients und dem Frontend gewährleistet, so dass Anforderung **NFA12** erfüllt werden kann. Darüber hinaus ermöglicht die erreichte Entkopplung durch die Client-Komponenten eine vom Frontend unabhängige Entwicklung von Framework-Clients, so dass auch Anforderung **FA54** umgesetzt werden kann.

- Die Komponenten *User-Client-Common* und *Admin-Client-Common* fassen jeweils wesentliche Funktionalität zusammen, die beim Zugriff auf das Frontend durch Benutzer oder Administratoren benötigt wird: Die Komponenten führen einerseits eine automatische Prüfung beim Zugriff auf das Frontend durch, ob der verwendete Client noch mit dem Server kompatibel ist. Falls dies nicht der Fall ist – beispielsweise weil sich die jeweilige Schnittstelle auf eine inkompatible Weise geändert hat – wird eine entsprechende Fehlermeldung

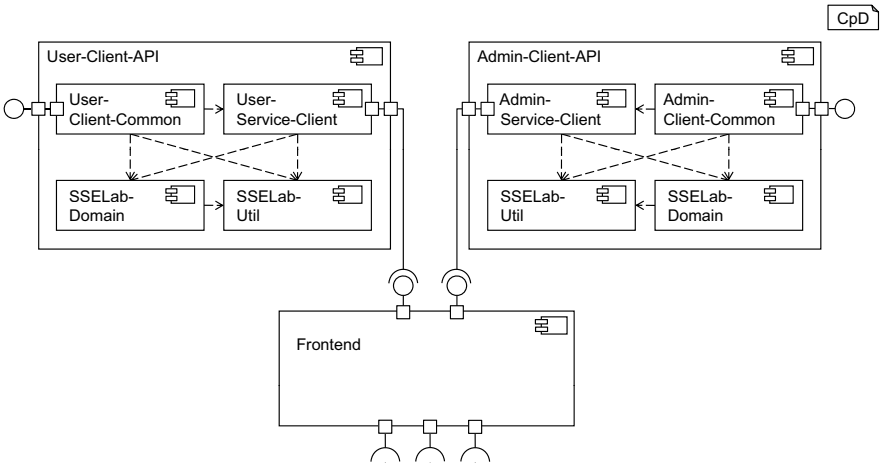


Abbildung 4.23: Komponenten der Framework-Client-APIs

erzeugt, damit der Benutzer oder Administrator auf die notwendige Aktualisierung des Clients hingewiesen werden kann. Andererseits wird durch diese Komponenten die Authentifizierung und Autorisierung eines Benutzers oder Administrators beim Frontend unterstützt.

Neben den Gemeinsamkeiten beider Komponenten gibt es natürlich auch Funktionalität, die auf Grundlage der rollenspezifischen Schnittstelle angeboten wird. Die Komponente *User-Client-Common* ermöglicht nach erfolgreicher Authentifizierung eines Benutzers beispielsweise den Abruf von Informationen über die Projekte des Benutzers und über alle verfügbaren Ostp-Services sowie deren Aufruf. Die Komponente *Admin-Client-Common* enthält dagegen Funktionen für die Installation und Deinstallation von Service-Plug-ins.

4.5.2 Realisierte Clients

Aufbauend auf den zuvor vorgestellten APIs wurden im Rahmen dieser Arbeit mehrere Framework-Clients realisiert, von denen einige in diesem Abschnitt vorgestellt werden. Abbildung 4.24 gibt einen Überblick über diese Clients. Die Verwendung der APIs zur Entwicklung weiterer Framework-Clients wird dann genauer im folgenden Kapitel 5 behandelt.

- Aufbauend auf der User-Client-API realisiert der *Eclipse-Client* die Integration in die Eclipse IDE [dRW04]. Unter Verwendung der Mechanismen von Eclipse können die Authentifizierung eines Benutzers und danach der Aufruf aller verfügbaren Ostp-Services direkt aus Eclipse heraus erfolgen. Durch diesen Client wird die servicebasierte Nutzung von Transformationswerkzeugen, die nicht für eine Ausführung in einer Serverumgebung konzipiert worden sind, aus einer IDE heraus unterstützt. Durch die Realisierung dieses

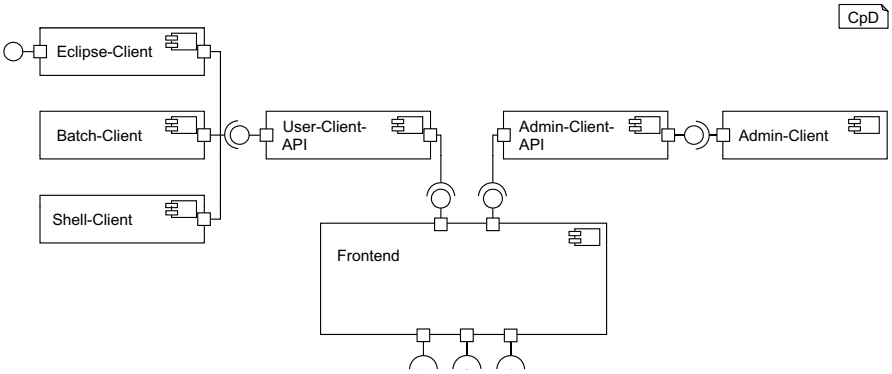


Abbildung 4.24: Realisierte Framework-Clients

Clients konnte die Praxistauglichkeit der Konzepte des Frameworks zur Kopplung von IDEs und CDEs gezeigt werden.

- Der *Batch-Client* realisiert ein Kommandozeilen-Client zur Ausführung von Ostp-Services. Beim Aufruf des Clients müssen alle benötigten Informationen angegeben werden, um den aufzurufenden Service und dessen Parametrisierung bestimmen zu können. Dieser Client wurde insbesondere in Szenarien erfolgreich eingesetzt, bei denen die Automatisierung von Abläufen im Vordergrund steht. Beispielsweise bei der Durchführung automatisierter Build-Prozesse.

- Der *Shell-Client* ist genau wie der Batch-Client eine Kommandozeilenanwendung für die Ausführung von Ostp-Services. Der Shell-Client läuft jedoch in einem interaktiven Modus, in dem die Eingabedateien und Parameterwerte eines Ostp-Services durch den Benutzer angegeben werden. Dieser Client wurde nur prototypisch umgesetzt und nicht produktiv eingesetzt.

- Der *Admin-Client* ermöglicht die Verwaltung von Service-Plug-ins über die Kommandozeile mit Hilfe eines Administrator- oder eines entsprechenden Bot-Accounts. Dabei können mit Hilfe des Clients sowohl Informationen über die aktuell installierten Plug-ins abgerufen als auch Plug-ins installiert, aktualisiert oder deinstalliert werden. Dieser Client wurde im Kontext automatisierter Deployment-Prozesse erfolgreich genutzt.

Wie durch diese verschiedenen Beispiele illustriert wird, ermöglichen die Client-APIs die Entwicklung diverser Framework-Clients. Diese Clients können jedoch selbst auch erweiterbar entworfen sein. Der Eclipse-Client ist ein Beispiel dafür. Dessen Erweiterung zur Unterstützung servicespezifischer Funktionen wird im folgenden Kapitel 5 behandelt. Ein Beispiel für eine konkrete Erweiterung des Clients wird dann in Kapitel 6 gegeben.

4.6 Zusammenfassung

In diesem Kapitel wurden die Konzepte und die Architektur eines Frameworks für webbasierte Projektportale und CDEs vorgestellt, das auf eine servicebasierte Nutzung von Werkzeugen ausgerichtet ist. Als Grundlage des Frameworks wurde zunächst das im Rahmen dieser Arbeit entwickelte Domänenmodell behandelt, durch das wesentliche Aspekte von Projektportalen und CDEs beschrieben werden. Dazu gehören Konzepte wie Benutzer, Projekte und Services sowie deren Beziehungen zueinander. Das Modell ist bewusst abstrakt gehalten, so dass einerseits eine vielfältige und große Menge an Werkzeugen und andererseits Projekte mit unterschiedlichen Prozessen und Methoden unterstützt werden können. Um spezifische Eigenschaften konkreter Werkzeuge berücksichtigen zu können, wurden im Kontext der Services mehrere Service-Kategorien definiert, die Gemeinsamkeiten verwandter Werkzeuge zusammenfassen. Das Framework ist jedoch so konzipiert, dass auch weitere Service-Kategorien bei Bedarf ergänzt werden können.

Aufbauend auf dem Domänenmodell wurden sowohl die Architektur des Frameworks als auch das Zusammenspiel der verteilten Framework-Komponenten detailliert diskutiert. Neben der Architektur wurden dabei die Unterstützung von Services in den Backends, die Verwaltungsmechanismen des Frontends sowie der Funktionsumfang der Clients in den Vordergrund gestellt. Dabei verdeutlicht insbesondere die Beschreibung des Frontends, dass die Realisierung eines solchen Frameworks in einer Mischung aus Framework- und Anwendungsentwicklung erfolgt: Einerseits wurde ein Framework mit mehreren Erweiterungspunkten und andererseits ein verteiltes System beschrieben, das bereits ohne konkrete Services und Clients lauffähig ist und eine umfangreiche Geschäftslogik enthält.

Kapitel 5

Methodik zur Erweiterung des Frameworks

Das SSELab ist ein Framework für webbasierte Projektportale und CDEs. Daher ist die Erweiterbarkeit ein wesentliches Konzept, das sich im Design und der Funktionalität aller Komponenten widerspiegelt. Aufbauend auf dem vorigen Kapitel, das die Konzepte der Architektur und der Verwaltungsmechanismen des Frameworks beschreibt, widmet sich dieses Kapitel den Erweiterungspunkten, also den Schnittstellen und APIs des Frameworks. In Abbildung 5.1 sind diese innerhalb der Architektur des Frameworks gezeigt.

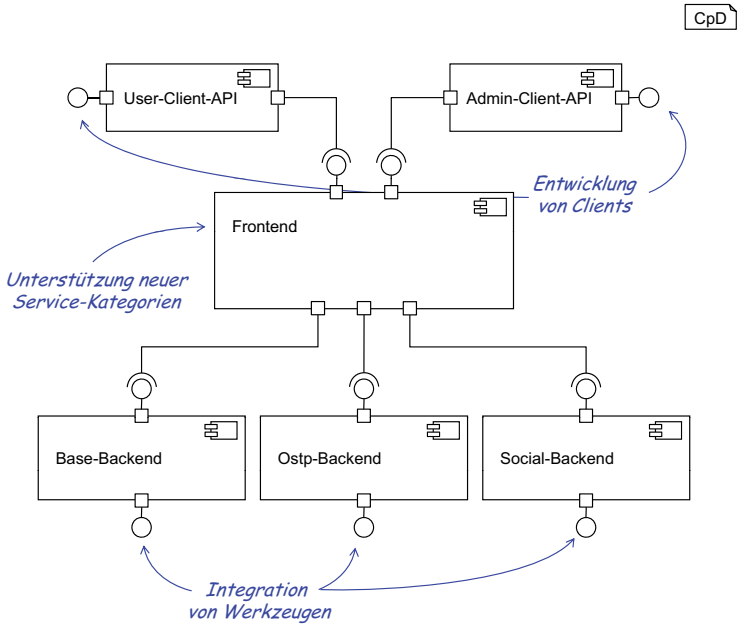


Abbildung 5.1: Erweiterungspunkte des SSELab-Frameworks

Die in der Abbildung dargestellten Erweiterungspunkte basieren auf den drei Kategorien, die in der folgenden Auflistung zusammengefasst sind und im weiteren Verlauf dieses Kapitels behandelt werden.

- Erweiterung des Frameworks zur Unterstützung neuer Service-Kategorien.
- Integration von Werkzeugen durch die Realisierung von Service-Plug-ins.
- Entwicklung und Erweiterung von Framework-Clients.

Im folgenden Abschnitt wird zunächst die Unterstützung neuer Service-Kategorien diskutiert. Anschließend werden die Konzepte, Methodiken und APIs zur Integration von Werkzeugen als Services beschrieben. Danach wird auf die Erweiterungspunkte der Clients eingegangen. In Kapitel 6 wird dann auf Grundlage dieser Abschnitte die Realisierung einiger repräsentativer Services und Clients behandelt.

5.1 Unterstützung neuer Service-Kategorien

Das Framework unterstützt aktuell drei Service-Kategorien mit ihren zugehörigen Backend-Arten. Diese wurden im Rahmen dieser Arbeit nacheinander konzipiert und zur Erweiterung des Frameworks genutzt: Zuerst wurden das Base- und das Ostp-Backend und später das Social-Backend realisiert. Die Entwicklung neuer Backend-Arten ist demzufolge eine Möglichkeit, die Funktionalität des Frameworks zu erweitern und neue Service-Kategorien zu unterstützen.

Die spezifischen Eigenschaften jeder Service-Kategorie und der Kontext, in dem ein Service genutzt wird, wirken sich erheblich auf die Benutzeroberflächen des Frontends und der Clients aus. Beispielsweise werden Base-Services im Kontext von Projekten genutzt und verfügen üblicherweise über eine eigene Web-Oberfläche. Ostp-Services können im Gegensatz dazu unabhängig von einem Projekt genutzt werden. Sie verarbeiten vorwiegend Dateien und basieren daher auf völlig anderen Interaktionskonzepten als Base-Services. Social-Services werden von einzelnen Benutzern für den Abruf von Profilinformationen aus sozialen Netzwerken verwendet. Diese drei Service-Kategorien veranschaulichen die unterschiedlichen Eigenschaften und die daraus resultierenden Anforderungen an eine Integration von Werkzeugen aus jeder Kategorie.

Bei der Integration von Werkzeugen sollte jedoch sowohl die Benutzerfreundlichkeit des gesamten Systems als auch der Funktionsumfang der Werkzeuge nicht beeinträchtigt werden [BB03] (vgl. auch Anforderungen **FA11** und **NFA5**). Im Rahmen dieser Arbeit wurde daher auf die Umsetzung einer generischen Lösung zur Unterstützung neuer Service-Kategorien durch das Framework verzichtet. Diese Entscheidung resultiert auch aus der Tatsache, dass neue Backend-Arten nur relativ selten erstellt werden müssen, da die Service-Kategorien so gewählt worden sind, dass sie jeweils die Integration einer großen Anzahl von Werkzeugen ermöglichen. Insgesamt ergibt sich daraus, dass zur Unterstützung neuer Service-Kategorien das Framework an mehreren Stellen entsprechend erweitert werden muss und eine genauere Kenntnis der Architektur vorausgesetzt wird. Die dafür notwendigen Schritte sind in Abbildung 5.2 in einem Aktivitätsdiagramm zusammengefasst. Die darin gezeigte Methodik wird in diesem Abschnitt skizziert.

Die Erweiterung des Frameworks zur Unterstützung neuer Service-Kategorien erfolgt in zwei Phasen. Zunächst liegt der Fokus auf der Bereitstellung der Infrastruktur für die neue Backend-Art. Daran anschließend treten servicespezifische Aspekte in den Vordergrund. In den folgenden

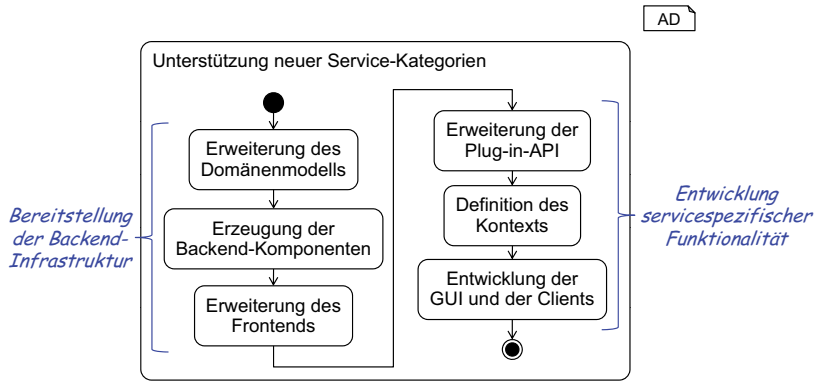


Abbildung 5.2: Methodik zur Unterstützung neuer Service-Kategorien

beiden Abschnitten werden die einzelnen Schritte dieser Phasen jeweils beschrieben. Zur Veranschaulichung der Beschreibung wird dabei in Anlehnung an [vDMC+10] ein Backend für eine webbasierte IDE (Web-IDE) als Beispiel verwendet.

5.1.1 Bereitstellung der Backend-Infrastruktur

In der ersten Phase wird primär die Infrastruktur für die neue Backend-Art durch die Erweiterung des Domänenmodells und in Form von Komponenten ohne servicespezifische Funktionalität bereitgestellt. Zunächst müssen dazu das Domänenmodell um neue Konzepte ergänzt, die Backend-Komponenten erzeugt und zuletzt das Frontend erweitert werden.

Erweiterung des Domänenmodells

Die Komponente SSELab-Domain enthält eine Realisierung des Domänenmodells und wird sowohl in den Backends als auch im Frontend und den Clients verwendet und bildet deren Grundlage (vgl. Abschnitte 4.3.1, 4.4.1 und 4.5.1). Dementsprechend müssen die Domänenklassen zur Unterstützung der neuen Service-Kategorie bzw. der neuen Backend-Art zunächst zu dieser Komponente hinzugefügt werden. Dabei müssen gemäß den Eigenschaften eines white-box Frameworks [FS97] Subklassen existierender Domänenklassen gebildet werden. Die wichtigsten Klassen, die erweitert werden müssen, sind die in Abschnitt 4.4.2 beschriebenen Klassen zur Service-, Plug-in- und Backend-Verwaltung.

Services: Jede Service-Kategorie wird durch eine Subklasse von `Service` repräsentiert. Zur Unterstützung einer neuen Kategorie muss daher eine entsprechend Klasse ergänzt werden, die einerseits grundlegende und andererseits servicespezifische Informationen verwaltet. Erstere müssen bei der initialen Erstellung in dieser Phase der Klasse berücksichtigt werden. Letztere müssen erst in der zweiten Phase ergänzt werden. Zur Umsetzung des Web-IDE-Backends müsste also die neue Klasse `WebIdeService` erstellt werden.

Service-Plug-ins: Für die Repräsentation der Service-Plug-ins der neuen Service-Kategorie wird analog eine entsprechende Subklasse von `ServicePlugin` benötigt (im Beispiel `WebIdeServicePlugin`). Durch diese Klasse können die Plug-ins durch das Frontend unter Verwendung der in Abschnitt 4.4.2 beschriebenen Mechanismen verwaltet werden.

Server-Mappings: Um den Zugriff des Frontends auf Instanzen der neuen Backend-Art zu ermöglichen, muss schließlich noch eine Subklasse von `ServerMapping` erzeugt werden (im Beispiel `WebIdeServerMapping`). Falls zusätzlich zu der Server-URL noch weitere Informationen durch das Server-Mapping gespeichert werden sollen, können diese in der zweiten Phase ergänzt werden.

Erzeugung der Backend-Komponenten

Jede Service-Kategorie wird durch eine eigene Backend-Art unterstützt, so dass im nächsten Schritt auf Grundlage des erweiterten Domänenmodells ein neues Backend erstellt werden muss. Die dabei zu entwickelnden Komponenten sind in Abbildung 5.3 für das Web-IDE-Backend dargestellt.

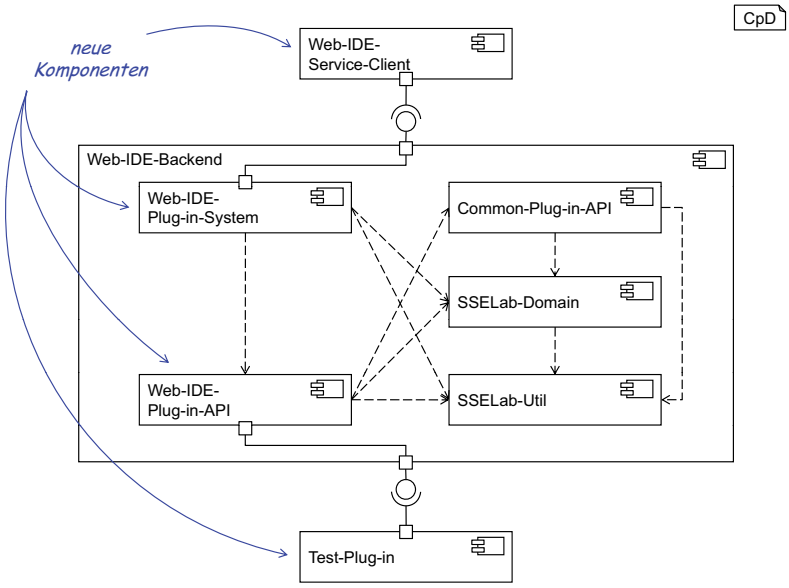


Abbildung 5.3: Neu zu erzeugende Komponenten eines Web-IDE-Backends

Web-IDE-Plug-in-API: Auf Grundlage der Komponente Common-Plug-in-API definiert diese Komponente die Schnittstelle für Service-Plug-ins der neuen Kategorie. Zur Realisierung und Instanziierung des neuen Backends muss in dieser Komponente initial nur eine Klasse

angelegt und eine Methode implementiert werden. Alle weiteren Methoden dieser Klasse, die die Plug-in-API der neuen Backend-Art definieren, können in der zweiten Phase iterativ ergänzt werden.

Web-IDE-Plug-in-System: Durch diese Komponente werden die Plug-ins in einer Backend-Instanz gemäß den in Abschnitt 4.3 beschriebenen Konzepten verwaltet. Als Grundlage der Realisierung dieser Komponente können die Komponenten der existierenden Backend-Arten übernommen und auf Grundlage der Anforderungen der neuen Service-Kategorie angepasst werden.

Web-IDE-Service-Client: Durch diese Komponente wird der Zugriff auf die Schnittstelle der Backend-Instanzen gekapselt. In der Realisierung dieser Komponente auf Basis von Web-Service-Technologien ist ausschließlich generierter Code enthalten, der aus der WSDL-Beschreibung der Komponente Web-IDE-Plug-in-System automatisiert erzeugt werden kann. Weitere Funktionalität muss hier nicht ergänzt werden.

Test-Plug-in: Im Anschluss an die Realisierung der übrigen Komponenten des Backends sollte ein Test-Plug-in aufbauend auf der Plug-in-API entwickelt werden. Dieses Plug-in kann einerseits als Template für zukünftige Plug-ins der Backend-Art und andererseits als Grundlage für Integrationstests genutzt werden.

Nach der Realisierung aller Komponenten kann eine Instanz der neuen Backend-Art gemäß der in Abschnitt 7.3 beschriebenen Methodik gebildet werden.

Erweiterung des Frontends

Nach der Entwicklung des Backends für die neue Service-Kategorie muss das Frontend erweitert werden, damit der Zugriff auf Instanzen des Backends sowie die Verwaltung der Services möglich ist. Abbildung 5.4 zeigt einen Ausschnitt der Architektur des Frontends in einem Komponentendiagramm.

Das Frontend des SSELab-Frameworks und damit auch die Komponente SSELab-Core basiert auf einer Schichtenarchitektur [BMR+96]. Zur Unterstützung einer neuen Service-Kategorie müssen einige der in Abbildung 5.4 gezeigten Komponenten erweitert und einige neu hinzugefügt werden. Im Folgenden werden die einzelnen Schritte zur Erweiterung des Frontends anhand der Schichten skizziert.

1. Das Frontend nutzt eine Datenbank, um die Objekte der Domänenklassen zu persistieren (vgl. Abschnitt 4.4.1). Das Schema der Datenbank wird im Rahmen der Entwicklung des Frameworks explizit modelliert, um beispielsweise eine Schema- und Datenmigration bei Releases neuer Versionen effektiv unterstützen zu können (vgl. Abschnitt 7.3). Damit die Services, Service-Plug-ins und Server-Mappings der neu zu erstellten Service-Kategorie mit Hilfe der Datenbank verwaltet werden können, muss als erster Schritt das Schema entsprechend angepasst werden, so dass Objekte der Klassen auf Tabellen der Datenbank abgebildet werden können.

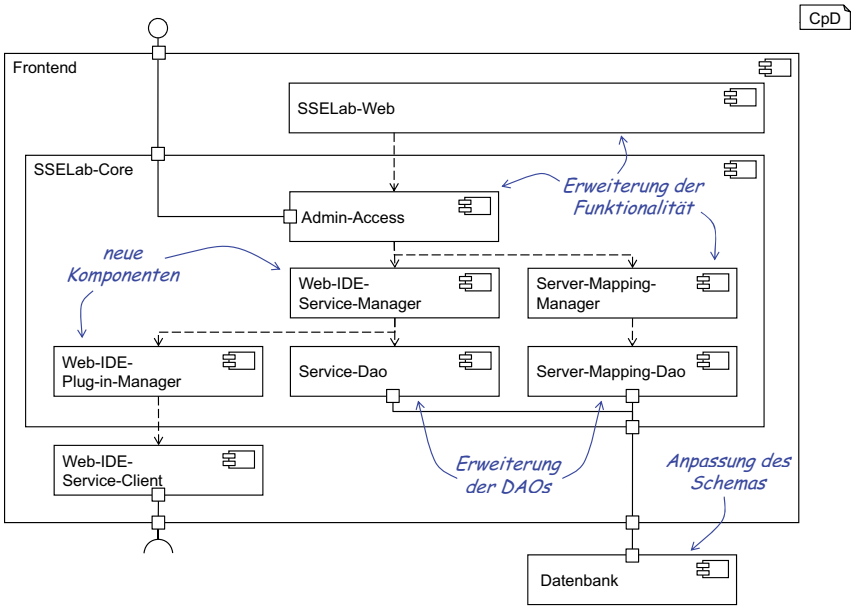


Abbildung 5.4: Erweiterung der Schichtenarchitektur des Frontends

2. Der Zugriff auf die Datenbank wird durch eine Schicht von Data Access Objects (DAOs) [KS06] bzw. Data Mapper Komponenten [Fow03] gekapselt. Für jede logisch zusammenhängende Gruppe von Domänenklassen existiert eine DAO-Komponente, die Funktionalität für deren Verwaltung anbietet. In Abbildung 5.4 sind die zwei DAO-Komponenten für Server-Mappings und Services gezeigt. Diese müssen erweitert werden, so dass die neuen Subklassen des Domänenmodells durch eine intuitiv verwendbare API unterstützt werden.

3. In SSELab-Core kapselt jeweils eine backendspezifische Plug-in-Manager-Komponente die technischen Details für den Zugriff auf alle im Frontend registrierten Instanzen einer Backend-Art. Für das hier verwendete Beispiel muss dementsprechend die Komponente Web-IDE-Plug-in-Manager neu erstellt werden. Darin werden technische Details zur Kommunikation mit den Backend-Instanzen mit Hilfe der bereits beschriebenen Komponente Web-IDE-Service-Client gekapselt. Diese Komponente muss beispielsweise Funktionalität zur Installation, Aktualisierung und Deinstallation von Plug-ins auf Grundlage der Schnittstelle der neuen Backend-Art anbieten. In der zweiten Phase wird hier auch servicespezifische Funktionalität hinzugefügt.

4. Die Kernfunktionalität bzw. die Geschäftslogik des Frontends wird durch mehrere Manager-Komponenten realisiert und den oberen Schichten in Form von grobgranularen APIs zur Verfügung gestellt. Die Service-Manager-Komponenten ermöglichen die Verwal-

tung von Services einer spezifischen Kategorie und verwenden dabei die Komponenten der DAO-Schicht und die jeweilige Plug-in-Manager-Komponente. Im Beispiel aus Abbildung 5.4 nutzt Web-IDE-Service-Manager sowohl die Komponenten Web-IDE-Plug-in-Manager als auch Service-DAO. Für jede neue Service-Kategorie muss daher eine entsprechende Komponente erzeugt werden. Die Komponente Server-Mapping-Manager muss darüber hinaus noch für die Verwaltung von Server-Mappings der neuen Kategorie erweitert werden.

5. Der Zugriff auf die Geschäftslogik der Manager-Komponenten wird durch eine weitere Schicht in rollenspezifischen Fassaden [GHJV95] aggregiert. Diese Fassaden werden einerseits von der Web-Schicht zur Realisierung der Web-Oberfläche und andererseits als Remote Facade [Fow03] für den Zugriff der Framework-Clients über Web-Service genutzt. Über die API der in Abbildung 5.4 gezeigten Komponente Admin-Access werden Methoden für Administratoren angeboten. Diese muss erweitert werden, so dass die Verwaltung von Services der neuen Kategorie durch Administratoren möglich ist.

6. Durch die Komponente SSELab-Web wird die Web-Oberfläche auf Grundlage der Funktionalität von SSELab-Core realisiert. Um die grundlegende Verwaltung von Services der neuen Kategorie über Web-Browser zu ermöglichen, müssen in dieser Komponente die GUI für die Verwaltung von Server-Mappings ergänzt und eine GUI zur Verwaltung von Services und Service-Plug-ins erstellt werden. Dies kann auf Grundlage der bereits existierenden Web-Oberflächen der anderen Service-Kategorien geschehen. Auf Details wird hier nicht weiter eingegangen.

Nach der beschriebenen Erweiterung des Frontends können Instanzen der neuen Backend-Art registriert und Service-Plug-ins installiert werden. Zu diesem Zeitpunkt bieten diese jedoch keine servicespezifische Funktionalität. Die dafür notwendigen Erweiterungen der Komponenten werden im folgenden Abschnitt diskutiert.

5.1.2 Entwicklung servicespezifischer Funktionalität

Das Hinzufügen servicespezifischer Funktionalität ist ein iterativer Prozess, bei dem, ausgehend von der Plug-in-API, Funktionalität in dem neuen Backend und in allen Schichten des Frontends realisiert werden muss. Zusätzlich muss auch der Kontext, in dem Services der neuen Kategorie genutzt werden sollen, definiert sowie die Web-Oberfläche und gegebenenfalls die Framework-Clients erweitert werden. All diese Erweiterungen sind von den Eigenschaften der neuen Service-Kategorie abhängig. Im Folgenden werden einige allgemeine Hilfestellungen in Bezug auf die Erweiterung gegeben.

Erweiterung der Plug-in-API

Im ersten Schritt muss die Plug-in-API um die Methoden ergänzt werden, die die Kernfunktionalität der Services anbieten sollen. Das Design dieser Methoden ist von den Anforderungen und den Interaktionskonzepten der neuen Service-Kategorie abhängig. Einige allgemeingültige Richtlinien für die Entwicklung sind in der folgenden Auflistung zusammengefasst.

- Bei dem Design der Methoden muss eine optimale Abstraktion gefunden werden. Einerseits müssen die Methoden allgemein genug entworfen sein, damit möglichst viele Werkzeuge bzw. Anwendungen durch die neue Service-Kategorie in das Framework integriert werden können. Andererseits sollten auch spezifische Funktionen einzelner Werkzeuge unterstützt werden, damit der volle Funktionsumfang eines Werkzeugs auch nach der Integration erhalten bleibt.

  Da solche Abstraktionen nur effektiv durch die Generalisierung von konkreten Beispielen gefunden werden können [RJ96], sollte eine repräsentative Menge von Werkzeugen der neuen Service-Kategorie ausgewählt, analysiert und darauf aufbauend die API entworfen werden. Die API muss im Anschluss daran im Zuge der Integration weiterer Werkzeuge iterativ verfeinert werden.

- Sobald eine Backend-Instanz und Services der neuen Service-Kategorie produktiv genutzt werden, muss strikt darauf geachtet werden, dass bei einer Verfeinerung der API deren Kompatibilität erhalten bleibt, da alle Schnittstellen der Plug-in-API veröffentlichte Schnittstellen gemäß [Fow02] sind. Dies gilt ebenfalls für die Schnittstelle des Backends.

Nach einer Erweiterung der Plug-in-API müssen die Änderungen in alle Schichten bis in das Frontend nachgezogen werden. Darüber hinaus sollte das Test-Plug-in ausgebaut werden, so dass es die neuen Methoden der API realisiert.

Definition des Kontexts der neuen Service-Kategorie

Die Services des Frameworks werden gemäß Abschnitt 4.1.3 in verschiedenen Kontexten genutzt. Demzufolge muss nach der Erweiterung der Plug-in-API der Kontext der neuen Service-Kategorie definiert werden, damit die Nutzung der neuen Funktionalität im Frontend möglich ist. Das bedeutet, dass im Domänenmodell die Services an geeigneter Stelle referenziert werden müssen. Funktionalität, die nicht in den Domänenklassen selbst umgesetzt werden kann, muss darüber hinaus durch Manager-Komponenten innerhalb des Frontends realisiert werden. Diese Funktionalität muss schließlich noch über die rollenspezifischen Fassaden des Frontends angeboten werden.

Entwicklung der GUI und der Clients

Nachdem die Funktionalität der neuen Service-Kategorie in den Domänenklassen und innerhalb des Frontends realisiert worden ist, können sowohl die Web-Oberfläche als auch die benötigten Framework-Clients realisiert oder erweitert werden. Dies geschieht auf Grundlage der Fassaden von SSELab-Core. Für weitere Details wird hier auf die Dokumentation der im Rahmen dieser Arbeit erzeugten Implementierung des Frameworks verwiesen.

5.1.3 Strategien zur effektiveren Unterstützung neuer Service-Kategorien

Die in diesem Abschnitt beschriebene Methodik zur Unterstützung neuer Service-Kategorien erfordert eine detaillierte Kenntnis der Architektur und der Realisierung des Frontends und der

Domänenklassen. Im Rahmen dieser Arbeit wurde darauf verzichtet, diesen Erweiterungspunkt des Frameworks von einem white-box Ansatz in Richtung eines black-box Ansatzes auszubauen, da es einerseits den Rahmen dieser Arbeit gesprengt hätte und andererseits neue Backend-Arten erfahrungsgemäß nur selten ergänzt werden müssen. Dennoch könnte die Erweiterung des Frontends und der Domänenklassen durch unterschiedliche Strategien vereinfacht werden. In diesem Abschnitt werden zwei mögliche Ansätze genannt.

Eine Möglichkeit, die Erweiterbarkeit zu vereinfachen, wäre der Einsatz von Plug-in-Systemen auch innerhalb des Frontends. In [MW08] und in [KD08] werden beispielsweise die erfolgreiche Integration eines OSGi-Containers in JEE-Application-Server beschrieben, um Teile der Geschäftslogik oder der Web-Oberfläche zur Laufzeit des System erweitern zu können. Ein entsprechender Ansatz könnte auch innerhalb des Frontends genutzt werden, um die Integration neuer Backend-Arten zu vereinfachen. Bei der Realisierung dieses Ansatzes muss jedoch berücksichtigt werden, dass die in [MW08] beschriebene Integration eines OSGi-Containers in einen EJB-Container die EJB-Spezifikation [EJB06] verletzt. Beispielsweise verbietet die Spezifikation eine Manipulation von Classloadern, worauf jedoch der in [MW08] vorgestellte Ansatz basiert. Da jedoch die meisten Application-Server die Regeln der Spezifikation nicht forcieren, ist eine solche Integration trotzdem technisch möglich. Die in [KD08] diskutierte Integration eines OSGi-Frameworks in einen Servlet-Container ist dagegen Konform zur Spezifikation und könnte zur dynamischen Erweiterung der Web-Oberfläche des Frontends eingesetzt werden.

Ein anderer Ansatz zur effektiveren Unterstützung neuer Service-Kategorien wäre eine modellbasierte und generative Entwicklung von Teilen des Frontends, wie sie beispielsweise in [Mar11] auf Grundlage von MontiCore [Kra10] beschrieben wird. Dabei werden aus UML/P-Klassendiagrammen [Rum11, Sch12] unter anderem die Domänenklassen, Data Access Objects als auch das Datenbankschema in Form von SQL-Skripten generiert. Durch die Modellierung wird eine höhere Abstraktionsebene erreicht, so dass auch eine Erweiterung der entsprechenden Frontend-Komponenten erleichtert werden könnte.

5.2 Integration von Services

Als wichtigste Erweiterungsmöglichkeit des SSELab-Frameworks erlauben die Backends eine Integration von Entwicklungswerkzeugen. Für jedes Werkzeug, das in das Framework als Service integriert werden soll, muss ein entsprechendes Service-Plug-in erstellt werden. Die Methodik zur Realisierung von Plug-ins wird in diesem Abschnitt behandelt. Bevor jedoch ein Plug-in realisiert werden kann, muss die Integration des Werkzeugs vorbereitet werden. Die dabei notwendigen Schritte sind wiederum abhängig von der jeweiligen Service-Kategorie.

- Bei der Integration einer webbasierten Anwendung als *Base-Service* müssen unter anderem die benötigten Serverkomponenten installiert, die Konfiguration der Anwendung automatisiert und das Rechtemanagement an die Sicherheitsmechanismen des SSELab-Frameworks adaptiert werden. Außerdem kann die Benutzeroberfläche angepasst werden, um ein einheitliches Look-and-Feel zu ermöglichen.

- Die Integration von Werkzeugen als *Ostp-Services* benötigt eine serverbasierte Installation des Werkzeugs, so dass es durch das Service-Plug-in aufgerufen werden kann. Darüber

hinaus müssen die Parametrisierungsmöglichkeiten des Werkzeugs durch die Realisierung des Plug-ins abgebildet werden, damit der Funktionsumfang des Werkzeugs erhalten bleibt.

- *Social-Services* integrieren Funktionen sozialer Netzwerke. Daher müssen sowohl die Authentifizierung der Benutzer beim Netzwerk als auch der automatisierte Abruf von Profilinformationen über frei zugängliche APIs unterstützt werden.

In den nachfolgenden Abschnitten wird für jede Backend-Art die Methodik zur Integration einer Anwendung bzw. eines Werkzeugs in das Framework diskutiert. Dabei werden sowohl die hier kurz skizzierten Schritte zur Vorbereitung der Integration als auch die Verwendung der jeweiligen Plug-in-API erläutert. Da jedoch alle Backend-Arten auf den gleichen Konzepten basieren und eine einheitliche Architektur besitzen, gibt es auch gemeinsame Aspekte der Plug-in-APIs und damit auch bei der Entwicklung von Service-Plug-ins. Diese Gemeinsamkeiten werden zur Bildung der Grundlage im folgenden Abschnitt zuerst beschrieben.

5.2.1 Gemeinsamkeiten von Service-Plug-ins

Alle Backends des SSELabs definieren ein black-box Framework [FS97] auf Grundlage einer reinen Plug-in-Architektur [Bir05, Rat11]. Die Eigenschaften und Bestandteile solcher Architekturen sowie deren Beziehungen zu Frameworks und Komponenten wurden im Grundlagenkapitel dieser Arbeit in Abschnitt 2.2.3 behandelt. Auf Basis der dort eingeführten Begriffe zeigt Abbildung 5.5 die Rollen der einzelnen Komponenten aller Backend-Arten. In der Abbildung sowie der folgenden Erläuterung wird das Base-Backend zur Veranschaulichung verwendet. Die beschriebenen Konzepte können jedoch auf die anderen Backend-Arten entsprechend übertragen werden.

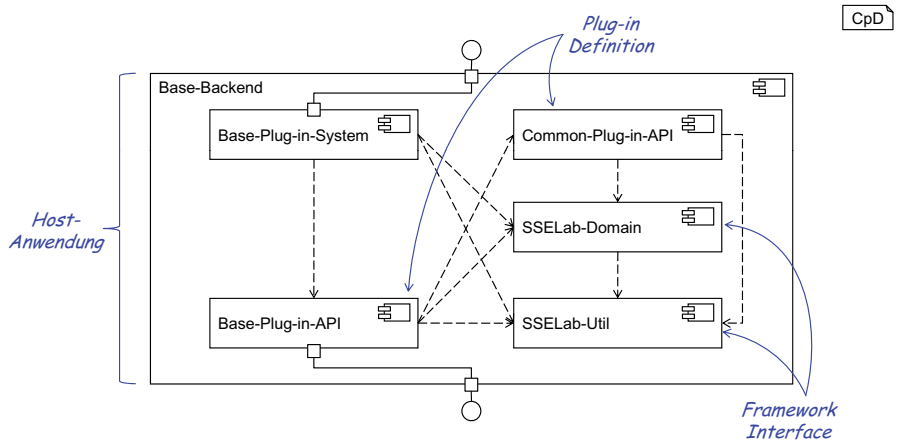


Abbildung 5.5: Rollen der Komponenten des Base-Backends in der Plug-in-Architektur

Das Base-Backend ist die *Host-Anwendung* für Base-Service-Plug-ins und enthält dementsprechend den *Plug-in-Vertrag*. Dieser besteht einerseits aus der *Plug-in Definition*, also den Schnittstellen, die von einem Plug-in realisiert werden müssen, und dem *Framework Interface*, das die von der Host-Anwendung bereitgestellten Dienste und Domänenklassen enthält. Im Fall des in Abbildung 5.5 gezeigten Base-Backends besteht die Plug-in Definition aus den Komponenten Common-Plug-in-API und Base-Plug-in-API. Die Komponenten SSELab-Domain und SSELab-Util bilden das Framework-Interface. Die Komponenten Common-Plug-in-API, SSELab-Domain und SSELab-Util werden in allen Backend-Arten eingesetzt und enthalten daher die APIs, die gemeinsame Aspekte beschreiben. Die Komponente Base-Plug-in-API enthält dagegen die Schnittstellen, die speziell auf die Base-Service-Kategorie zugeschnitten sind.

Bei der Realisierung eines Base-Service-Plug-ins müssen dementsprechend Schnittstellen aus den beiden Komponenten Common-Plug-in-API und Base-Plug-in-API verwendet werden. Gemäß Anforderung **FA52** (Entwicklung von Integrationskomponenten für Werkzeuge) soll die Entwicklung von Service-Plug-ins möglichst einfach sein. Daher wurde bei dem Design der Komponenten auf die Erstellung einer kompakten API geachtet. Das Resultat ist, dass bei der Implementierung eines Service-Plug-ins nur eine Klasse der jeweiligen Plug-in-API erweitert werden muss. Zur Veranschaulichung ist in Abbildung 5.6 die Vererbungshierarchie des Subversion-Plug-ins gezeigt, auf dessen Realisierung genauer in Kapitel 6 eingegangen wird.

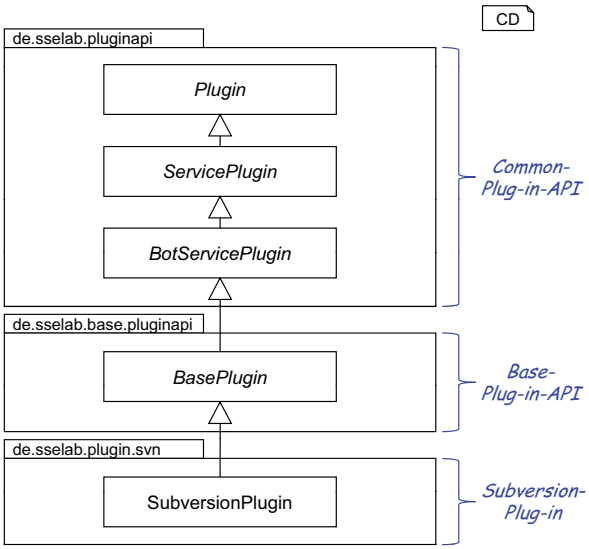


Abbildung 5.6: Vererbungshierarchie des Subversion-Plug-ins

Durch das im Beispiel gezeigt Service-Plug-in wird das Versionsverwaltungssystem Subversion [PCF08, Col02] als Base-Service in das Framework integriert. Dementsprechend ist das Plug-in auf Grundlage der Komponente Base-Plug-in-API realisiert: Die Klasse `Subver-`

sionPlugin erweitert die abstrakte Klasse BasePlugin dieser Komponente, die ihrerseits wiederum Klassen aus Common-Plug-in-API erweitert.

In diesem Abschnitt werden hauptsächlich die Konzepte und Schnittstellen der Komponente Common-Plug-in-API erläutert, die durch das Framework zur Verfügung gestellt wird. Abbildung 5.7 zeigt die wichtigsten Klassen der Komponente und deren Beziehungen zu einigen Interfaces der OSGi-API.

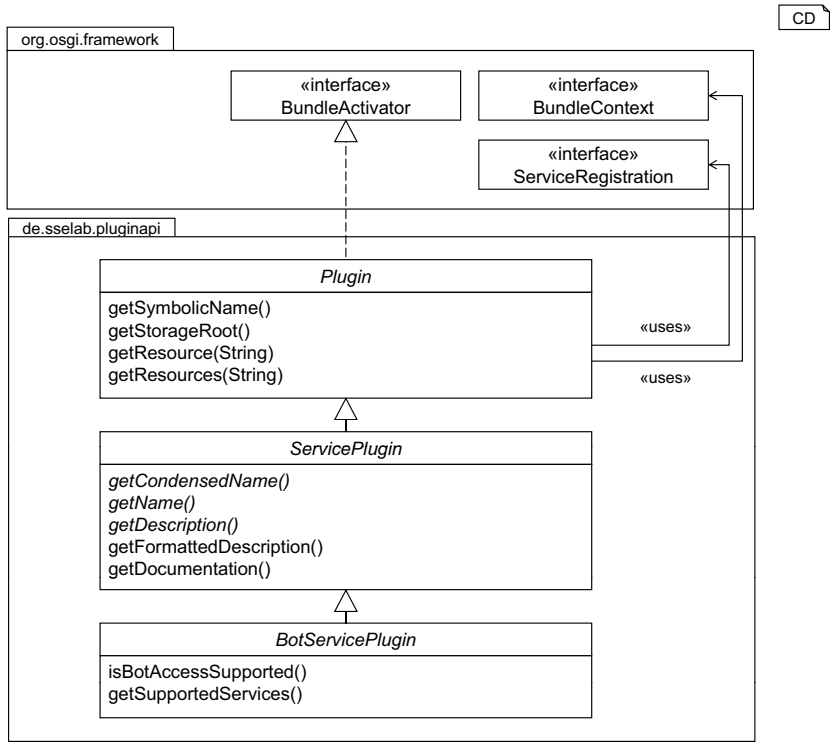


Abbildung 5.7: Klassen der Komponente Common-Plug-in-API

Die in der Abbildung gezeigten Klassen Plugin, ServicePlugin und BotService-Plugin sind die gemeinsamen Oberklassen aller Plug-in-APIs. In diesen Klassen ist einerseits Funktionalität enthalten, die technische Aspekte der Integration von Service-Plug-ins in die Backends generisch löst und kapselt, so dass Service-Entwickler sich auf die eigentliche Integration der Werkzeuge konzentrieren können. Andererseits enthalten diese Klassen grundlegende Methoden, die bei der Implementierung eines Service-Plug-ins entweder realisiert werden müssen oder optional überschrieben werden können. Die Aufgaben dieser drei Klassen lassen sich wie folgt zusammenfassen.

`Plugin`: Diese Klasse enthält Code zur Integration von Plug-ins in das Plug-in-System einer Backend-Instanz und bietet Hilfsmethoden für deren Entwicklung an.

`ServicePlugin`: In dieser Klasse werden Methoden definiert, durch die grundlegende Informationen über ein Service-Plug-in zur Verfügung gestellt werden.

`BotServicePlugin`: Diese Klasse stellt Methoden bereit, mit deren Hilfe die Nutzung von Bot-Accounts für einen Service gesteuert werden können.

Der Funktionsumfang und die Verwendung der APIs dieser Klassen werden in den folgenden drei Abschnitten behandelt.

Integration in ein Plug-in-System

In der im Rahmen dieser Arbeit erzeugten Realisierung aller Backend-Arten werden Plug-in-Systeme auf Grundlage von OSGi [OSG09a] eingesetzt. Die Implementierung der `Plugin`-Klasse kapselt jedoch alle OSGi-spezifischen Implementierungsdetails sowie den benötigten Integrationscode, so dass die Subklassen von `Plugin` keinen OSGi-spezifischen Code mehr enthalten müssen. Die Entwicklung von Service-Plug-ins ist daher ohne genaue Kenntnis der OSGi-Spezifikation und -APIs möglich. Dadurch wird in Übereinstimmung mit Anforderung **FA52** (Entwicklung von Integrationskomponenten für Werkzeuge) der Einarbeitungsaufwand für die Entwicklung von Service-Plug-ins reduziert. Darüber hinaus lassen sich die in dieser Arbeit entwickelten Konzepte auch mit Hilfe anderer Technologien realisieren, so dass bei der Implementierung des Frameworks auf eine Minimierung der Abhängigkeiten auf eine konkrete Technologie – in diesem Fall OSGi – geachtet wurde. Ein Austausch der Plug-in-Technologie sollte daher einfach erfolgen können.

Die Methoden der `Plugin`-Klasse für die Integration von Service-Plug-ins in einen OSGi-Container sind nicht in Abbildung 5.7 gezeigt. Auf deren technischen Details wird hier ebenfalls nicht weiter eingegangen. Bei Bedarf können diese Details der Dokumentation der Realisierung entnommen werden. Zusätzlich zu diesen Methoden werden in der Klasse noch weitere Methoden angeboten, deren Einsatzzweck in der folgenden Auflistung kurz beschrieben wird.

`getSymbolicName()`: Ein Service und das zugehörige Service-Plug-in werden innerhalb aller Backends sowie des Frontends durch einen symbolischen Namen – gegebenenfalls zusammen mit einer Versionsnummer – eindeutig identifiziert. Durch diese Methode wird der symbolischen Namen eines konkreten Plug-ins zurückgegeben.

`getStorageRoot()`: Bei der Konfiguration oder dem Aufruf eines Werkzeugs kann es notwendig sein, temporäre Dateien zu erzeugen und zu verwalten. Diese Methode liefert ein eindeutiges Verzeichnis zurück, in dem entsprechende Dateien von einem Service-Plug-in abgelegt werden können.

`getResource(String)` und `getResources(String)`: Mit Hilfe dieser beiden Methoden können eine bzw. mehrere Dateien aus einem laufenden Plug-in geladen und in das eindeutige Plug-in-Verzeichnis kopiert werden. Durch die Funktionalität dieser Methoden

ist es möglich, dass alle Dateien, die zur Laufzeit eines Service-Plug-ins benötigt werden – wie beispielsweise Konfigurationsskripte oder Dokumentation – im Plug-in selbst mit ausgeliefert und bei Bedarf geladen werden können. Dadurch können Plug-ins als atomare Deployment-Einheiten realisiert werden.

Grundlegende Informationen über Services

Die `ServicePlugin`-Klasse ergänzt die Plug-in-API einerseits um Methoden, durch die einige wesentliche Informationen von Service-Plug-ins und damit der zugehörigen Services definiert werden. Diese Informationen werden für Services aller Kategorien benötigt und sind damit Bestandteil der gemeinsam genutzten Komponente Common-Plug-in-API. Andererseits bietet die Klasse die Möglichkeit, reichhaltige Dokumentation eines Services in Form von HTML-Seiten bereitzustellen. Die Konzepte der API dieser Klasse werden im Folgenden kurz erläutert.

`getCondensedName()`: Da der symbolische Name eines Service-Plug-ins relativ lang sein kann, wird über diese Methode eine Kurzfassung des symbolischen Namens erzeugt. Diese Kurzfassung wird beispielsweise zum Aufbau von URLs genutzt, die auf die zugehörigen Service-Instanzen zeigen.

`getName()`: Der symbolische Name und seine Kurzfassung sind primär für die Verwendung innerhalb des Frameworks gedacht, beispielsweise für die Identifikation von Services oder den Aufbau von URLs. Im Gegensatz dazu ist der Name, der über diese Methode zurück gegeben wird, für die Benutzer sichtbar. Diese Methode muss in jedem Service-Plug-in implementiert werden. Dabei sollte der gängige Name des Werkzeugs zurückgegeben werden, das als Service integriert wird.

`getDescription()` und `getFormattedDescription()`: Diese beiden Methoden geben die Beschreibungen eines Services zurück. Durch die Realisierung der ersten Methode muss in jedem Service-Plug-in eine kompakte Beschreibung des Services zurückgegeben werden. Durch die zweite Methode – deren Implementierung optional ist – kann eine detailliertere Beschreibung angegeben werden.

`getDocumentation()`: Diese Methode kann überschrieben werden, um die Dokumentation eines Services in Form statischer Webseiten bereitzustellen. Die Realisierung der Methode muss mindestens eine Datei mit dem Namen `index.html` zurückgeben. Zusätzlich können beliebig viele Dateien, beispielsweise weitere HTML-Dateien oder Bilder, Bestandteil der Dokumentation sein. Eine Implementierung dieser Methode ist nicht zwingend erforderlich. Es ist jedoch empfehlenswert, diese Methode zu überschreiben und Dokumentation für ein Service-Plug-in bereitzustellen.

Verwendung von Bot-Accounts

Die Klasse `BotServicePlugin` enthält die im Folgenden erläuterte API, durch die die Verwendung von Bot-Accounts für einen Service gesteuert werden kann. Diese Klasse unterstützt daher die in Abschnitt 4.4.3 beschriebene Service-Service-Interaktion.

`isBotAccessSupported()`: Diese Methode gibt an, ob der Zugriff durch Bot-Accounts auf den zugehörigen Service überhaupt erlaubt werden soll. Standardmäßig ist der Zugriff durch Bot-Accounts abgeschaltet. Falls der Zugriff durch Bot-Accounts für einen konkreten Service unterstützt werden soll, muss in dem zugehörigen Service-Plug-in diese Methode überschrieben werden.

`getSupportedServices()`: Mit Hilfe dieser Methode kann ein Service-Plug-in definieren, auf welche anderen Services es mit Hilfe von Bot-Accounts zugreifen möchte. Dabei wird eine Sammlung symbolischer Namen zurückgegeben. Durch die Verwendung des symbolischen Namens wird hier eine lose Kopplung erreicht, so dass ein Service-Plug-in in einer entsprechenden Backend-Instanz installiert werden kann, in der die unterstützen Services nicht verfügbar sind.

Ein Service kann nur dann mit Hilfe von Bot-Accounts auf einen anderen Service zugreifen, wenn der Ziel-Service den passenden symbolischen Namen besitzt und den Zugriff durch Bot-Accounts grundsätzlich erlaubt. Abbildung 5.8 illustriert diesen Sachverhalt anhand des CI- und des Subversion-Plug-ins. Der durch das CI-Plug-in konfigurierte Continuous Integration Server soll auf Subversion-Repositories mit Hilfe eines Bot-Accounts zugreifen können. Daher wird der symbolische Name des Subversion-Plug-ins in der Realisierung des CI-Plug-ins angegeben. Zusätzlich erlaubt das Subversion-Plug-in grundsätzlich den Zugriff durch Bot-Accounts.

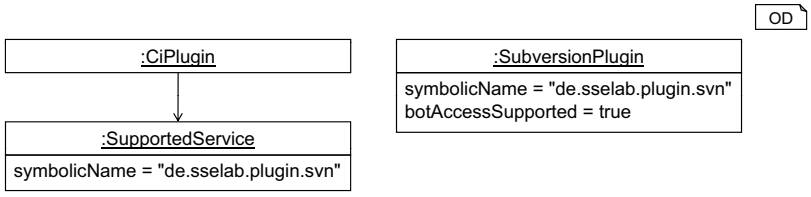


Abbildung 5.8: Konfiguration für den Zugriff auf Subversion-Repositories durch den CI-Server mit Hilfe von Bot-Accounts

Durch die Abbildung wird auch die lose Kopplung zwischen den Plug-ins illustriert. Die beiden Objekte sind über keinen direkten Link assoziiert, so dass die zugehörigen Plug-ins unabhängig voneinander genutzt werden können. Erst wenn beide gleichzeitig in einer Backend-Instanz installiert sind, wird die Konfiguration der Verbindung über Bot-Accounts ausgewertet.

5.2.2 Integration von Base-Services

Durch Base-Services werden server- bzw. webbasierte Anwendungen über eine Framework-Instanz zur Verfügung gestellt. Die Services dieser Kategorie können ausschließlich im Kontext von Projekten genutzt werden und bilden deren Kerninfrastruktur. Die Methodik zur Integration solcher Anwendungen in das Framework wird durch das Aktivitätsdiagramm in Abbildung 5.9 illustriert. Die einzelnen Schritte werden nachfolgend genauer behandelt.

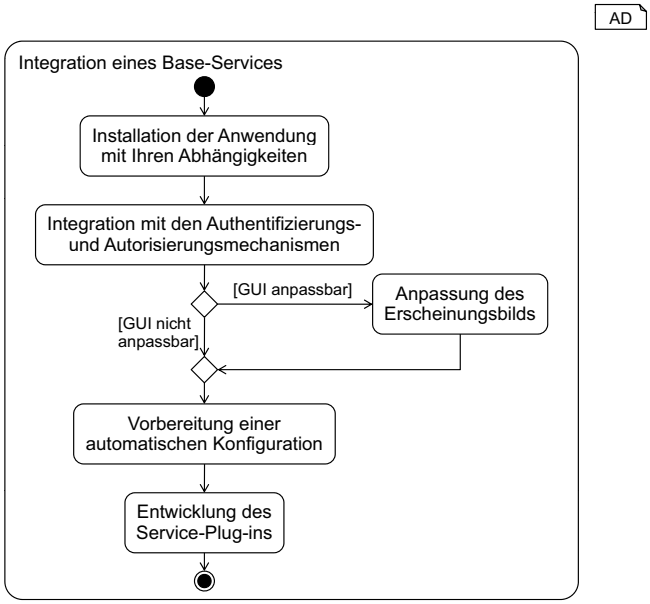


Abbildung 5.9: Methodik zur Integration von Anwendungen als Base-Services

Installation der Anwendung mit ihren Abhängigkeiten. Der erste Schritt für die Integration einer Anwendung als Base-Service ist die Installation der Software mit ihren Abhängigkeiten und der gesamten Laufzeitumgebung. Alle Schritte der Installation sollten durch Skripte automatisiert werden, damit beispielsweise neue Base-Backend-Instanzen mit geringem Aufwand erzeugt werden können. Insbesondere muss dabei darauf geachtet werden, dass ein wiederholtes Ausführen der Skripte nicht zu Fehlern führt.

Integration mit den Authentifizierungs- und Autorisierungsmechanismen. Bei der Integration einer Anwendung als Base-Service muss gemäß Anforderung **FA33** darauf geachtet werden, dass sich Benutzer nur einmalig bei einer Framework-Instanz anmelden müssen, also Single-Sign-On (SSO) unterstützt wird. Die dafür benötigten Benutzerinformationen werden über das LDAP-Verzeichnis des Frontends angeboten und können entsprechend abgefragt werden. Für die Integration einer Anwendung muss diese im zweiten Schritt daher so konfiguriert werden, dass die LDAP-Informationen zur Authentifizierung und Autorisierung genutzt werden. Bei der Autorisierung muss insbesondere darauf geachtet werden, dass ausschließlich die Teilnehmer eines Projekts Zugriff auf die entsprechenden Seiten der Anwendung besitzen und keine anderen Benutzer der Framework-Instanz.

Anpassung des Erscheinungsbilds. Viele webbasierte Anwendungen erlauben es, dass Erscheinungsbild ihrer Instanzen zu ändern. Beispielsweise durch das Ändern von Grafiken

oder durch das Hinzufügen eigenständiger Skins. In einem dritten Schritt sollte die Anpassung des Erscheinungsbilds durchgeführt werden, um ein einheitliche Benutzeroberfläche über die Grenzen der integrierten Anwendungen hinweg zu ermöglichen, also um Presentation Integration gemäß [Was90] zu erreichen.

Vorbereitung einer automatischen Konfiguration. Die Konfiguration von Instanzen der Anwendung erfolgt nach der Integration vollständig automatisiert durch das zugehörige Service-Plug-in. Die dafür verwendbaren Schnittstellen und Konfigurationsdateien der Anwendung müssen in diesem vorletzten Schritt identifiziert werden. Manche Anwendungen bieten ausschließlich eine dateibasierte Konfiguration an, andere stellen REST-Schnittstellen oder eigene Client-Komponenten zur Verfügung. Unabhängig von der Art muss das Hinzufügen und Entfernen von Benutzern zu den projektspezifischen Instanzen der Anwendung möglich gemacht werden. Auch die Erstellung und Deaktivierung von Instanzen muss hier unterstützt werden.

Die Integration einer Anwendung als Base-Services wird schließlich durch die Entwicklung eines Service-Plug-ins abgeschlossen. Grundlage der Realisierung bilden die Komponenten Common-Plug-in-API und Base-Plug-in-API, deren wichtigste Klassen mit ihren Beziehungen in Abbildung 5.10 dargestellt sind.

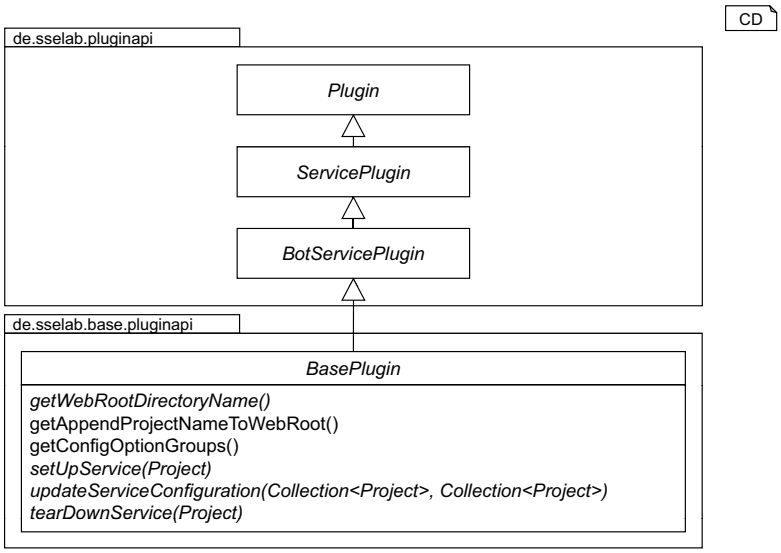


Abbildung 5.10: Klassen der Komponenten Base-Plug-in-API und Common-Plug-in-API

Die Konzepte der Klassen aus Common-Plug-in-API wurden bereits im vorigen Abschnitt 5.2.1 behandelt. Darauf aufbauend definiert die Komponente Base-Plug-in-API die Schnittstelle

für Base-Services. Die Methoden der Klasse `BasePlugin` ermöglichen daher die Integration und Konfiguration von Base-Services und werden im Folgenden beschrieben.

Bildung eindeutiger URLs

Gemäß Anforderung **NFA3** darf ein Benutzer nur Zugriff auf seine Projekte und die assoziierten Services besitzen. Für die Umsetzung eines geeigneten Autorisierungsmechanismus werden in einer Framework-Instanz eindeutige URLs für Base-Service-Instanzen gebildet. Der grundsätzliche Aufbau von URLs ergibt sich dabei aus der Kombination der Domäne mit einem eindeutigen Pfad. In Quellcode 5.11 ist der Aufbau des Pfads in einer vereinfachten EBNF-Grammatik [ISO96] definiert.

```
1  urlPath =
2    "/", serverName,
3    "/", serviceVisibility,
4    "/", serviceWebRoot,
5    ["/", projectName] "/"];
6
7  serverName = "lab", digit, { digit } ;
8  serviceVisibility = "private" | "public" ;
9  serviceWebRoot = nonWSChar , { nonWSChar } ;
10 projectName = ? valid project name ? ;
11
12 digit = ? all digits ? ;
13 nonWSChar = ? all non white space characters ?
14
15 (*
16 Beispiel: Pfad zum Subversion-Repository des MontiCore-Projekts
17 /lab2/private/svn/MontiCore/
18 *)
```

Quellcode 5.11: Aufbau des Pfads von Service-URLs

Die URL zu einer Instanz eines Base-Services wird auf Grundlage von Informationen des Services selbst sowie des Projekts und der beteiligten Base-Backend-Instanz gebildet. Jedem Projekt ist eine Base-Backend-Instanz über ein Server-Mapping zugeordnet (vgl. Abschnitt 4.4.4). Der Servername ist Bestandteil des Server-Mappings und wird zur Identifikation der Base-Backend-Instanz als erster Teil des Pfads verwendet. Im Beispiel in QuellCode 5.11 wird der Servername `lab2` verwendet. Das nachfolgende Pfadsegment bestimmt, ob es sich um eine projektinterne (`private`) oder um eine öffentlich zugängliche (`public`) Service-Instanz handelt. Danach folgt der eindeutige Name des Wurzelverzeichnisses der Anwendung (im Beispiel `svn`) und optional der Name des Projekts. Ob der Name des Projekts zur Bildung der URL verwendet wird, ist abhängig davon, wie die Anwendung instanziiert wird. In Bezug auf die Instanziierung lassen sich die folgenden zwei Kategorien von Anwendungen unterscheiden.

- Server- oder webbasierte Anwendungen, die mehrmals instanziiert werden können, gehören zur ersten Kategorie. Ein Beispiel für eine Anwendung in dieser Kategorie ist Subversion, da es Instanziierung mehrerer unabhängiger Repositories erlaubt. Die Autorisierung

für den Zugriff auf eine Instanz erfolgt in diesem Fall über den eindeutigen Projektnamen in der URL.

- Bei Anwendungen, die nur eine Instanz erlauben, kann die Autorisierung von Benutzern ausschließlich über das anwendungsspezifische Rechtemanagement erfolgen, so dass der Projektname nicht Bestandteil der URL ist. Ein Beispiel für eine Anwendung dieser Kategorie ist Jenkins.

Damit das Framework die URLs für Service-Instanzen beider Kategorien bilden kann, sind die beiden Methoden `getWebRootDirectoryName()` und `getAppendProjectNameToWebRoot()` Bestandteil der Base-Plug-in-API. Über die erste Methode muss ein Service-Plugin den Namen des Wurzelverzeichnisses der Anwendung zurückgeben. Durch Überschreiben der zweiten Methode kann zusätzlich noch festgelegt werden, ob der Projektname beim Aufbau der URL berücksichtigt werden soll. Standardmäßig sind diese Methoden für Anwendungen der ersten Kategorie ausgelegt.

Konfigurationsoptionen von Base-Services

Server- und webbasierte Anwendungen bieten üblicherweise vielfältige Konfigurationsmöglichkeiten, die beispielsweise über Konfigurationsdateien oder -skripte, über die graphische Benutzeroberfläche oder über REST-Schnittstellen verändert werden können. Bei der Integration von Anwendungen sollten die Konfigurationsmöglichkeiten, die für die Benutzer der Anwendung relevant sind, gemäß Anforderung **FA11** erhalten bleiben. In einigen Fällen ist dies jedoch nicht ohne Weiteres möglich. Beispielsweise bieten manche Anwendungen die Änderung der Konfiguration ausschließlich über Dateien an. Diese können die Benutzer einer Framework-Instanz nicht verwenden, da sie keinen Zugriff auf das Dateisystem der Servermaschinen erhalten. Für diese Fälle ermöglicht die Base-Plug-in-API die Definition von Konfigurationsoptionen mit Hilfe der Methode `getConfigOptionGroups()`, die den Benutzern über das Frontend zugänglich gemacht und innerhalb des Service-Plug-ins auf die konkrete Konfiguration der Anwendung abgebildet werden können. Die Domänenklassen, die für die Definition dieser Konfigurationsoptionen verwendet werden, sind in Abbildung 5.12 gezeigt.

Mehrere Konfigurationsoptionen können mit Hilfe der Klasse `ConfigOptionGroup` gruppiert werden. Jede Gruppe besitzt einen für den Service eindeutigen symbolischen Namen zur Identifizierung, ist benannt und referenziert eine Menge von `ConfigOption`-Objekten, die jeweils eine einzelne Option repräsentieren. Jede Option verfügt dabei über die folgenden Eigenschaften.

`symbolicName`, `name` und `description`: Der symbolische Name einer Option dient ebenfalls der Identifikation und muss daher im Kontext eines Services eindeutig gewählt werden. Darüber hinaus besitzt eine Option einen Namen und eine Beschreibung. Der Name sollte möglichst kompakt gewählt werden und die Option präzise beschreiben.

`optionType`: Dieses Attribut definiert, ob es sich bei der Option um eine projektweite oder um eine benutzerspezifische Konfigurationsoption handelt. Projektweite Optionen können

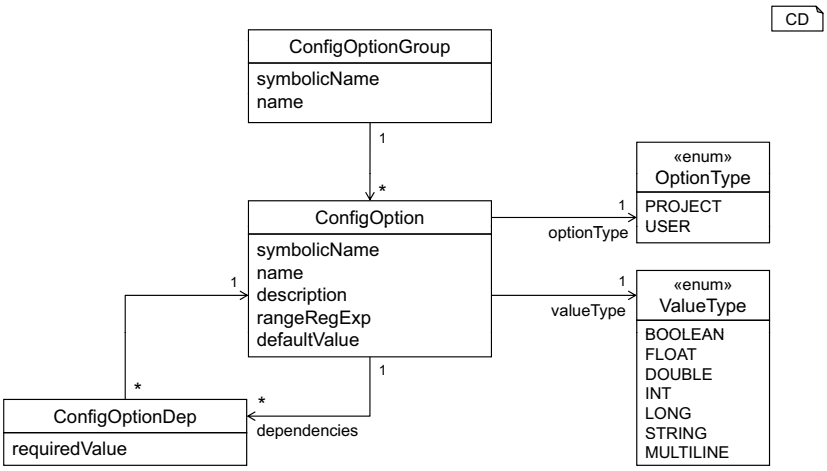


Abbildung 5.12: Klassen zur Definition von Konfigurationsoptionen eines Base-Services

nur durch die Manager eines Projekts angepasst werden. Benutzerspezifische Optionen ermöglichen es jedem Benutzer individuell, das Verhalten eines Services zu bestimmen.

`valueType`: Dieses Attribut definiert den Typ einer Konfigurationsoption. Der Typ wird in der Web-Oberfläche des Frontends genutzt, um die benötigten GUI-Elemente zu erzeugen und die Verarbeitung der übergebenen Werte zu ermöglichen.

`rangeRegExp`: Für eine noch genauere Einschränkung des Wertebereichs einer Konfigurationsoption kann ein regulärer Ausdruck angegeben werden. Bei der Veränderung des Wertes einer Option durch einen Benutzer wird der reguläre Ausdruck zur Validierung des neuen Wertes genutzt.

Die Verwendung eines regulären Ausdrucks ist optional. Bei Konfigurationsoptionen mit komplexen Wertebereichen kann es sinnvoll sein, keinen regulären Ausdruck zu verwenden, sondern einen Parse-Algorithmus im Service-Plug-in zu realisieren. In diesem Fall würde die Validierung bei Werteänderungen durch einen Benutzer erst im Plug-in in der Methode `updateServiceConfiguration()` erfolgen können.

`defaultValue`: Mit Hilfe dieses Attributs muss der Default-Wert der Konfigurationsoption angegeben werden, der initial bei der Aktivierung des Services für ein Projekt verwendet wird.

Abhängigkeiten zwischen einzelnen Konfigurationsoptionen können mit Hilfe der Klasse `ConfigOptionDep` modelliert werden. Über das Attribut `requiredValue` der Klasse wird angegeben, welcher Wert in einer Option gesetzt sein muss, damit die abhängige Option überhaupt anwendbar ist.

Instanziierung und Konfiguration von Base-Services

Die eigentliche Instanziierung und Konfiguration der integrierten Anwendung erfolgt durch Aufrufe der Methoden `setUpService()`, `updateServiceConfiguration()` bzw. `tearDownService()`. Die Prinzipien, die bei deren Implementierung der Methoden berücksichtigt werden müssen, werden im Folgenden diskutiert.

`setUpService(Project)`: Diese Methode wird vom Frontend aufgerufen, um die Anwendung für das übergebene Projekt zu instanziieren und initial zu konfigurieren. Der Aufruf erfolgt dabei entweder nach der Aktivierung eines Projekts, das den zugehörigen Service enthält oder sobald der Service für ein aktiviertes Projekt ausgewählt wird. Nach der Ausführung dieser Methode wird direkt im Anschluss die Methode `updateServiceConfiguration()` vom Frontend aufgerufen, so dass in dieser Methode ausschließlich die Schritte ausgeführt werden sollten, um die Anwendung für das Projekt erstmalig zu instanziieren.

`updateServiceConfiguration(...)`: Viele Änderungen an einem Projekt oder dessen Mitglieder erfordern eine Aktualisierung der Service-Konfiguration. Ein Beispiel dafür ist das Hinzufügen zu oder Entfernen eines Benutzers aus einem Projekt. Dabei müssen alle Services des Projekts aktualisiert werden, um dem Benutzer den Zugriff zu ermöglichen bzw. zu entziehen. Ein weiteres Beispiel ist die Änderung der primären E-Mail-Adresse eines Benutzers. In diesem Fall müssen alle Services in den Projekten dieses Benutzers aktualisiert werden, die die E-Mail-Adresse verwenden.

Im Framework sind gemäß den Eigenschaften von Plug-in-Architekturen keine konkreten Service-Plug-ins bekannt. Daher sind auch keine Informationen darüber verfügbar, welche Änderungen für eine konkrete Anwendung relevant sind. Dementsprechend wird ein Service-Plug-in bei jeder Änderung im Frontend über einen Aufruf der `updateServiceConfiguration`-Methode benachrichtigt, dass eine Änderung stattgefunden hat. Ob die jeweilige Änderung für das Plug-in relevant ist, muss dann in der Implementierung der Methode entschieden werden. Um dies zu ermöglichen, werden der Methode beim Aufruf alle Projekte übergeben: einmal in dem Zustand vor der Änderung, die zu dem Aufruf der Methode führte und einmal in dem Zustand nach der erfolgten Änderung. Durch den Vergleich der Zustände der übergebenen Projekte kann die konkrete Änderung ermittelt und dann die Konfiguration der Anwendung entsprechend aktualisiert werden, falls dies nötig ist. Für die Ermittlung der Änderungen wird in der Komponente SSELab-Domain entsprechende Funktionalität angeboten.

`tearDownService(Project)`: Diese Methode wird vom Frontend aufgerufen, wenn ein Projekt mit dem zugehörigen Service oder der Service selbst für das Projekt deaktiviert wird. Durch die Realisierung dieser Methode muss den Benutzern der Zugriff auf den Service entzogen werden. Die Daten, die von der zugehörigen Anwendung gespeichert werden, dürfen dabei nicht gelöscht werden, so dass nach einer erneuten Aktivierung noch alle Daten vorhanden sind.

Die Implementierung dieser Methoden in den konkreten Service-Plug-ins variiert erheblich, umfasst aber in vielen Fällen die Erzeugung von Verzeichnissen im Dateisystem, die Ausführung von Skripten, die Initialisierung von Datenbanktabellen oder den Zugriff auf eine Web-Service- oder eine REST-Schnittstelle der zu konfigurierenden Anwendung. Um die Implementierung von Service-Plug-ins zu vereinfachen, werden einige Hilfsklassen für Dateioperationen, Prozessausführung, HTTP-Anfragen und Manipulation von Konfigurationsdateien in den Komponenten SSELab-Util und Base-Plug-in-API angeboten. Auf diese Klassen wird in diesem Kapitel nicht weiter eingegangen. In Kapitel 6 wird jedoch anhand einiger konkreter Service-Plug-ins die Verwendung dieser Hilfs-APIs behandelt.

Bei der Realisierung der Methoden sollte einerseits darauf geachtet werden, dass die Konfiguration der Anwendung nur dann angepasst wird, wenn es aufgrund einer Änderung tatsächlich notwendig ist. Andererseits sollte bei aufeinander folgenden Aufrufen mit identischen Parametern nur der erste Aufruf zu einer Aktualisierung der Konfiguration führen. Durch eine Beachtung dieser Prinzipien durch einen Entwickler wird eine effiziente Implementierung sichergestellt, indem beispielsweise unnötige Dateioperationen oder Prozessaufrufe vermieden werden.

5.2.3 Integration von Ostp-Services

Durch Ostp-Services werden dateiverarbeitende Werkzeuge in das Framework integriert. Im Gegensatz zu Base-Services speichern diese keine projektspezifischen Daten und müssen dementsprechend nicht explizit für ein Projekt konfiguriert werden, können jedoch im Kontext von Projekten genutzt werden. Ihr Verhalten ist dabei unabhängig davon, ob die Nutzung im Kontext eines Projekts erfolgt oder nicht. Die Integration solcher Werkzeuge als Ostp-Services wird ebenfalls in mehreren Schritten durchgeführt. In Abbildung 5.13 ist ein Aktivitätsdiagramm als Überblick der Methodik gezeigt.

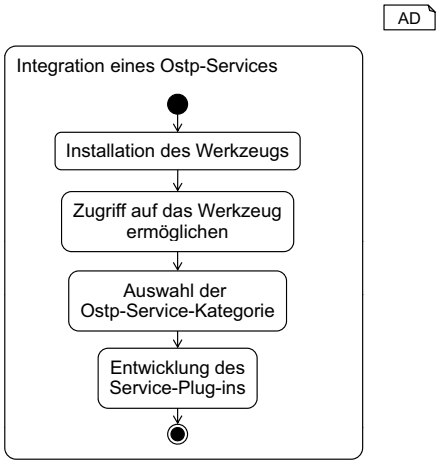


Abbildung 5.13: Methodik zur Integration von Werkzeugen als Ostp-Services

Installation des Werkzeugs. Als erster Schritt muss das zu integrierenden Werkzeug auf einem Server installiert werden. Dabei gibt es mehrere Varianten. Beispielsweise kann das Werkzeug über eine eigene Servermaschine bereitgestellt werden. Eine andere Möglichkeit ist eine Installation auf der Maschine der Backend-Instanz oder aber die Installation innerhalb der Backends als Plug-in. Welche Variante geeignet ist, muss für jedes Werkzeugs individuell entscheiden werden. Die Installation sollte in jedem Fall automatisiert werden.

Zugriff auf das Werkzeug ermöglichen. Je nach Installationsvariante muss im zweiten Schritt der Zugriff auf das Werkzeug ermöglicht werden. Bei der Installation auf einer eigenen Servermaschine kann der Zugriff beispielsweise über Web-Service oder SSH erfolgen. Die entsprechenden Mechanismen müssen hier bereitgestellt werden. Bei einer Installation auf der Maschine der Backend-Instanz oder innerhalb des Plug-in-Systems selbst kann der Zugriff über das Betriebssystem bzw. über Aufrufe einer API erfolgen.

Im Anschluss muss eine der vier verfügbaren Kategorien für den neu zu erstellenden Ostp-Service ausgewählt werden. Ausschlaggebend ist dabei das Ein- und Ausgabeverhalten des Werkzeugs. Abbildung 5.14 veranschaulicht das charakteristische Verhalten von Werkzeugen der vier Kategorien.

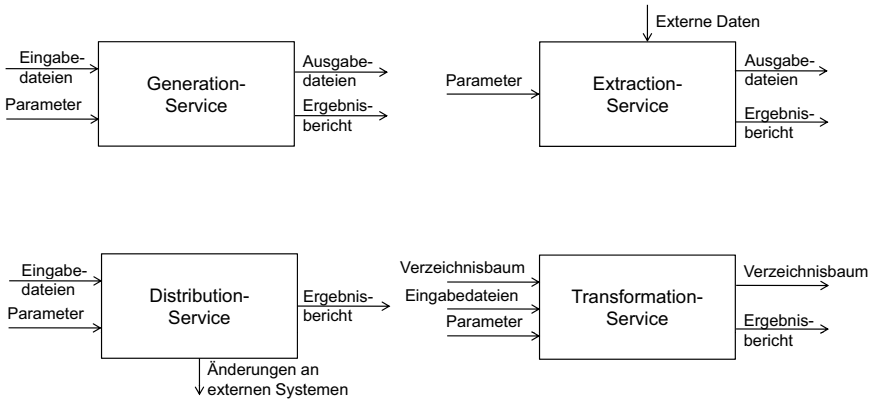


Abbildung 5.14: Kategorien von Ostp-Services mit ihrem charakteristischen Ein- und Ausgabeverhalten

Generation-Service: Die Werkzeuge dieser Kategorie sind dadurch charakterisiert, dass sie eine Menge beliebiger Eingabedateien verarbeiten und eine Menge von Ausgabedateien erzeugen. In welcher Beziehung die Eingabe- und Ausgabedateien zueinander stehen, ist dabei nicht relevant.

Extraction-Service: Diese Werkzeuge beziehen die zu verarbeitenden Eingabedaten nicht aus Dateien, die vom Benutzer angegeben worden sind, sondern extrahieren sie aus externen Systemen, beispielsweise aus Web-Systemen oder durch Aufrufe von Web-Services.

Distribution-Service: Als Distribution-Services werden Werkzeuge bezeichnet, die mit Hilfe der übergebenen Eingabedateien Änderungen an externen Systemen vornehmen.

Transformation-Service: Werkzeuge dieser Kategorie sind dadurch charakterisiert, dass die Eingabe- und Ausgabedateien im Kontext einer Verzeichnishierarchie verarbeitet werden. Die Eingabe ist daher eine Teilmenge von Dateien einer Verzeichnishierarchie. Die Ausgabe ist die entsprechend veränderte Hierarchie. Dadurch kann beispielsweise ein vollständiges Source-Code-Projekt einer IDE wie Eclipse durch einen entsprechenden Ostp-Service verarbeitet werden.

Prinzipiell könnten die Werkzeuge der Kategorien Extraction- und Distribution-Service auch durch einen Generation-Service repräsentiert werden. Das Ein- und Ausgabeverhalten eines Werkzeugs hat jedoch Auswirkungen auf die Benutzeroberfläche, die zur Ausführung des zugehörigen Ostp-Services angeboten wird. So ist es bei einem Distribution-Service beispielsweise nicht notwendig, ein Ausgabeverzeichnis vom Benutzer angeben zu lassen, da als Ergebnis nur ein Bericht ausgegeben wird. Aus diesem Grund wurden die beiden Kategorien Extraction- und Distribution-Service zusätzlich definiert.

Die Komponente Ostp-Plug-in-API ermöglicht die Realisierung von Ostp-Services aller vier Kategorien. Die Konzepte und die Verwendung der API werden im Folgenden auf Grundlage des Klassendiagramms aus Abbildung 5.15 vorgestellt.

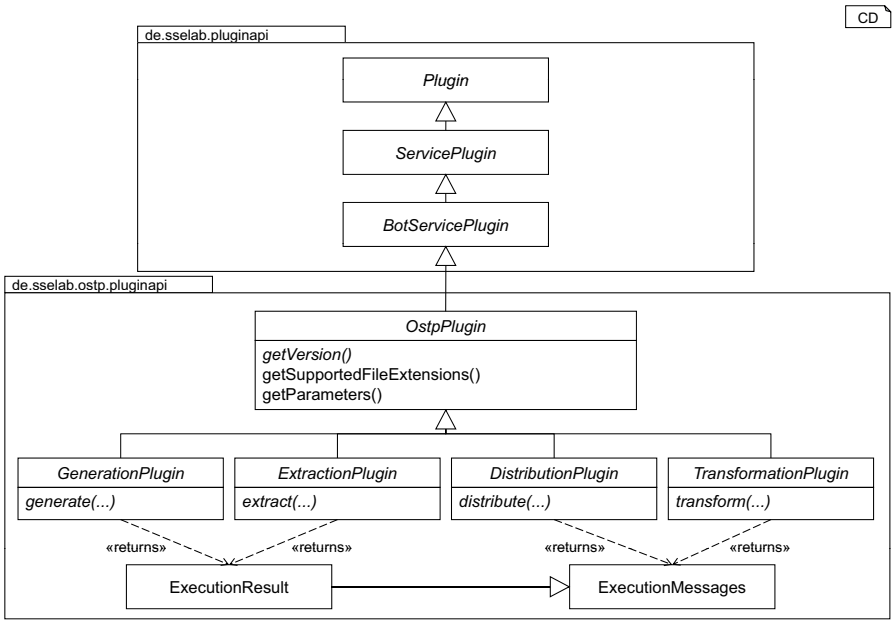


Abbildung 5.15: Klassen der Komponenten Ostp-Plug-in API und Common-Plug-in-API

Die Klasse `OstpPlugin` ergänzt die API der Komponente Common-Plug-in-API um die folgenden Methoden zur Realisierung von Ostp-Service-Plug-ins.

`getVersion()`: Im Gegensatz zu den Base-Services kann ein Ostp-Service gleichzeitig in mehreren Versionen in einer Backend-Instanz installiert sein. Für jede Version wird dabei ein eigenes Service-Plug-in benötigt. Die Version des Services muss entsprechend durch diese Methode zurückgegeben werden.

`getSupportedFileExtensions()`: Diese Methode definiert die vom Service unterstützten Eingabeformate, indem durch sie eine Liste der zugehörigen Dateinamenserweiterungen zurückgegeben wird. Die Standardimplementierung der Methode liefert keine Erweiterungen zurück, so dass sie beispielsweise für einen Extraction-Service – der keine Eingabedateien verarbeitet – nicht überschrieben werden muss.

Parametrisierung von Ostp-Services

Durch Anforderung **FA11** wird gefordert, dass die Parametrisierungsmöglichkeiten eines Werkzeugs auch nach dessen Integration in das Framework erhalten bleiben sollen. Zur Umsetzung dieser Anforderung können Parameter auf Grundlage des in Abbildung 5.16 gezeigten Domänenmodells definiert werden. In einem Service-Plug-in können diese Parameter dann über die die Methode `getParameters()` zurückgegeben werden.

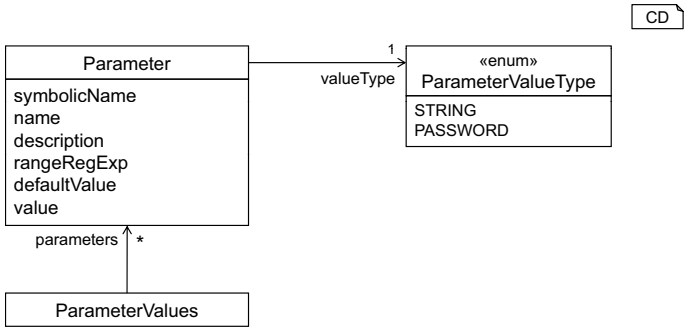


Abbildung 5.16: Klassen zur Definition der Parameter eines Ostp-Services

Die Klasse `Parameter` wird sowohl für die Definition von Parametern als auch für die Übergabe konkreter Parameterwerte verwendet. Die zugrunde liegenden Konzepte sind aus Konsistenzgründen verwandt mit den im vorigen Abschnitt 5.2.2 vorgestellten Konfigurationsoptionen für Base-Services.

`symbolicName`, `name` und `description`: Der symbolische Name dient zur internen Identifikation von Parametern. Der Name und die Beschreibung werden dagegen dem Benutzer angezeigt. Die Beschreibung sollte so ausführlich sein, dass die Verwendung des Parameters verständlich ist.

`valueType`: Durch den Typ können zusätzlich zu normalen Parametern auch Passwortparameter unterstützt werden, für die ein Passwortfeld zur Eingabe erzeugt wird.

`rangeRegExp`: Der Wertebereich eines Parameters kann durch einen regulären Ausdruck definiert werden. Bei der Angabe eines Parameterwertes durch einen Benutzer wird der Ausdruck zur Validierung verwendet und eine Fehlermeldung ausgegeben, falls der Wert nicht im richtigen Bereich liegt.

`defaultValue`: Über dieses Attribut wird der Default-Wert des Parameters bestimmt. Der Wert muss in dem über `rangeRegExp` definierten Bereich liegen. Der Default-Wert wird verwendet, wenn ein Benutzer selbst keinen Wert für den Parameter angibt.

`value`: Dieses Attribut enthält den Wert, der durch einen Benutzer beim Aufruf des Services angegeben wird. Damit die Werte komfortabel ausgelesen werden können, bietet die Klasse `ParameterValues` Hilfsmethoden an, mit deren Hilfe über den symbolischen Namen der Wert abgefragt werden kann. Bei der Definition eines Parameters muss dieses Attribut nicht gesetzt werden.

Unterstützung der vier Ostp-Service-Kategorien

In Plug-in-Systemen können verschiedene Plug-in-Typen durch mehrere Schnittstellen oder Oberklassen definiert werden [MMS03]. Dementsprechend werden die vier Service-Kategorien jeweils durch eine eigene Subklasse von `OstpPlugin` in der Plug-in-API repräsentiert. Die Subklassen definieren jeweils eine abstrakte Methode, die in einem konkreten Service-Plug-in implementiert werden muss, um den Aufruf des Werkzeugs zu realisieren. Die Signaturen dieser Methoden, die in Abbildung 5.17 gezeigt sind, spiegeln das Ein- und Ausgabeverhalten der vier Ostp-Service-Kategorien wider.

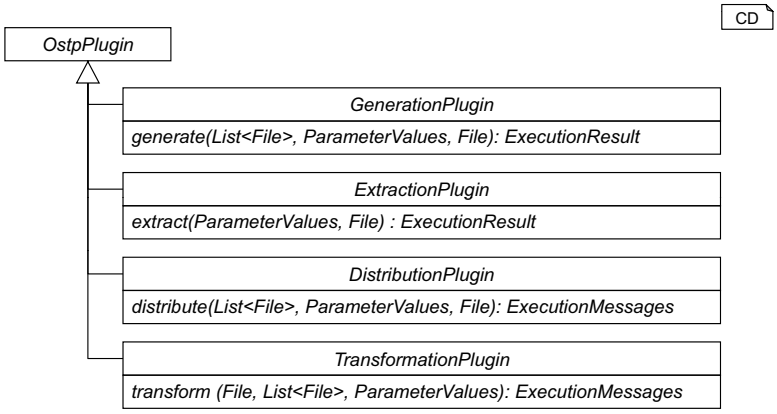


Abbildung 5.17: Methoden der Ostp-Plug-in-API zum Aufruf eines dateiverarbeitenden Werkzeugs

Beim Aufruf der drei Methoden `generate()`, `extract()` und `distribute()` wird jeweils zusätzlich zu den benötigten Eingabedateien und Parameterwerten noch ein Verzeichnis übergeben, das in der Implementierung des Plug-ins verwendet werden kann, um Ausgabe- oder temporäre Dateien zu speichern. Der `transform()`-Methode werden neben den Parameterwerten das Wurzelverzeichnis der Hierarchie und eine Liste von Eingabedateien aus der Hierarchie übergeben.

Die Rückgabetypen der Methoden sind in Abbildung 5.18 dargestellt. Die beiden Methoden `generate()` und `extract()` geben jeweils ein Objekt vom Typ `ExecutionResult` zurück. In der Implementierung der Methoden müssen dem Objekt explizit die Ausgabedateien gesetzt werden, damit sie zum Frontend übertragen werden. Zusätzlich können über beliebig viele `Message`-Objekte die Ausgaben des Werkzeugaufrufs übertragen werden. Dazu kann der Typ der Nachricht sowie ein Text gesetzt werden. Falls sich die Meldung auf eine bestimmte Datei bezieht, kann diese über das Attribut `resource` sowie die genaue Position in der Datei mit Hilfe von `line` und `column` angegeben werden. Darüber hinaus kann bei Bedarf ein Fehlercode gesetzt werden.

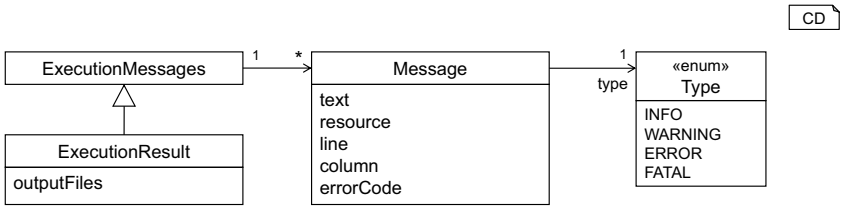


Abbildung 5.18: Rückgabetypen der Ostp-Plug-in-API

Die beiden Methoden `distribute()` und `transform()` geben jeweils ein Objekt vom Typ `ExecutionMessages` und damit keine Ausgabedateien zurück. Im Fall eines Distribution-Services ist dies durch das Ausgabeverhalten der Service-Kategorie begründet. Bei einem Transformation-Service wird im Gegensatz dazu der gesamte Verzeichnisbaum automatisch übertragen. Dadurch müssen in der Implementierung der `transform()`-Methode nicht explizit die Ausgabedateien gesetzt werden.

5.2.4 Integration von Social-Services

Soziale Netzwerke verwalten primär drei Arten von Informationen über Benutzer [KCSS10]. Einerseits werden Informationen über die Benutzer selbst gespeichert. Dazu gehören die Daten, die einen Benutzer eindeutig identifizieren, Profilinformationen und Datenschutzrichtlinien. Darüber hinaus werden die Kontakte von Benutzern durch soziale Graphen repräsentiert und gespeichert. Außerdem werden digitale Inhalte wie Nachrichten, Fotos und Videos von Benutzern verwaltet. Durch die Social-Services des Frameworks können momentan ausschließlich Profilinformationen aus sozialen Netzwerken oder anderen webbasierten Projektportalen durch Benutzer in eine Framework-Instanz importiert werden. Eine Erweiterung, um auch soziale Graphen verarbeiten und nutzen zu können, ist jedoch denkbar.

Die meisten sozialen Netzwerke bieten APIs zur Authentifizierung und Autorisierung an, damit diese Daten von externen Anwendungen und Webseiten genutzt werden können. Die Implementierungen dieser APIs sowie die eingesetzten Technologien und Protokolle unterscheiden sich jedoch [KCSS10]. Beim initialen Design des Social-Backends wurde daher auf die Verwendung offener und weit verbreiteter APIs und Standards geachtet [Man12]. Nach einer Analyse vorhandener Spezifikationen wurde das Social-Backend und die zugehörige Plug-in-API auf Basis von OAuth [HL10, Han11] und OpenSocial [OS11a, Häs11] konzipiert. Im Folgenden werden einige Grundlagen beider Spezifikationen erläutert, die für das Verständnis der nachfolgenden Methodik zur Entwicklung von Social-Plug-ins notwendig sind. Eine weiterführende Diskussion der Spezifikationen und der Design-Entscheidungen sind in [Man12] enthalten.

OAuth (Open Authorization) ist ein offener Protokollstandard für den autorisierten Zugriff auf geschützt Daten [HL10]. In einem klassischen Authentifizierungsmodell kann ein Client auf Daten, die auf einem Server gespeichert sind, üblicherweise nur mit Hilfe eines gültigen Benutzernamens und des zugehörigen Passworts zugreifen. OAuth erweitert dieses Modell, so dass Dritten ein Datenzugriff gewährt werden kann, ohne dass Benutzername und Passwort offen gelegt werden müssen. Dazu wird in OAuth eine zusätzliche Rolle, der Ressourceneigentümer (engl. resource owner) definiert, so dass ein Client im Auftrag des Eigentümers Zugriff auf die Daten erhält, ohne dessen Benutzername und Passwort zu kennen.

In der OAuth-Spezifikation [HL10] werden einige Begriffe definiert, die für das Verständnis der Authentifizierungs- und Autorisierungsmechanismen relevant sind. Die Verwendung dieser Begriffe wird im Folgenden zunächst kurz erläutert, bevor auf die Autorisierung im OAuth-Protokoll eingegangen wird.

Client: Ein HTTP-Client gemäß [FGM+99], der Anfragen unter Verwendung von OAuth stellen kann.

Server: Ein HTTP-Server gemäß [FGM+99], der OAuth-Anfragen entgegennehmen und verarbeiten kann.

Geschützte Ressource: Eine Ressource, die dem Ressourceneigentümer gehört und vom Server über eine OAuth-Anfrage abgerufen werden kann (engl. protected resource).

Ressourceneigentümer: Der Benutzer, der die geschützten Ressourcen auf dem Server besitzt und mit Hilfe von Credentials darauf zugreifen und sie kontrollieren kann (engl. resource owner).

Credentials: Ein Berechtigungsnachweis (engl. credentials) besteht aus einer eindeutigen Kennung und einem zugehörigen geteiltem Geheimnis. In der OAuth-Spezifikation werden drei Arten von Credentials unterschieden, die zur Identifikation und Authentifizierung von Clientanfragen, Autorisierungsanfragen und der Zugriffsberechtigung dienen. Diese drei Arten von Credentials werden als Client-, temporäre und Token-Credentials bezeichnet.

Token: Tokens werden im OAuth-Protokoll vom Server ausgestellt, um einem Client den autorisierten Zugriff auf Daten des Ressourceneigentümers zu ermöglichen. Ein Token dient zur eindeutigen Identifikation eines Clients und besitzt ein geteiltes Geheimnis, das vom

Client genutzt wird, um den Zugriff auf die Daten anstelle des Ressourceneigentümers zu ermöglichen.

Die Ausstellung von Tokens zur Autorisierung eines Clients für den Zugriff auf geschützte Ressourcen kann auf verschiedene Weisen erfolgen. Die gängigste Variante basiert auf der Verwendung von HTTP-Weiterleitung (engl. HTTP redirection) und wird dementsprechend Redirection-based Authorization genannt [HL10]. Abbildung 5.19 zeigt die Interaktion von Ressourceneigentümer, Client und Server bei dieser Variante in einem Aktivitätsdiagramm.

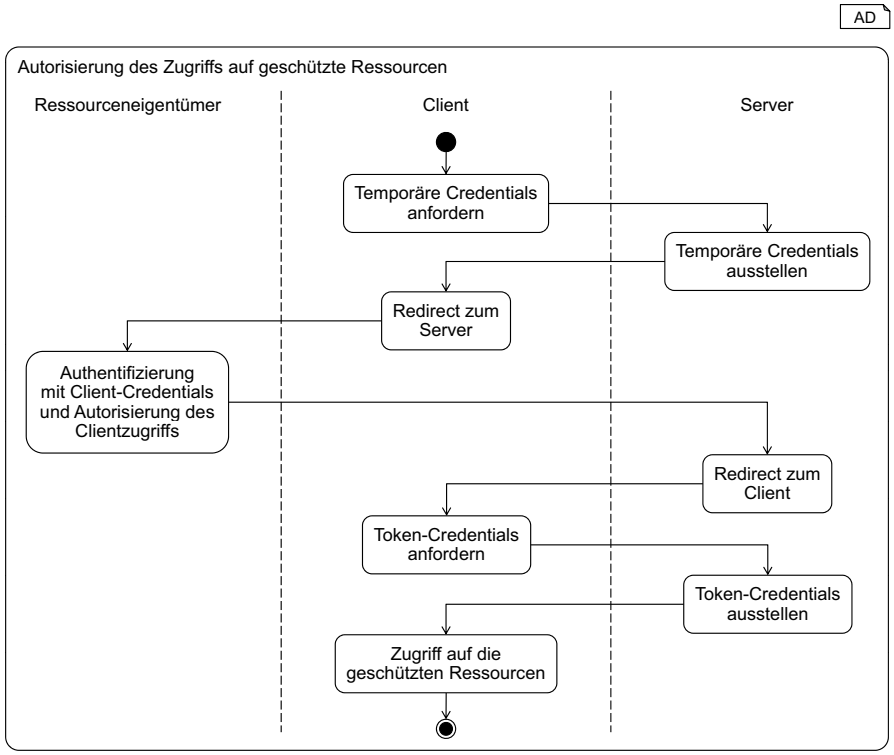


Abbildung 5.19: Autorisierung des Zugriffs auf geschützte Ressourcen durch einen Client mit Hilfe von OAuth

Zunächst fordert der Client, beispielsweise ein Social-Service-Plug-in, temporäre Credentials vom Server an. Diese werden durch den Server ausgestellt und sind zunächst nicht durch den Ressourceneigentümer autorisiert. Zu deren Autorisierung wird der Browser des Ressourceneigentümers auf die Seiten des Servers umgeleitet. Darauf muss sich der Ressourceneigentümer einloggen und die Anfrage des Clients autorisieren. Danach führt der Server ein Redirect zu dem

Client durch, so dass der Ressourceneigentümer wieder auf den Seiten des Clients ist. Der Client verwendet dann die autorisierten temporären Credentials, um vom Server Token-Credentials anzufordern, die dann für den Zugriff auf die geschützten Ressourcen genutzt werden können.

Damit überhaupt die Schritte zur Autorisierung aus Abbildung 5.19 durchgeführt werden können, muss ein Client zunächst beim Server registriert werden. Dazu muss der Entwickler des Clients ein gültiges Konto bei dem entsprechenden sozialen Netzwerk besitzen, über das er eine Client ID und ein zugehöriges Geheimnis beantragen kann. Dabei müssen üblicherweise noch einige zusätzliche Informationen angegeben werden, wie beispielsweise Kontaktdaten, der Name und eine Beschreibung der Anwendung sowie Informationen über die Organisation, die den Client anbietet.

OpenSocial baut unter anderem auf OAuth auf und ist eine offene Spezifikation, die eine Menge von APIs für die Entwicklung von Anwendungen im Kontext sozialer Netzwerke definiert [OS11a]. Mit Hilfe dieser APIs können Anwendungen geschrieben werden, die auf soziale Graphen und auf Profilinformationen sozialer Netzwerke zugreifen können [Häs11]. Die APIs basieren auf standardisierten Web-Technologien wie HTML und JavaScript und ermöglichen die Wiederverwendung von Anwendungen im Kontext mehrerer sozialer Netzwerke, die die Spezifikation erfüllen. Die sozialen Netzwerke, in deren Kontext eine Anwendung ausgeführt wird, werden in der Spezifikation als Container bezeichnet.

Aufbauend auf der vorigen Einführung beider Spezifikationen wird nachfolgend die Methodik zur Integration von Social-Services in das Framework diskutiert. Das Aktivitätsdiagramm in Abbildung 5.20 illustriert die dabei durchzuführenden Schritte.

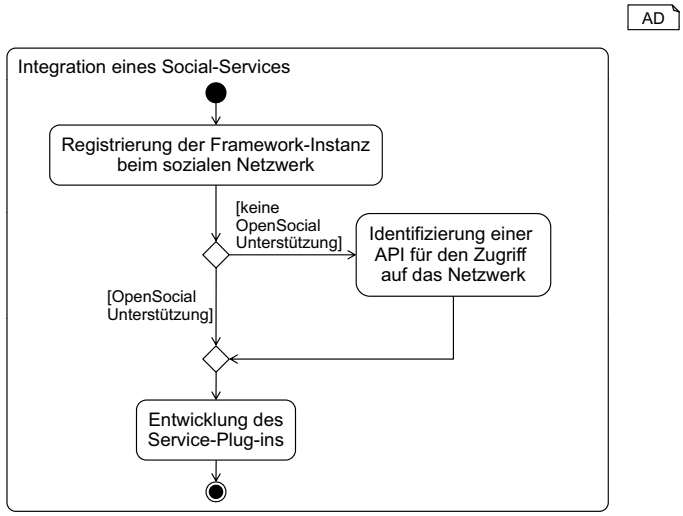


Abbildung 5.20: Methodik zur Integration von Funktionen sozialer Netzwerke als Social-Services

Registrierung der Framework-Instanz beim sozialen Netzwerk. Jedes Service-Plug-in agiert als ein HTTP-Client eines sozialen Netzwerks und verwendet OAuth für den autorisierten Zugriff auf die Profilinformationen von Benutzern. Dementsprechend muss vor Beginn der Entwicklung des Plug-ins die Framework-Instanz bei dem sozialen Netzwerk registriert und eine entsprechende Client-ID beantragt werden. Die meisten sozialen Netzwerke bieten dafür ein entsprechendes Web-Formular an.

Identifizierung einer API für den Zugriff auf das Netzwerk. Im darauf folgenden Schritt muss überprüft werden, ob das soziale Netzwerk die OpenSocial-Spezifikation unterstützt. Ist das nicht der Fall, kann bei der Entwicklung des Service-Plug-ins nicht auf die OpenSocial-API zurückgegriffen werden. Daher muss eine API identifiziert oder erstellt werden, die den Zugriff auf das soziale Netzwerk ermöglicht.

Schließlich muss ein Service-Plug-in für die Integration des entsprechenden Social-Services in das Framework erstellt werden. In Abbildung 5.21 sind die Klassen der Komponente Social-Plug-in-API gezeigt, die dabei genutzt werden müssen.

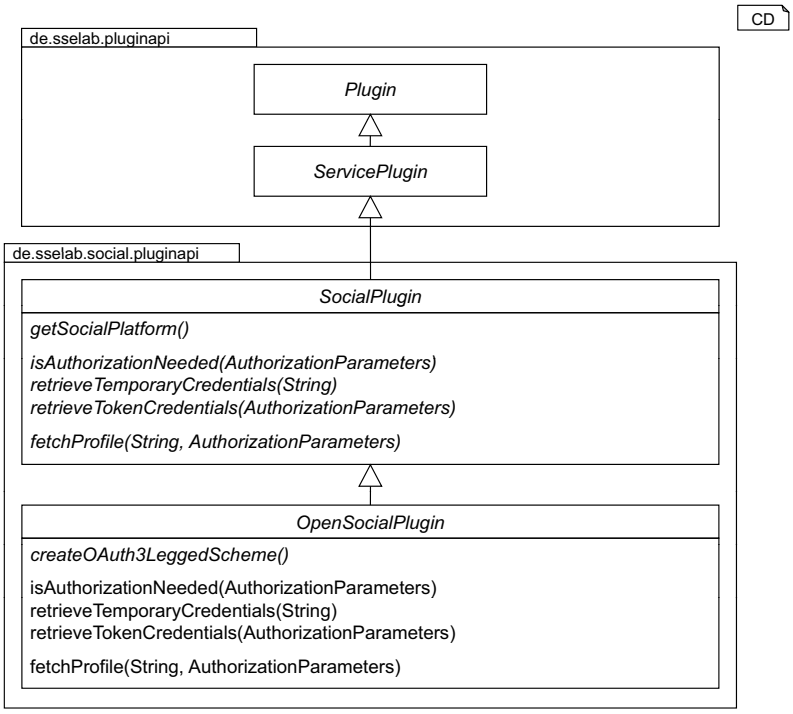


Abbildung 5.21: Klassen der Komponenten Social-Plug-in-API und Common-Plug-in-API

Die Klasse `SocialPlugin` erweitert direkt die `ServicePlugin`-Klasse der Komponente Common-Plug-in-API. `BotServicePlugin` ist in diesem Fall nicht Bestandteil der Vererbungshierarchie, da der Zugriff auf die eingebundenen sozialen Netzwerke nicht mit Hilfe von Bot-Accounts erfolgen soll, sondern ausschließlich durch die Benutzer selbst. In der Klasse sind alle Methoden definiert, die in einem konkreten Service-Plug-in realisiert werden müssen oder überschrieben werden können. Zusätzlich zu den grundlegenden Informationen über einen Social-Service, die über die Methoden der Oberklassen definiert werden, wird durch die Methode `getSocialPlatform()` der Name des sozialen Netzwerks bzw. des Portals angegeben, aus dem Profilinformationen durch das zugehörige Plug-in abgerufen werden können. Die weiteren Methoden der Klasse ermöglichen die Authentifizierung und Autorisierung eines Benutzers bei dem Netzwerk sowie den Abruf der Profilinformationen. Die Methodik zur Implementierung und Verwendung dieser Methoden wird in den nachfolgenden Abschnitten diskutiert.

Darüber hinaus sind in der Subklasse `OpenSocialPlugin` einige der Methoden auf Grundlage der OpenSocial-API implementiert. Die Realisierung von Service-Plug-ins für soziale Netzwerke, die die Spezifikation erfüllen, wird durch die Verwendung dieser Klasse dementsprechend vereinfacht. Die Methodik zur Erweiterung dieser Klasse wird ebenfalls im Folgenden behandelt.

Autorisierung mittels OAuth

Jedes Social-Service-Plug-in kapselt die Kommunikation mit einem konkreten sozialen Netzwerk und ist daher gemäß der OAuth-Spezifikation ein Client. Da ein Benutzer durch ein Plug-in jedoch nicht per HTTP-Weiterleitung auf die Seiten des sozialen Netzwerks umgeleitet werden kann, wird der zuvor beschriebene OAuth-Autorisierungsprozess durch ein Plug-in in Zusammenarbeit mit der Komponente SSELab-Web durchgeführt. Die Methoden, die in einem Plug-in zur Realisierung des Prozesses enthalten sind, werden nachfolgend beschrieben. Diese Methoden verwenden unter anderem Objekte vom Typ `AuthorizationParameters` als Rückgabewerte und Parameter. In Objekten dieser Klasse werden die Daten gespeichert, die beim Durchlauf des Autorisierungsprozesses erzeugt werden. Abbildung 5.22 zeigt diese Klasse und die assoziierte Klasse `Credential`, mit deren Hilfe die temporären und die Token-Credentials gespeichert werden. Zusätzlich werden die Autorisierungs-URL und der Verifikationscode, der zur Identifikation eines Benutzers dient, hier abgelegt.

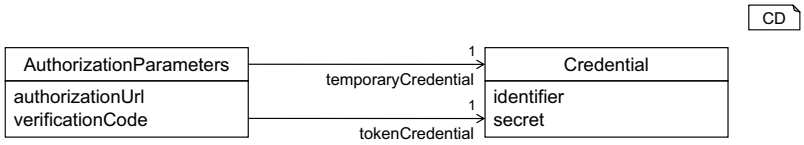


Abbildung 5.22: Parameter- und Rückgabetyp der Social-Plug-in-API

`isAuthorizationNeeded()`: Diese Methode kann jederzeit vom Frontend aufgerufen werden, um abzufragen, ob der Autorisierungsprozess durchlaufen werden muss. In der Implementierung dieser Methode muss entsprechend geprüft werden, ob das Plug-in noch

für den Zugriff auf das soziale Netzwerk anstelle des Benutzers autorisiert ist. Eine Erneuerung der Autorisierung durch den Benutzer muss erfolgen, falls der Autorisierungsprozess noch nicht durchgeführt wurde oder wenn die Token-Credentials abgelaufen sind.

`retrieveTemporaryCredentials()`: Durch diese Methode wird der erste Schritt im OAuth-Prozess definiert, in dem die temporären Credentials angefordert werden. In der Realisierung dieser Methode müssen die temporären Credentials vom Server abgefragt und die URL durch eine Anfrage an den Server ermittelt werden, unter der der Benutzer den Zugriff des Plug-ins freigeben kann. Sowohl die URL als auch die Credentials müssen in einem `AuthorizationParameters`-Objekt gespeichert und zurückgegeben werden. Beim Aufruf der Methode wird zusätzlich eine Callback-URL vom Frontend übergeben, die vom Server zum Aufbau der Autorisierungs-URL genutzt wird, so dass nach erfolgreicher Autorisierung der Benutzer wieder zu der Framework-Instanz zurückkehren kann.

`retrieveTokenCredentials()`: Diese Methode muss in einem konkreten Plug-in realisiert werden, so dass die Token-Credentials vom Server abgefragt werden können. In der Implementierung muss dann ein entsprechendes `Credential`-Objekt erzeugt und über das übergebene `AuthorizationParameters`-Objekt zurückgegeben werden.

Abruf von Profilinformationen

Mit Hilfe der Methode `fetchProfile()` können die Profilinformationen eines Benutzers von dem zugehörigen sozialen Netzwerk abgerufen werden. Beim Aufruf der Methode werden der Benutzername und die zuvor beschriebenen Autorisierungsparameter übergeben. Die Implementierung der Methode ist dabei abhängig von dem konkreten sozialen Netzwerk. Es sollten jedoch bei einem Aufruf möglichst alle benötigten Daten auf einmal abgerufen werden, damit unnötige Verbindungen vermieden werden können. Nach dem Abruf der Profilinformationen vom Netzwerk müssen die Daten in ein Objekt vom Typ `SocialProfile` gespeichert und schließlich zurückgegeben werden.

Zugriff auf OpenSocial-Container

In der Klasse `OpenSocialPlugin` sind die Methoden zur Autorisierung mit Hilfe von OAuth und zum Abruf von Profilinformationen unter Verwendung der OpenSocial-API bereits implementiert, so dass diese Methoden in einer Subklasse nicht mehr implementiert werden müssen. Nur die neu definierte Methode `createOAuth3LeggedScheme()` muss in einer Subklasse realisiert und mit Hilfe der OpenSocial-API ein Objekt für das zugehörige soziale Netzwerk erzeugt und zurückgegeben werden. Im folgenden Kapitel 6 wird ein Beispiel für die Implementierung dieser Methode gezeigt.

Eine analoge Realisierung der Funktionalität der `SocialPlugin`-Klasse auf Grundlage anderer APIs wäre ebenfalls möglich. Dadurch könnte die Erstellung von Service-Plug-ins auf Grundlage dieser APIs entsprechend vereinfacht werden.

5.3 Entwicklung und Erweiterung von Framework-Clients

In Abschnitt 4.5 wurden die wichtigsten Konzepte und die Architektur der Framework-Clients beschrieben. In diesem Abschnitt werden darauf aufbauend sowohl die Entwicklung neuer Clients auf Grundlage der Client-APIs als auch die Erweiterung des Eclipse-Clients behandelt.

5.3.1 Entwicklung von Framework-Clients

Die in Abschnitt 4.5.1 vorgestellten APIs erlauben die Entwicklung von Framework-Clients für Benutzer und Administratoren einer Framework-Instanz. In Abbildung 5.23 ist noch einmal die Architektur beider Client-APIs gezeigt.

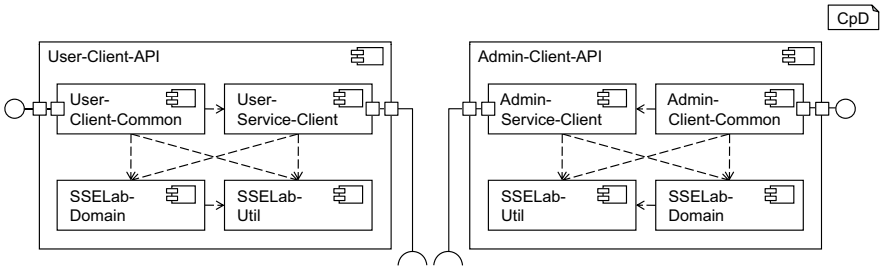


Abbildung 5.23: Architektur der Framework-Client-APIs

Die Komponenten User-Service-Client und Admin-Service-Client ermöglichen den Zugriff auf Funktionen des Frontends. Darauf aufbauend stellen die Komponenten User-Client-Common und Admin-Client-Common jeweils eine API zur Entwicklung von Framework-Clients für Benutzer und Administratoren zur Verfügung. Ein Auszug aus beiden APIs ist in Abbildung 5.24 gezeigt. Darin sind unter anderem Methoden für die Authentifizierung, für den Abruf von Informationen über Projekte und Services und den Aufruf von Funktionen der Ostp-Backends enthalten. Alle wesentlichen Methoden werden über die beiden Klassen `UserAccess` und `AdminAccess` angeboten. Bei der Entwicklung neuer Framework-Clients muss daher nur die API dieser beiden Klassen verwendet werden. Anhand einiger repräsentativer Methoden wird im Folgenden die Verwendung dieser APIs erläutert.

- Die Klassen `UserAccess` und `AdminAccess` sind beide als Singleton [GHJV95] realisiert, so dass die Instanzen der beiden Klassen jeweils durch Aufruf der `getInstance()`-Methode abgerufen werden können.
- Mit Hilfe der Klasse `AuthenticationHandler` erfolgen die Authentifizierung eines Benutzers bzw. Administrators und die Verwaltung der Zertifikate für den Zugriff auf das Frontend. Über die drei Methoden `login(String, String)`, `logout()` und `isLoggedIn()` wird diese Funktionalität auch über die beiden Access-Klassen angeboten. Im Zuge der Authentifizierung wird auch die Kompatibilität der Client-API mit der Frontend-Instanz überprüft und bei Inkompatibilität ein entsprechender Fehler erzeugt.

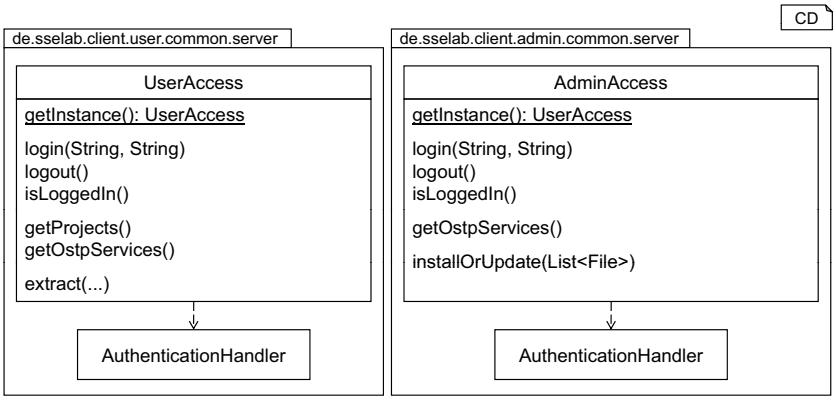


Abbildung 5.24: Auszug der APIs aus User-Client-Common und Admin-Client-Common

- Nach einer erfolgreichen Authentifizierung können die übrigen Methoden beider Klassen aufgerufen werden. Die Methoden erlauben einerseits, Informationen vom Frontend abzurufen. In Abbildung 5.24 sind beispielsweise die Methoden `getProjects()` und `getOstpServices()` gezeigt, über die die Projekte des authentifizierten Benutzers bzw. Informationen über alle verfügbaren Ostp-Services abgerufen werden können.

  Zusätzlich zu den beschriebenen Methoden ermöglichen die beiden Klassen noch die Interaktion mit den Service-Backends und deren Services. Die Klasse `UserAccess` erlaubt die Ausführung von Ostp-Services. In Abbildung 5.24 ist zur Veranschaulichung die Methode `extract()` gezeigt, mit deren Hilfe Ostp-Services der Kategorie Extraction-Service ausgeführt werden können. Die Klasse `AdminAccess` enthält dagegen Methoden zur Verwaltung von Service-Plug-ins. Mit Hilfe der Methode `installOrUpdate()` können beispielsweise Service-Plug-ins installiert oder aktualisiert werden.

Aufbauend auf der Funktionalität dieser APIs können einerseits Framework-Clients und andererseits APIs mit einer höheren Abstraktion entwickelt werden. Im Rahmen dieser Arbeit wurde beispielsweise eine High-Level-API zur Ausführung von Ostp-Services auf Grundlage von `UserAccess` realisiert. Abbildung 5.25 zeigt die Klasse `OstpExecutionHandler` für die Ausführung von Ostp-Services. Diese API wird beispielsweise von den in Abschnitt 4.5.2 vorgestellten Eclipse-, Shell- und Batch-Clients verwendet.

Die notwendigen Informationen zur Ausführung eines Ostp-Services werden dabei in einem `ExecutionData`-Objekt gekapselt, das dann als Parameter beim Aufruf der Methode `executeOstpService()` übergeben werden kann. Die Daten werden dann durch die Methoden der Klasse `ExecutionDataValidator` überprüft und anschließend das entsprechende Ostp-Service-Plug-in aufgerufen.

Diese kurze Einführung der Client-APIs soll nur ein Gefühl für deren Verwendung und Eigenschaften vermitteln. Bei dem Design der APIs wurde insbesondere darauf geachtet, dass kompakte und einfach zu verwendende Schnittstellen entstehen, damit die Entwicklung von

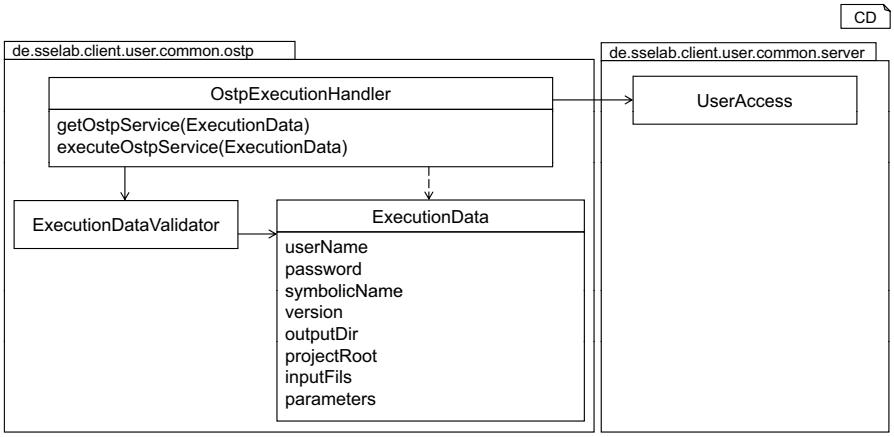


Abbildung 5.25: High-Level-API für die Ausführung von Ostp-Services

Framework-Clients effizient möglich ist. Weiterführende Dokumentation dieser APIs ist in der Implementierung des Frameworks vorhanden.

5.3.2 Erweiterung des Eclipse-Clients

Der Eclipse-Client basiert auf der im vorigen Abschnitt vorgestellten API für Benutzer und realisiert die Integration in die Eclipse-Plattform. Über diesen Client kann sich ein Benutzer aus Eclipse heraus bei einer Framework-Instanz authentifizieren und danach beliebige Ostp-Services aufrufen. Dazu bietet der Eclipse-Client einen Dialog an, in dem nach einem erfolgreichen Login alle verfügbaren Ostp-Services angezeigt werden. Einzelne Ostp-Services können nach der Authentifizierung auf konkreten Dateien oder Projekten in der IDE aufgerufen werden.

Insgesamt können mit Hilfe dieses Clients alle Ostp-Services genutzt werden, die in der Framework-Instanz verfügbar sind. Darüber hinaus ist es auch möglich, diesen Client aus anderen Eclipse-Plug-ins heraus aufzurufen. Dafür wird eine kompakte API zum Aufruf von Ostp-Services durch eine Klasse des Clients angeboten. Bei der Verwendung dieser Klasse wird automatisch geprüft ob bereits eine Authentifizierung erfolgt ist und bei Bedarf ein Authentifizierungsdialog geöffnet. Außerdem werden nach der Ausführung des Services alle produzierten Ausgaben in eine Konsole von Eclipse geschrieben sowie alle erzeugten oder geänderten Dateien in der IDE aktualisiert.

Insgesamt ermöglicht diese API eine für den Benutzer transparente Nutzung von Ostp-Services durch andere Eclipse-Plug-ins. Nur durch den bei Bedarf erscheinenden Authentifizierungsdialog wird die Kommunikation mit einer Framework-Instanz deutlich. Durch die Realisierung und Nutzung dieses Clients konnte daher die Kopplung von IDE und CDE auch im Kontext der servicebasierten Nutzung dateiverarbeitender Werkzeuge erfolgreich gezeigt werden.

5.4 Zusammenfassung

In diesem Kapitel wurden die Methodiken zur Erweiterung des SSELab-Frameworks vorgestellt. Zunächst wurde die Unterstützung neuer Service-Kategorien mit den zugehörigen Backend-Arten durch eine Erweiterung des Frameworks skizziert. Durch die Anforderungen an die Funktionalität des Frontends und dadurch, dass neue Service-Kategorien vergleichsweise selten hinzugefügt werden müssen, besitzt dieser Erweiterungspunkt deutliche white-box Anteile und erfordert eine detailliert Kenntnis der Frontend-Architektur. Ansätze, um eine effektivere Unterstützung neuer Service-Kategorien zu erreichen, wurden daher im Anschluss an die Beschreibung dieses Erweiterungspunkts skizziert.

Im Vordergrund diese Kapitels standen die Methodiken zur Integration von Services. Für jede Service-Kategorie wurden die durchzuführenden Schritte vorgestellt sowie die Entwicklung von Service-Plug-ins auf Grundlage der Framework-APIs detailliert behandelt. Die Entwicklung von Plug-ins für die drei unterstützten Service-Kategorien basiert dabei auf einem black-box Ansatz auf Grundlage der reinen Plug-in-Architekturen der Backends.

Abschließend wurden die Erweiterungspunkte der Framework-Clients beschrieben. Dabei wurde sowohl auf Entwicklung von Framework-Clients als auch auf die Erweiterung des Eclipse-Clients eingegangen.

Kapitel 6

Integrierte Werkzeuge und Clients

In diesem Kapitel werden aufbauend auf den in Kapitel 5 beschriebenen Methodiken zur Erweiterung des Frameworks die Integration konkreter Werkzeuge sowie Erweiterungen von Framework-Clients diskutiert. Dazu wird zunächst ein Überblick über alle Werkzeuge gegeben, die im Rahmen dieser Arbeit als Services in das Framework integriert worden sind. Anschließend werden anhand einiger repräsentativer Werkzeuge die Konfiguration und Integration sowie die Entwicklung der zugehörigen Service-Plug-ins für die verschiedenen Backend-Arten erläutert. Abschließend werden einige Anforderungen an Werkzeuge diskutiert, die auf den gesammelten Erfahrungen basieren und deren Umsetzung die Integration von Werkzeugen erleichtern würde.

6.1 Übersicht der integrierten Werkzeuge

Ein Framework vereint ein abstraktes Design und verallgemeinerte Konzepte in einer halbfertigen Architektur für Anwendungen einer bestimmten Domäne. Da jedoch geeignete Abstraktionen und zugehörige Konzepte nur effektiv auf Grundlage konkreter Beispiele gefunden werden können [RJ96], wurden im Rahmen dieser Arbeit vielfältige Werkzeuge in das Framework integriert. Eine weitere Motivation für die Integration konkreter Werkzeuge war, dass Instanzen des Frameworks produktiv in Forschungsprojekten und in der Lehre eingesetzt werden sollten, um die Praxisrelevanz und die Nutzbarkeit des Designs und der Konzepte zu zeigen (vgl. Kapitel 8). Aus diesem Grund wurden primär Werkzeuge in das Framework integriert, die am Lehrstuhl für Software Engineering bereits isoliert voneinander genutzt worden sind. Außerdem wurden auch die Eigenentwicklungen des Lehrstuhls berücksichtigt, so dass diese effizient über die Grenzen des Lehrstuhls hinaus den Projektpartnern und anderen Interessenten zugänglich gemacht werden können.

In diesem Abschnitt wird ein Überblick über alle integrierten Werkzeuge gegeben, wobei auch der Bezug zu den Anforderungen aus Kapitel 3 hergestellt und bei Bedarf weitere Gründe für die Wahl der Werkzeuge diskutiert werden.

6.1.1 Base-Services

Im Rahmen dieser Arbeit wurden die folgenden Werkzeuge als Base-Services integriert, die damit über eine Framework-Instanz als Kerninfrastruktur für Projekte angeboten werden können. Gemäß Anforderung **FA21** (Integration existierender Werkzeuge) wurde einerseits auf die Wahl gängiger und weit verbreiteter Anwendungen geachtet und andererseits die vorhandene Infrastruktur am Lehrstuhl berücksichtigt.

Subversion und Git: Versionsverwaltungssysteme [Lou06b] sind ein wichtiger Teil der Kerninfrastruktur heutiger Software-Entwicklungsprojekte, über die üblicherweise die Datenintegration erfolgt [Whi07]. Um Anforderung **FA26** zu erfüllen, wurden im Rahmen dieser Arbeit die Versionsverwaltungssysteme Subversion [PCF08, Col02] und Git [Cha09] in das Framework integriert. Subversion wurde gewählt, da es bereits Bestandteil der Lehrstuhlinfrastruktur war. Git wurde zusätzlich integriert, weil es frei verfügbar ist, sich in letzter Zeit schnell verbreitet hat und im Gegensatz zu Subversion ein dezentrales System ist.

Storage: Für Artefakte, die nicht über ein Versionsverwaltungssystem ausgetauscht werden sollen, wurde gemäß Anforderung **FA27** ein Storage-Service integriert. Dieser Service basiert auf dem WebDAV-Protokoll [WG99] und kann mit WebDAV-fähigen Clients zur zentralen Speicherung unversionierter Dateien verwendet werden. Der Download der Dateien ist sowohl über solche Clients als auch über die Webseite des Services möglich.

Trac: Für die Umsetzung der Anforderungen **FA28** (Bereitstellung eines Projektmanagementwerkzeugs) und **FA29** (Integration eines Bug-Trackers) wurde Trac [Tra12] als Service in das Framework integriert. Trac vereint mehrere Funktionen: Es verfügt über einen Bug-Tracker, über ein Projekt-Management-Modul, über ein Wiki und über einen Repository-Browser für Subversion. Weiterhin kann Trac über Plug-ins erweitert werden. Trac wurde gewählt, da es einerseits über die beschriebenen Funktionen verfügt, aber auch ein leichtgewichtiges Projektmanagement-Werkzeug ist, das für agile Prozesse geeignet ist und bereits am Lehrstuhl im Einsatz war.

MediaWiki: Wikis sind weit verbreitet im Software Engineering [STvDC10]. Sie werden unter anderem für die Unterstützung informeller Kommunikation, beim Bug-Tracking, zur Dokumentation, im Requirements-Engineering, Testmanagement und auch zur Erzeugung von Projektportalen genutzt [ACGL08, Lou06a]. Im Rahmen dieser Arbeit wurde gemäß Anforderung **FA30** auch ein Wiki in das Framework integriert. Als Wiki-System wurde MediaWiki [MW12] ausgewählt, da es insbesondere durch den Einsatz für Wikipedia ein viel genutztes und weit verbreitetes System ist. MediaWiki wurde in zwei Varianten als Service integriert. Einmal als Variante bei der nur die Teilnehmer eines Projekts die Inhalte nutzen können (*internal Wiki*) und einmal als Variante bei der die Inhalte auch ohne Authentifizierung gelesen, aber nur von Teilnehmern eines Projekts editiert werden können (*external Wiki*). Die externe Variante kann beispielsweise für die Außendarstellung von Projekten genutzt werden, so dass Anforderung **FA31** umgesetzt wird.

Web-Space: Zusätzlich zu dem externen Wiki wurde der Web-Space-Service für die Außendarstellung von Projekten gemäß Anforderung **FA31** realisiert und in das Framework integriert. Mit Hilfe dieses Services können Projektmitglieder statische Webseiten über WebDAV [WG99] hochladen, die dann über eine öffentliche URL ohne Authentifizierung abgerufen werden können. Im Gegensatz zum externen Wiki wird dabei kein spezifisches Design vorgegeben, so dass die statischen Webseiten beliebig an die Bedürfnisse eines Projekts angepasst werden können.

Mailing-Listen: Um Anforderung **FA32** zu erfüllen, wurde eine Mailing-Liste unter Verwendung von Courier MLM [Cou12] in das Framework integriert. Courier MLM wurde ausgewählt, da es über Textdateien und kommandozeilenbasierte Administrationswerkzeuge konfiguriert werden kann. Zur webbasierten Archivierung der Mails wird zusätzlich MHonArc [Mho12] als Teil des Mailing-Listen-Services eingesetzt.

Feedback-System: Das Feedback-System [HKR12a] ist eine Eigenentwicklung des Lehrstuhls, die gemäß Anforderung **FA22** (Integration von Eigenentwicklungen) ebenfalls als Service integriert worden ist. Das Werkzeug bietet mehrere Client-Komponenten an, die in ein Produkt – beispielsweise eine Web-, Eclipse-RCP- oder Android-Anwendung – integriert werden können. Mit Hilfe dieser Komponenten können die Kunden bzw. die Nutzer des Produkts Feedback an dessen Entwickler senden – beispielsweise im Rahmen von Akzeptanztests in verteilten Entwicklungsprojekten [LHK+12]. Dazu kommunizieren die Client-Komponenten mit einem Server, der das Feedback in einem Bug-Tracking-System speichern kann. Die in das Framework integrierte Version des Werkzeugs verwendet Trac zur Speicherung von Feedback.

Jenkins: Continuous Integration [Fow06] beschreibt eine Methode des Software Engineering, bei der die Integration eines Systems bzw. einer Anwendung kontinuierlich und nicht erst nachgelagert erfolgt. Es existieren viele verschiedene frei verfügbare und kommerzielle Server, die diese Methode unterstützen. Im Rahmen dieser Arbeit wurde Jenkins [Sma11] als Continuous Integration Server in das Framework integriert, da er weit verbreitet ist und auch am Lehrstuhl bereits unabhängig vom SSELab eingesetzt worden ist.

Nexus: In Java-Entwicklungsprojekten werden häufig Maven-Repositories für die Verwaltung von Artefakt-Abhängigkeiten (größtenteils Java-Archive) genutzt. Die zentralen Repositories von Maven sind jedoch primär für einen lesenden Zugriff gedacht. Für eine flexible Verwaltung eigener und externer Artefakte sollten daher interne Repositories verwendet werden. Nexus [MOB+11, Nex12] ist ein frei verfügbarer Repository-Server und wurde im Rahmen dieser Arbeit für die Integration ausgewählt, da er eine ausreichend umfangreiche REST-API zur Konfiguration bereitstellt.

Wordpress: Blogs bieten genau wie Wikis eine einfache Möglichkeit, die informelle Kommunikation in verteilten Projekten zu verbessern [ACGL08]. Außerdem werden Blogs für die Dokumentation von „How-Tos“, zur Diskussion neuer Features, zum Austausch technischen Wissens und im Kontext des Requirements Engineering eingesetzt [PM09, STvDC10]. Im Rahmen dieser Arbeit wurde WordPress [WP12] als Blog-Plattform ausgewählt und als Service integriert.

6.1.2 Ostp-Services

Als dateiverarbeitende Werkzeuge wurden hauptsächlich Eigenentwicklungen des Lehrstuhls in das Framework integriert. Die folgende Auflistung gibt eine Übersicht über diese Werkzeuge.

MontiCore: MontiCore [GKRS06, KRV07b, KRV08, KRV10, Kra10] ist ein Framework für die Entwicklung textueller domänenspezifischer Sprachen (DSLs). Die DSLs werden in

einem grammatikbasierten Format definiert, das von MontiCore für die Generierung von sprachverarbeitenden Werkzeugen genutzt wird. Beispiele dafür sind Parser, Klassen zur Repräsentation der abstrakten Syntax, Symboltabellen und Pretty-Printer. Darüber hinaus unterstützt MontiCore die Generierung von Editoren für eine DSL zur Integration in die Eclipse IDE [KRV07a].

MontiArc: Unter Verwendung von MontiCore definiert MontiArc [HRR12] eine Modellierungssprache zur Beschreibung von Architekturen reaktiver Softwaresysteme mit Hilfe von Komponenten und Konnektoren. Neben der Sprache selbst bietet MontiArc Editorunterstützung für Eclipse sowie ein Simulationsframework mit einem Java-Generator, das als separater Service integriert worden ist.

MontiWIS: MontiWIS [RR12] basiert ebenfalls auf MontiCore und ist der Nachfolger von MontiWeb [DRRS09]. Es verwendet mehrere Sprachen für die Generierung von Web-Informationssystemen. Unter anderem werden Klassen- und Aktivitätsdiagramme eingesetzt, um die Struktur und das Verhalten solcher Systeme zu definieren.

XML Schema Generator: Mit Hilfe dieses Werkzeugs kann aus einer Menge von XML-Dokumenten [BPM+08] eine XML Schema Definition (XSD) [TBMM04, BM04] abgeleitet werden, die die Struktur aller übergebenen XML-Dokumente beschreibt [Her05].

WikiBot: Durch dieses Werkzeug können Seiten bzw. Artikel, die in einem Wiki gespeichert sind, sowohl abgerufen als auch hochgeladen und dadurch aktualisiert werden. Bei der Integration dieses Werkzeugs wurde der Zugriff auf Wikis eingeschränkt, die innerhalb einer SSELab-Instanz angeboten werden.

ppt2eps: Durch dieses Werkzeug können Microsoft Office Dokumente in EPS- oder PDF-Abbildungen konvertiert werden [Mec12]. Das Werkzeug ist selbst als erweiterbarer Hosted Service realisiert, der über einen Ostp-Service in das Framework integriert worden ist.

6.1.3 Social-Services

Im Rahmen dieser Arbeit wurden für die folgenden sozialen Netzwerke Service-Plug-ins realisiert. Diese initiale Auswahl hat vorwiegend technische Gründe, da die Social-Plug-in-API auf Grundlage dieser Netzwerke entwickelt worden ist.

Google: Google unterstützt OpenSocial und erlaubt unter Verwendung eines gültigen Accounts den Abruf einiger Informationen von den OpenSocial-Containern (z.B. iGoogle, Gmail).

MySpace: MySpace ist ebenfalls ein OpenSocial-Container, bietet jedoch im Gegensatz zu Google einen Zugriff auf umfangreichere Informationen über einen Benutzer. Das MySpace-Plug-in wurde daher primär zum Test der API und der Datenstruktur von Social-Plug-ins verwendet.

LinkedIn: LinkedIn bietet keine Unterstützung für OpenSocial und wurde gewählt, damit die Social-Plug-in-API auch für die Nutzung von Protokollen und Schnittstellen anderer Netzwerke ausgelegt werden konnte.

6.2 Integration repräsentativer Werkzeuge

In diesem Abschnitt wird die Integration von Werkzeugen und die Realisierung von Service-Plug-ins sowie die Erweiterung der Framework-Clients anhand einiger repräsentativer Beispiele diskutiert. Dabei werden nicht alle Details der Integration und der Plug-in-Entwicklung beschrieben, sondern nur die wichtigsten Aspekte hervorgehoben. Die Erläuterungen bauen darüber hinaus aufeinander auf, so dass bereits eingeführte Konzepte und Lösungsansätze nicht wiederholt werden. Das Ziel dieses Abschnitts ist, die im vorigen Kapitel 5 beschriebenen Methodiken zur Integration von Werkzeugen anhand konkreter Beispiele zu illustrieren. Abschließend wird jeweils diskutiert, welche positiven und negativen Erfahrungen bei der Integration eines Werkzeugs gemacht worden sind.

6.2.1 Subversion

Subversion [PCF08, Col02] ist ein serverbasiertes Werkzeug und als Versionsverwaltungssystem eine der Kernkomponenten von Software-Entwicklungsprojekten. Für die Integration von Subversion wurde daher ein Base-Service-Plug-in realisiert, so dass das Werkzeug als Kerninfrastruktur über eine Framework-Instanz angeboten werden kann. Im Folgenden werden die in Abschnitt 5.2.2 diskutierten Schritte der Methodik zur Integration eines Base-Services am Beispiel von Subversion behandelt.

Installation der Anwendung mit ihren Abhängigkeiten. Da Subversion ein frei verfügbares und weit verbreitetes Versionsverwaltungssystem ist, existieren installierbare Pakete für alle gängigen Plattformen. Darüber hinaus unterstützt Subversion mehrere Server [PCF08], wie beispielsweise Apache HTTPD [LL02], so dass auch die Installation der Abhängigkeiten sowie der gesamten Laufzeitumgebung problemlos möglich ist. Für die Integration in das Framework wurde Apache HTTPD als Server gewählt. Darüber hinaus wurde die Installation aller Komponenten für die Zielplattform der unter `http://sselab.de` verfügbaren Instanz über Skripte automatisiert.

Integration mit den Authentifizierungs- und Autorisierungsmechanismen. Das Frontend des Frameworks kann bei der Erzeugung einer Instanz für verschiedene Authentifizierungsarten konfiguriert werden. Für eine möglichste einfache Unterstützung von Single-Sign-On wurde bei der Realisierung HTTP Basic Authentication [FHBH+99] in Kombination mit einer TLS-Verschlüsselung gewählt. Unter Verwendung des Apache Web-Servers kann der Zugriff auf Subversion-Repositories ebenfalls durch HTTP Basic Authentication geschützt werden, so dass die Integration der Authentifizierungsmechanismen problemlos möglich ist.

Die Verwaltung der Zugriffsrechte der Benutzer aller Subversion-Repositories erfolgt über eine zentrale Konfigurationsdatei. Durch die Anpassung dieser Datei können zur Laufzeit

des Servers die Zugriffsrechte geändert werden. Die Integration mit den Autorisierungsmechanismen des Frameworks erfolgt daher durch eine Generierung dieser Datei durch das Service-Plug-in.

Anpassung des Erscheinungsbilds. Subversion ist primär für die Nutzung über eigenständige Clients und nicht für die Nutzung über eine Web-Oberfläche konzipiert. Daher unterstützt die Standard-Installation von Subversion nur eine Web-Oberfläche für einen lesenden Zugriff. Die Seite wird durch ein XSLT-Skript [Cla99] erzeugt, durch dessen Modifikation das Erscheinungsbild der Web-Oberfläche angepasst werden kann. Darüber hinaus existieren auch Web-Clients für Subversion mit umfangreicher Funktionalität, die jedoch im Rahmen dieser Arbeit nicht integriert worden sind, da Trac einen solchen Client bereitstellt.

Vorbereitung einer automatischen Konfiguration. Subversion stellt eine Vielzahl kommandozeilenbasierter Administrationswerkzeuge zur Verfügung und nutzt Textdateien und Skripte zur Konfiguration. Die automatische Erzeugung von Repositories sowie deren Konfiguration ist daher ohne weitere Vorbereitung möglich.

Die Integration von Subversion in das Framework erfolgt über das zugehörige Subversion-Plug-in, dessen Oberklasse und realisierte Methoden in Abbildung 6.1 dargestellt sind. Da Subversion als Base-Service integriert worden ist, erweitert die Klasse `SubversionPlugin` die Komponente Base-Plug-in-API.

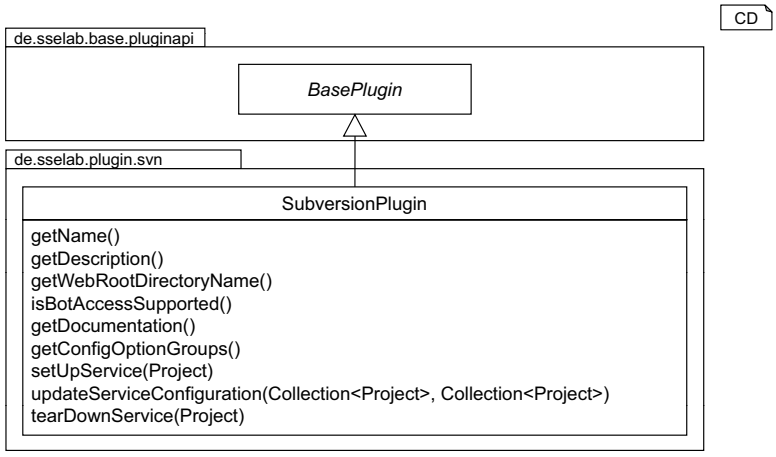


Abbildung 6.1: Beziehungen zur Base-Plug-in-API und realisierte Methoden des Subversion-Plug-ins

Die Methoden `getName()`, `getDescription()` und `getWebRootDirectoryName()` geben den Namen, die Beschreibung und die Bezeichnung des Wurzelverzeichnisses, in dem alle Subversion-Repositories enthalten sind, zurück. Darüber hinaus wird der Zugriff auf

Subversion-Repositories durch Bot-Accounts durch die Realisierung der entsprechenden Methode freigegeben. Weiterhin ist in dem Plug-in grundlegende Dokumentation von Subversion enthalten, die mit Hilfe der in Abschnitt 5.2.1 beschriebenen Mechanismen über die Methode `getDocumentation()` bereitgestellt wird.

Im Subversion-Plug-in werden die in Abschnitt 5.2.2 erläuterten Konfigurationsoptionen genutzt, um den Teilnehmern eines Projekts die Konfiguration des Commit-Mail-Verhaltens und der Zugriffsrechte eines Repositories zu ermöglichen. Durch die Realisierung der Methode `getConfigOptionGroups()` werden daher einige Konfigurationsoptionen zurückgegeben, von denen einige in Abbildung 6.2 dargestellt sind.

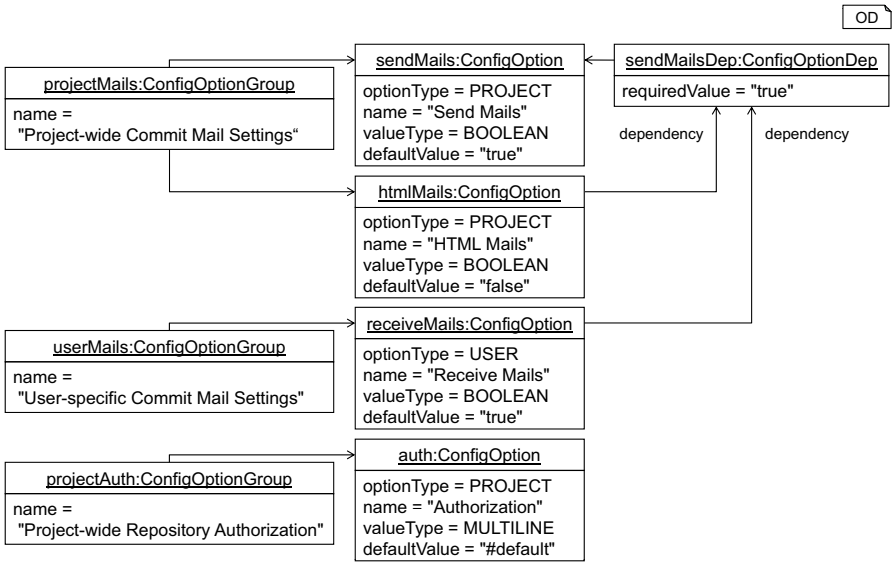


Abbildung 6.2: Konfigurationsoptionen des Subversion-Plug-ins

Die Konfigurationsoptionen eines Subversion-Repositories sind in drei Gruppen aufgeteilt. Über die `sendMails`-Option kann festgelegt werden, ob für ein Projekt überhaupt Commit-Mails versendet werden sollen und über die `htmlMails`-Option, ob diese im HTML-Format oder als reiner Text geschickt werden sollen. Diese beiden Optionen gelten für das gesamte Projekt und sind in der Gruppe `projectMails` zusammengefasst. Über die `receiveMails`-Option kann jeder Benutzer individuell bestimmen, ob er über Commits in das zugehörige Subversion-Repository per Mail informiert werden möchte oder nicht. Diese Option ist in einer eigenen Gruppe für Benutzer enthalten. Schließlich können noch die Zugriffsrechte auf das Repository über die Option `auth` geändert werden.

Die beiden Optionen `htmlMails` und `receiveMails` sind über eine Abhängigkeit an den Wert der Option `sendMails` gebunden, da beide Optionen nur sinnvoll sind, wenn überhaupt Mails verschickt werden. Daher wird über die in der Abbildung gezeigte Abhängigkeit der Wert

`true` für die Option `sendMails` gefordert. Nur wenn dies der aktuelle gesetzte Wert ist, können die anderen Optionen genutzt bzw. geändert werden. Zur Veranschaulichung der beschrieben Optionen sind in Abbildung 6.3 einige Ausschnitte aus der Web-Oberfläche dargestellt, die auf deren Grundlage erzeugt wird.

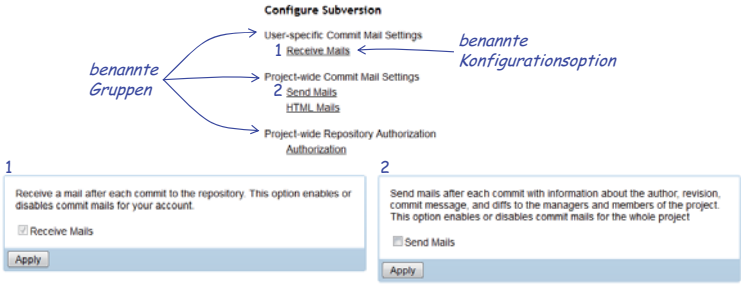


Abbildung 6.3: Ausschnitte aus der Web-Oberfläche zur Konfiguration von Subversion

Im oberen Teil der Abbildung sind die benannten Gruppen mit den Namen ihrer einzelnen Konfigurationsoptionen als Link zu sehen. Beim Klick auf eine der Optionen öffnet sich einer der im unteren Teil der Abbildung gezeigten Dialoge. Die Optionen *Receive Mails* (1) und *Send Mails* (2) legen wie zuvor beschrieben das Commit-Mail-Verhalten fest und sind voneinander abhängig. Im gezeigten Beispiel ist der Wert von *Send Mails* auf `false` gesetzt, so dass *Receive Mails* deaktiviert ist und dadurch nicht geändert werden kann.

Die Realisierung der Methode `setUpService()` im Subversion-Plug-in erhält ein Projekt als Parameter, für das ein Subversion-Repository erstellt werden soll. Die Schritte, die zur Initialisierung und Konfiguration eines Subversion-Repositories von der Methode ausgeführt werden, sind in Abbildung 6.4 in einem Aktivitätsdiagramm dargestellt.

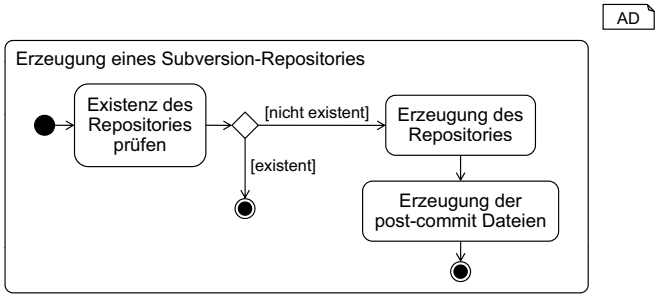


Abbildung 6.4: Ablauf der Erzeugung eines Subversion-Repositories

Zunächst wird überprüft, ob bereits ein Repository für das übergebene Projekt existiert. Ist das der Fall, so wird keine weitere Aktion durchgeführt, da ein existierendes Repository nicht überschrieben werden darf. Falls für das Projekt noch kein Repository existiert, wird es erzeugt.

Im Anschluss werden in das neue Repository alle benötigten Dateien für die Verarbeitung der Commit-Mails unter Berücksichtigung der Default-Werte der zuvor beschriebenen Konfigurationsoptionen geschrieben. Die benötigten Konfigurationsdateien werden dabei aus dem Plug-in geladen.

Die Aktualisierung der Konfiguration von Subversion-Repositories erfolgt durch die Methode `updateServiceConfiguration()`. Der Ablauf der Aktualisierung ist in Abbildung 6.5 in einem Aktivitätsdiagramm dargestellt.

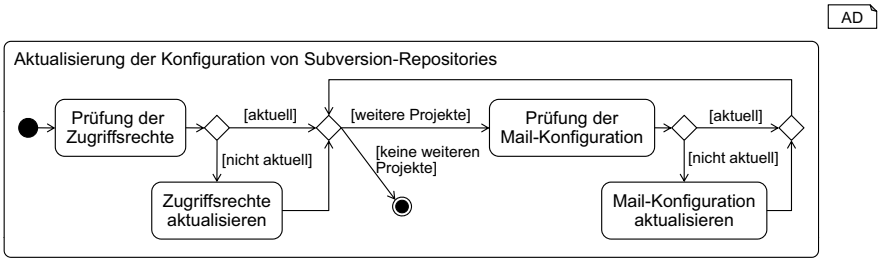


Abbildung 6.5: Ablauf der Aktualisierung eines Subversion-Repositories

Wie in Abschnitt 5.2.2 beschrieben, werden beim Aufruf der Methode alle aktivierten Projekte übergeben – jeweils in dem Zustand vor und nach der Änderung die zum Aufruf geführt hat. Im Subversion-Plug-in werden zunächst die Zugriffsrechte aller Projekte überprüft, die Subversion als Service verwenden. Dies geschieht für alle Projekte und nicht für jedes Projekt individuell, da die Zugriffsrechte für alle Projekte auf einem Backend-Server über eine zentrale Datei konfiguriert werden müssen. Anschließend wird die Commit-Mail-Konfiguration jedes Projekts einzeln geprüft und gegebenenfalls aktualisiert. Dabei wird beispielsweise überprüft, ob neue Benutzer zu dem Projekt hinzugefügt worden sind, ob ein Benutzer seine E-Mail-Adresse geändert hat oder ob die Konfiguration für die Commit-Mails des Projekts geändert worden ist. Gegebenenfalls werden dann die Konfigurationsdateien für die Commit-Mails entsprechend aktualisiert.

Die `tearDownService()`-Methode des Subversion-Plug-ins enthält einen leeren Methodenrumpf. Dies ergibt sich durch die zentrale Konfigurationsdatei für alle Subversion-Repositories. Sobald ein Projekt deaktiviert wird, erfolgt eine Aktualisierung der Datei durch die Methode `updateServiceConfiguration()`. Da das deaktivierte Projekt nicht mehr bei der Erstellung der Datei berücksichtigt wird, ist anschließend kein Zugriff auf das zugehörige Subversion-Repository mehr möglich.

Die Integration von Subversion in das Framework konnte insgesamt mit geringem Aufwand erreicht werden. Insbesondere durch die verfügbaren kommandozeilenbasierten Administrationswerkzeuge und die dateibasierte Konfiguration konnte die Verwaltung der Repositories effektiv automatisiert werden. Zusätzlich ist die Nutzung verbreiteter Web-Server und die Kopplung mit anderen Anwendungen ein großer Vorteil bei der Integration. Als Nachteil hat sich die zentrale Datei zur Konfiguration der Zugriffsrechte herausgestellt. Eine Datei pro Repository könnte effizienter generiert werden und eine striktere Mandantentrennung ermöglichen. Außerdem würden sich Fehler in der Konfiguration nicht auf alle Repositories auswirken, sondern nur auf einzelne.

6.2.2 Trac

Trac [Tra12] wurde ebenfalls als Base-Service in das Framework integriert. Die einzelnen Schritte der Integration sowie die Realisierung des zugehörigen Service-Plug-ins werden in diesem Abschnitt behandelt.

Installation der Anwendung mit ihren Abhängigkeiten. Die Installation von Trac und der benötigten Laufzeitumgebung ist unkompliziert möglich. Trac kann neben anderen Web-Servern ebenfalls Apache HTTPD nutzen. Darüber hinaus kann eine Trac-Instanz entweder mit SQLite, PostgreSQL oder MySQL als Datenbank betrieben werden. Für die Integration in das Framework wurde SQLite gewählt, so dass kein eigenständiger Server für die Trac-Datenbank betrieben werden muss.

Integration mit den Authentifizierungs- und Autorisierungsmechanismen. Trac verfügt über eigene Implementierungen für die Authentifizierung und für das Rechtemanagement. Es ist jedoch auch möglich, die Authentifizierung von Apache HTTPD durchführen zu lassen, so dass auch HTTP Basic Authentication genutzt werden kann. In der in das Framework integrierten Version von Trac müssen jedoch die Benutzer in der Trac-Datenbank vorhanden sein, damit ihnen beispielsweise Tickets zugewiesen werden können. Daher musste zusätzlich ein Skript geschrieben werden, durch das dies automatisiert wird, so dass die Autorisierung innerhalb von Trac korrekt funktioniert.

Anpassung des Erscheinungsbilds. Trac setzt eine Template-Engine für die Erzeugung der Web-Oberfläche ein. Daher kann das Erscheinungsbild einer Trac-Instanz durch die Erstellung von Templates beliebig geändert werden. Bei der Integration von Trac in das Framework wurde ein solches Template erstellt, durch das die GUI-Integration realisiert wird.

Vorbereitung einer automatischen Konfiguration. Trac bietet genau wie Subversion Administrationswerkzeuge und eine dateibasierte Konfiguration an, durch die die Erstellung neuer Trac-Umgebungen, die Vergabe von Rollen und damit assoziierten Rechten sowie die Konfiguration der Funktionen einer Instanz ermöglichen. Daher war auch in diesem Fall keine weitere Vorbereitung notwendig.

Die Umsetzung des Trac-Plug-ins ähnelt der des Subversion-Plug-ins. In diesem Abschnitt werden daher nicht alle Details des Trac-Plug-ins diskutiert, sonder der Fokus auf die Besonderheiten im Vergleich zum Subversion-Plug-in gelegt. In Abbildung 6.6 sind die realisierten Methoden des Trac-Plug-ins dargestellt.

Das Trac-Plug-in basiert ebenfalls auf der Base-Plug-in-API und enthält einige der entsprechenden Methoden. Im Gegensatz zum Subversion-Plug-in bietet das Trac-Plug-in keine Konfigurationsoptionen an. Der Grund dafür ist, dass über die Web-Oberfläche einer Trac-Instanz die wichtigsten Konfigurationsmöglichkeiten den Teilnehmern eines Projekts angeboten werden. Darüber hinaus ist der Zugriff durch Bot-Accounts in der aktuellen Version des Plug-ins ebenfalls nicht erlaubt, so dass die zugehörige Methode nicht überschrieben wird.

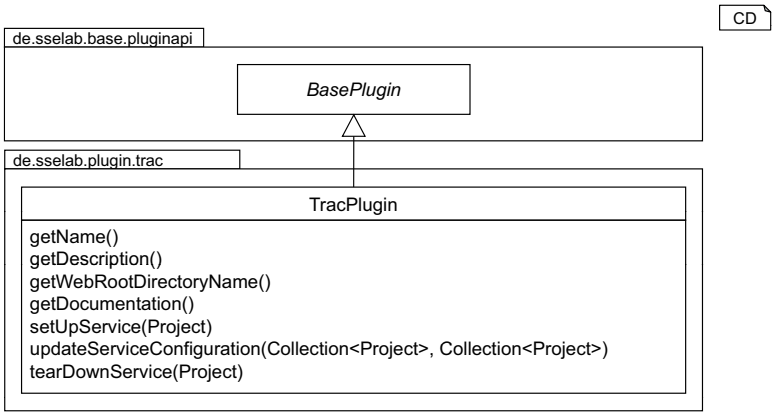


Abbildung 6.6: Oberklasse und realisierte Methoden des Trac-Plug-ins

Die `setUpService()`-Methode wird im Plug-in realisiert, um für ein Projekt eine Trac-Instanz erzeugen zu können. Die Schritte, die bei der Ausführung der Methode durchgeführt werden, sind in Abbildung 6.7 in einem Aktivitätsdiagramm zusammengefasst.

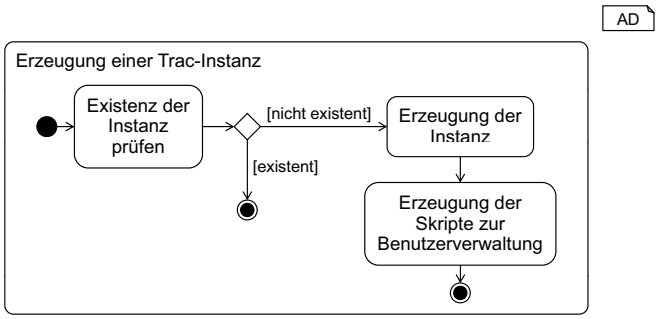


Abbildung 6.7: Ablauf der Erzeugung einer Trac-Instanz

Wie auch bei Subversion, wird zunächst überprüft, ob für das übergebene Projekt bereits eine Trac-Instanz existiert. Falls nicht, wird eine neue Instanz erstellt. Anschließend wird ein Skript aus dem Service-Plug-in geladen, das eine Aktualisierung der Benutzerinformationen in der Datenbank einer Trac-Instanz ermöglicht. Da für die Integration in das Framework das Rechtemanagementmodul von Trac nicht genutzt wird, werden die Benutzer über dieses Skript angelegt, aktualisiert oder gelöscht. Das Skript basiert auf einigen APIs von Trac für den Zugriff auf die Datenbank.

Die Aktualisierung von Trac-Instanzen, die unter anderem bei Änderungen an Projekten oder Benutzerkonten angestoßen wird, erfolgt über die Methode `updateServiceConfigura-`

`tion()`. Die dabei durchgeführten Schritte sind in dem Aktivitätsdiagramm in Abbildung 6.8 dargestellt und werden nachfolgend kurz beschrieben.

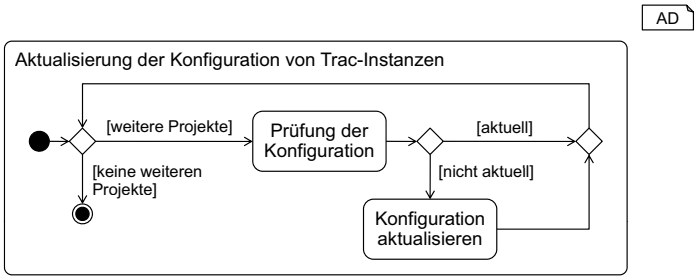


Abbildung 6.8: Ablauf der Aktualisierung einer Trac-Instanz

Im Gegensatz zu Subversion wird bei Trac jede Instanz eigenständig konfiguriert, so dass die Aktualisierung eines jeden Projekts unabhängig von den anderen erfolgt. Zunächst wird im Plug-in geprüft, ob eine Aktualisierung einer Trac-Instanz überhaupt notwendig ist. Im Fall von Trac muss eine Aktualisierung durchgeführt werden, wenn Benutzer zu dem Projekt hinzugefügt oder vom Projekt entfernt worden sind, wenn sich die primäre E-Mail-Adresse eines Projektteilnehmers geändert hat oder nach der Erzeugung einer neuen Trac-Instanz. Dabei wird zunächst die Konfigurationsdatei für die Autorisierung des Zugriffs auf die Instanz aktualisiert, so dass alle Projektteilnehmer Zugriff erhalten. Anschließend wird das aus dem Plug-in geladene Skript aufgerufen, so dass neue Benutzer in die Datenbank geschrieben, nicht mehr im Projekt enthaltene Benutzer entfernt und die E-Mail-Adressen der Benutzer aktualisiert werden können. Alle diese Aktionen werden nur dann ausgeführt, wenn sie tatsächlich notwendig sind. Weiterhin wird überprüft, ob Subversion im aktuellen Projekt genutzt wird und bei Bedarf die Integration von Trac mit Subversion konfiguriert. Analog könnte an dieser Stelle auch die Git-Integration von Trac konfiguriert werden.

Durch die `tearDownService()`-Methode wird jeder weitere Zugriff auf eine Trac-Instanz verhindert. Die Aktionen, die bei der Ausführung der Methode ausgeführt werden, sind in Abbildung 6.9 zusammengefasst.

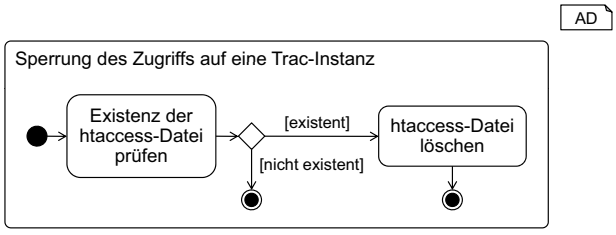


Abbildung 6.9: Sperrung des Zugriffs auf eine Trac-Instanz

Der Zugriff auf alle Trac-Instanzen wird in einer zentralen Konfigurationsdatei nicht gestattet. In projektspezifischen Dateien kann diese Konfiguration überschrieben und der Zugriff für die darin explizit aufgeführten Projektteilnehmer ermöglicht werden. Durch diese Art der Konfiguration muss durch die `tearDownService()`-Methode nur die projektspezifische Datei entfernt werden, so dass kein Zugriff auf eine Trac-Instanz mehr möglich ist.

Die Integration von Trac in das Framework war in Bezug auf den Aufwand vergleichbar mit der Integration von Subversion. Dies liegt unter anderem an der guten Unterstützung einer automatischen Konfiguration durch Administrationswerkzeuge und Konfigurationsdateien. Hervorzuheben ist dabei auch die Möglichkeit, für alle Trac-Instanzen eine zentrale Konfigurationsdatei bereitstellen und diese in instanzspezifischen Dateien erweitern zu können. Dies erlaubt eine flexible und effiziente Konfiguration aller Trac-Instanzen. Außerdem existieren viele Erweiterungen von Trac, die zur Laufzeit des Systems installiert, konfiguriert und aktiviert werden können. Die Unterstützung gängiger Web- und Datenbank-Server ist ebenfalls ein Pluspunkt. Bei der Integration von Trac wurde SQLite als Datenbank gewählt, da so jede Trac-Instanz über eine von den anderen unabhängige Datenbank verfügt und die Mandantentrennung vereinfacht wird. Eine Einschränkung dabei ist jedoch, dass die Performanz nicht so gut wie bei den Alternativen ist. Zuletzt ist auch der Einsatz einer Template-Engine für die Anpassung des Erscheinungsbilds ein positiver Aspekt von Trac. Ein Nachteil ist die fehlende API zur automatischen Verwaltung von Benutzerkonten in der Datenbank. Daher musste im Zuge der Integration ein Skript auf Grundlage interner Trac-APIs entwickelt werden. Da die Stabilität solcher interner APIs jedoch nicht gewährleistet ist, wird eine Aktualisierung dadurch unnötig erschwert.

6.2.3 Jenkins

Jenkins [Sma11] ist ein frei verfügbarer Continuous Integration Server. Die wichtigsten Schritte zur Integration von Jenkins in das Framework werden im Folgenden zusammengefasst. Anschließend wird die Realisierung des Service-Plug-ins diskutiert.

Installation der Anwendung mit ihren Abhängigkeiten. Jenkins ist eine Java-basierte Web-Anwendung und bringt einen eigenen Server mit, kann jedoch auch in einem beliebigen Servlet-Container ausgeführt werden. Für die Integration in das Framework wurde Tomcat als Container gewählt. Sowohl die Installation von Tomcat als auch von Jenkins war auf der Zielplattform problemlos möglich. Durch die Verwendung eines eigenen Servers bzw. eines Servlet-Containers muss jedoch eine Instanz von Jenkins für alle Projekte eines Backend-Servers genutzt werden.

Integration mit den Authentifizierungs- und Autorisierungsmechanismen. Jenkins verfügt genau wie Trac über eigene Implementierungen zur Authentifizierung und Autorisierung. Da jedoch eine Vielzahl von Erweiterungen für Jenkins verfügbar sind, konnte unter Verwendung einer dieser Erweiterungen Jenkins so angepasst werden, dass HTTP Basic Authentication und damit Single-Sign-On unterstützt werden. Die Zugriffsrechte können in Jenkins relativ feingranular konfiguriert werden. Jedoch sind die Mechanismen auf eine manuelle und nicht auf eine automatisierte Konfiguration ausgelegt. Daher mussten einige Verwaltungsmechanismen innerhalb von Jenkins abgeschaltet und über die Konfigurationsoptionen des zugehörige Service-Plug-ins ausgelagert werden.

Anpassung des Erscheinungsbilds. Jenkins erlaubt nur eine minimale Anpassung des Erscheinungsbilds, beispielsweise über das Einbinden von Grafiken. Darüber hinaus existieren einige Erweiterungen, mit deren Hilfe bestimmte Aspekte der Oberfläche angepasst werden können. Dennoch wird eine Modifikation der Oberfläche insgesamt nicht ausreichend unterstützt.

Vorbereitung einer automatischen Konfiguration. Die Konfiguration von Jenkins erfolgt teilweise über XML-Dateien und teilweise über eine REST-API, über die einige administrative Funktionen angeboten werden. Auf Grundlage dieser API wurde das im Folgenden beschriebene Service-Plug-in erstellt.

Die Beziehungen zu einigen Klassen und die realisierten Methoden des Jenkins-Plug-ins zeigt Abbildung 6.10. Das Plug-in ist genau wie die bisher vorgestellten als Base-Service-Plug-in realisiert.

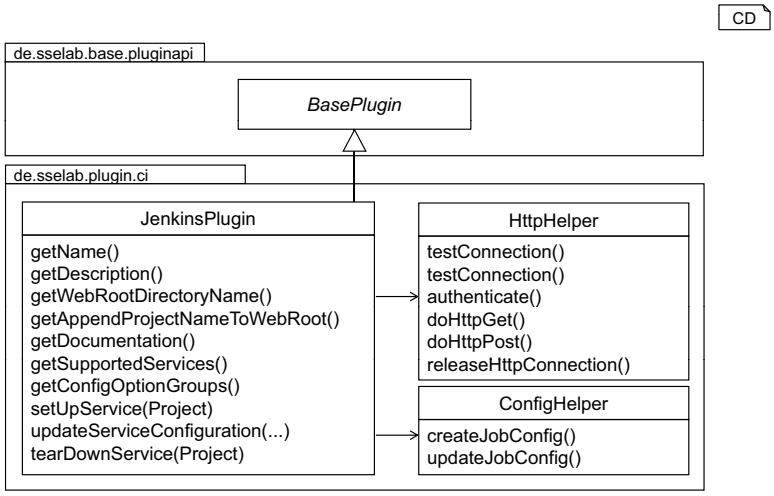


Abbildung 6.10: Oberklasse, realisierte Methoden und Hilfsklassen des Jenkins-Plug-ins

Die Implementierung des Jenkins-Plug-ins erfolgt in drei Klassen. Die `HttpHelper`-Klasse enthält Hilfsmethoden für den Zugriff auf die REST-API von Jenkins. Dazu gehören beispielsweise Methoden, um eine Verbindung zu einer Jenkins-Instanz zu testen, die Authentifizierung mit einem Account zur Konfiguration durchzuführen und mehrere Methoden, um GET- und POST-Anfragen auszuführen. Die Klasse `ConfigHelper` stellt Hilfsmethoden für die Verarbeitung der XML-basierten Konfigurationsdateien für die Zugriffsrechte von Jenkins bereit. Diese können über die REST-API von Jenkins abgerufen, modifiziert und schließlich wieder an Jenkins zur Aktualisierung gesendet werden. Durch die `JenkinsPlugin`-Klasse werden Methoden der Plug-in-API auf Grundlage der anderen Klassen implementiert. Im Gegensatz zu Subversion und Trac wird im Plug-in die `getAppendProjectNameToWebRoot()`-Methode überschrieben,

da nur eine Instanz von Jenkins pro Backend-Server existiert. Außerdem wird über die Methode `getSupportedServices()` die Liste der Services zurückgegeben, auf die mit Hilfe von Bot-Accounts zugegriffen werden soll. Über die Konfigurationsoptionen werden die sogenannten Jobs – also die Build-Prozesse für ein Projekt – konfiguriert. Standardmäßig wird zunächst ein Job mit dem Namen des Projekts angelegt, der dann über die Web-Oberfläche von Jenkins konfiguriert werden kann. Über die Konfigurationsoptionen können dann weitere Jobs erzeugt und existierende aktualisiert oder deaktiviert werden.

Da der Jenkins-Server bereits laufen muss, bevor er durch das Service-Plug-in konfiguriert werden kann, enthält die `setUpService()`-Methode des Plug-ins einen leeren Rumpf. Die `tearDownService()`-Methode ist ebenfalls leer, da die gesamte Konfiguration in der `updateServiceConfiguration()`-Methode durchgeführt wird. Die einzelnen Schritte dieser Konfiguration sind in Abbildung 6.11 in einem Aktivitätsdiagramm dargestellt.

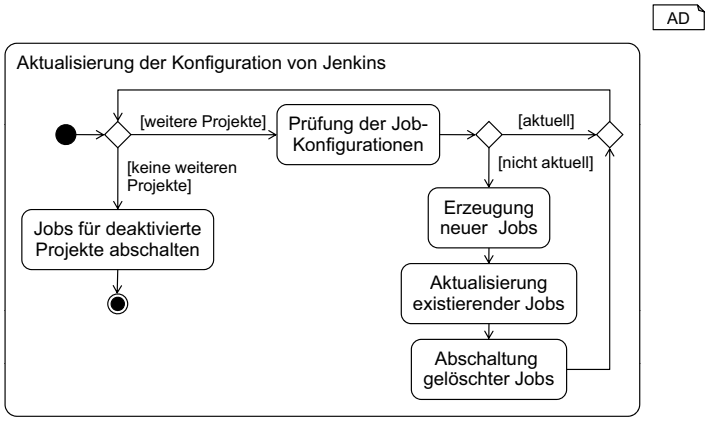


Abbildung 6.11: Ablauf der Aktualisierung von Jenkins

Für jedes Projekt wird überprüft, ob eine Aktualisierung der Job-Konfiguration notwendig ist. Wenn das der Fall ist, werden gemäß der Konfiguration des Projekts alle benötigten neuen Jobs erzeugt, die existierenden Jobs aktualisiert und gelöschte Jobs deaktiviert. Jobs werden aktuell nicht gelöscht, da die teilweise umfangreiche Konfiguration, die nach der automatischen Erstellung durch das Plug-in direkt in Jenkins durch die Projektteilnehmer erfolgen kann, erhalten bleiben soll.

Jenkins verfügt über eine aktive Entwicklergruppe, die regelmäßig und in kurzen Abständen neue Versionen bereitstellt. Darüber hinaus gibt es viele Erweiterungen, durch die Jenkins angepasst werden kann. Insgesamt hat sich jedoch die Integration von Jenkins in das Framework als schwierig erwiesen. Ein Grund dafür ist ein Mangel an APIs zur automatischen Konfiguration. Jenkins bietet zwar eine REST-API an, jedoch sind viele Konfigurationsoptionen nicht aufrufbar. Der Fokus auf eine interaktive Nutzung und Konfiguration des Systems erschwert die Integration in Frameworks wie das SSELab erheblich. Darüber hinaus ist die Verwaltung der Zugriffsrechte feingranular und komplex. Außerdem ist Jenkins nicht optimal für einen Multi-Projektbetrieb

mit strikter Mandantentrennung geeignet, bei dem mehrere unabhängige Projekte eine einzelne Instanz verwenden. Daher musste auch die Erstellung von Jobs durch Konfigurationsoptionen in das Plug-in ausgelagert werden. Schließlich ist auch eine Anpassung des Erscheinungsbilds nur unzureichend möglich.

6.2.4 Feedback-System

Das Feedback-System [HKR12a] wird im Rahmen einer Promotion am Lehrstuhl für Software Engineering entwickelt. Das System kann eingesetzt werden, damit die Endnutzer eines Softwareprodukts auf einfache Weise Fehler oder Anmerkungen an dessen Entwickler senden können. Das Feedback-System besteht aus einer serverbasierten Anwendung und mehreren Clients für verschiedene Zielplattformen. Beispielsweise existieren ein Client für Eclipse-RCP-Anwendungen, ein JavaScript-Client für Web-Anwendungen und ein Android-Client. Mit Hilfe dieser Clients kann ein Endnutzer Screenshots vom Softwareprodukt erstellen, diese noch annotieren und zusammen mit einem Kommentar an die Serverkomponente schicken. Der Server verbindet sich dann mit einem Bug-Tracking-System und erstellt ein Ticket mit dem Screenshot und dem Kommentar für die Entwickler.

Das System kann in verschiedenen Phasen eines Entwicklungsprojekts eingesetzt werden. Beispielsweise zur Anforderungserhebung in frühen Projektphasen, zur Durchführung von Abnahmetests vor einer Auslieferung oder zur Unterstützung der Kunden, um Anmerkungen an die Entwickler zu schicken. Die wichtigsten Komponenten der Architektur des Systems zeigt Abbildung 6.12.

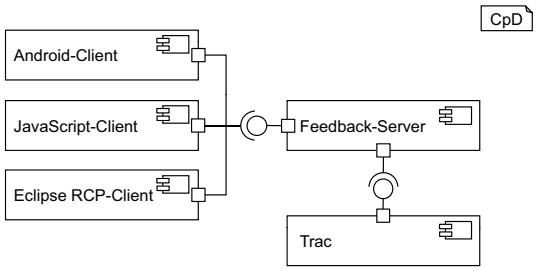


Abbildung 6.12: Architektur des Feedback-Systems

Der Feedback-Server stellt über eine Web-Service-Schnittstelle alle Funktionen für die Clients bereit. Zusätzlich ermöglicht diese Komponente den Zugriff auf verschiedene Bug-Tracking-Systeme. In der Version, die in das Framework integriert worden ist, kommt Trac zum Einsatz. Die Clients des Feedback-Systems greifen jeweils über den Web-Service auf die verfügbaren Funktionen zu und realisieren die clientspezifischen Benutzeroberflächen. Sie verwenden jeweils Konfigurationsdateien, über die die konkrete Trac-Instanz festgelegt ist, in der das Feedback gespeichert werden soll.

Das Feedback-System wurde von Anfang an auf eine Integration in das SSELab-Framework ausgelegt, so dass diese entsprechend effizient umzusetzen war. Die einzelnen Schritte werden

im Folgenden kurz skizziert, bevor im Anschluss einige Aspekte des Service-Plug-ins vorgestellt werden.

Installation der Anwendung mit ihren Abhängigkeiten. Die serverseitige Komponente des Feedback-Systems benötigt einen Servlet-Container zur Ausführung. Durch die Laufzeitumgebung der Framework-Instanz steht ein solcher Container bereits zur Verfügung, so dass die Installation ohne Weiteres möglich ist.

Integration mit den Authentifizierungs- und Autorisierungsmechanismen. Die Benutzer der Clients des Feedback-Systems müssen sich nicht authentifizieren, sondern können anonym Feedback geben. Damit das Feedback vom Server richtig zugeordnet werden kann, wird jeder Client für ein bestimmtes Projekt konfiguriert. Die Serverkomponente verwendet dann die Autorisierungsmechanismen der entsprechenden Framework-Instanz für den Zugriff auf das Projekt.

Anpassung des Erscheinungsbilds. Die Serverkomponente des Feedback-Systems verfügt über keine Web-Oberfläche, so dass eine Anpassung nicht notwendig ist.

Vorbereitung einer automatischen Konfiguration. Nach der Installation der Serverkomponente müssen nur noch die Clients automatisch konfiguriert werden können. Die Mechanismen dafür werden im Folgenden im Kontext des Service-Plug-ins beschrieben.

Das Feedback-System ist ebenfalls als Base-Service realisiert. Die Methoden des zugehörigen Plug-ins sowie einige Hilfsklassen sind in Abbildung 6.13 gezeigt.

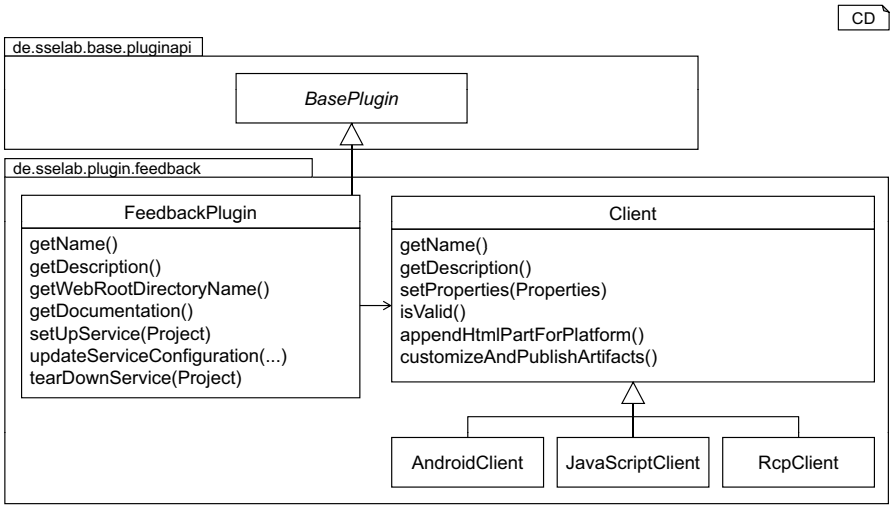


Abbildung 6.13: Oberklasse, realisierte Methoden und Hilfsklassen des Feedback-Plug-ins

Die drei beschriebenen Client-Arten, die jeweils durch eine Subklasse von `Client` repräsentiert werden, werden bei der Installation des Plug-ins zunächst initialisiert. Nach der erfolgreichen Initialisierung können der Service und damit die Clients für ein Projekt über die `setUpService()`-Methode bereitgestellt werden. Die in der Methode durchgeführten Schritte sind in Abbildung 6.14 gezeigt.

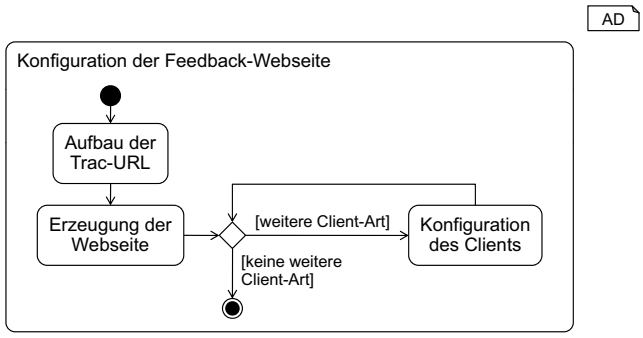


Abbildung 6.14: Ablauf der Erzeugung der Webseite und der Clients des Feedback-Services

Bei der Konfiguration des Services für ein Projekt wird eine Webseite generiert, auf der die für das Projekt konfigurierten Clients verlinkt sind. Zunächst wird dazu die Trac-URL bestimmt, die zu der Trac-Instanz des Projekts gehört. Da der Feedback-Service eine Abhängigkeit auf den Trac-Service besitzt, ist in jedem Fall eine Trac-Instanz vorhanden, so dass die URL immer gebildet werden kann. Anschließend wird die Webseite generiert und schließlich die Clients für das Projekt konfiguriert. Dabei werden die Trac-URL und ein zugehörigen Hashwert in Dateien der Clients geschrieben. In den `updateServiceConfiguration()`- und der `tearDownService()`-Methoden werden dann nur noch die Dateien zur Konfiguration der Zugriffsrechte aktualisiert bzw. gelöscht.

Da das Feedback-System eine Eigenentwicklung des Lehrstuhls für Software Engineering ist und es von Anfang an auch auf die Integration in das Framework ausgerichtet war, konnte diese insgesamt problemlos umgesetzt werden.

6.2.5 MontiCore

MontiCore [Kra10] ist ein Java-basiertes Framework für die Entwicklung domänenspezifischer Sprachen (DSLs), das am Lehrstuhl für Software Engineering entwickelt wird. Es ist ein dateiverarbeitendes Werkzeug, so dass es als Ostp-Service in das Framework integriert worden ist. In der Methodik zur Integration solcher Services wurden neben der Auswahl der Ostp-Service-Kategorie und Realisierung des Service-Plug-ins die folgenden beiden Schritte hervorgehoben.

Installation des Werkzeugs. In Abschnitt 5.2.3 wurden die verschiedenen Möglichkeiten zur Installation eines Werkzeugs diskutiert. Im Fall von MontiCore bietet sich eine Installation

in den Backend-Instanzen an, da durch die vorhandene Eclipse-Integration alle MontiCore-Komponenten bereits als Backend-kompatible Plug-ins zur Verfügung stehen und damit ohne weitere Modifikation darin installiert werden können.

Zugriff auf das Werkzeug ermöglichen. Der Zugriff auf MontiCore erfolgt nach der Installation in eine Backend-Instanz über die Mechanismen der OSGi-Spezifikation zur Kopplung von Plug-ins und damit letztendlich über API-Aufrufe von MontiCore durch das im Folgenden beschriebenen Service-Plug-in.

Insgesamt ergibt sich durch die Integration von MontiCore in das Framework die in Abbildung 6.15 dargestellte Verteilung der Laufzeitkomponenten.

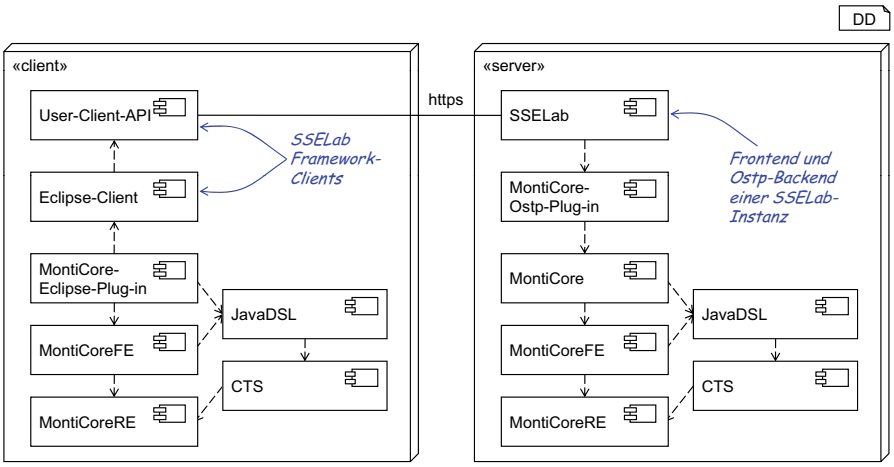


Abbildung 6.15: Laufzeitkomponenten von MontiCore (adaptiert aus [Kra10])

Auf der Serverseite wird der Generator (Komponente MontiCore) und dessen Abhängigkeiten (Komponenten MontiCoreFE, MontiCoreRE, JavaDSL und CTS) installiert. Bei der Installation dieser Komponenten in die Ostp-Backends wird automatisch erkannt, dass es sich nicht um Service-Plug-ins handelt. Auf Grundlage dieser Komponenten ist das Service-Plug-in für MontiCore realisiert (MontiCore-Ostp-Plug-in). Auf der Clientseite werden die Laufzeitabhängigkeiten für die Integration in Eclipse wiederverwendet. Zusätzlich sind auf der Clientseite noch das Eclipse-Plug-in von MontiCore und dessen zusätzliche Abhängigkeiten (in der Abbildung nicht gezeigt) installiert. Dieses erweitert den Eclipse-Client des SSELab-Frameworks und nutzt dessen API zum Aufruf des im Ostp-Backend installierten Generators. Wenn der Generator auch auf der Clientseite installiert ist, wird dies automatisch erkannt und es erfolgt kein Aufruf des Servers.

In Abschnitt 5.2.3 wurden die vier verschiedenen Kategorien von Ostp-Services behandelt. MontiCore selbst wurde als Transformation-Service in das Framework integriert. Abbildung 6.16 zeigt die Klassen des Service-Plug-ins sowie einige Klassen aus MontiCore.

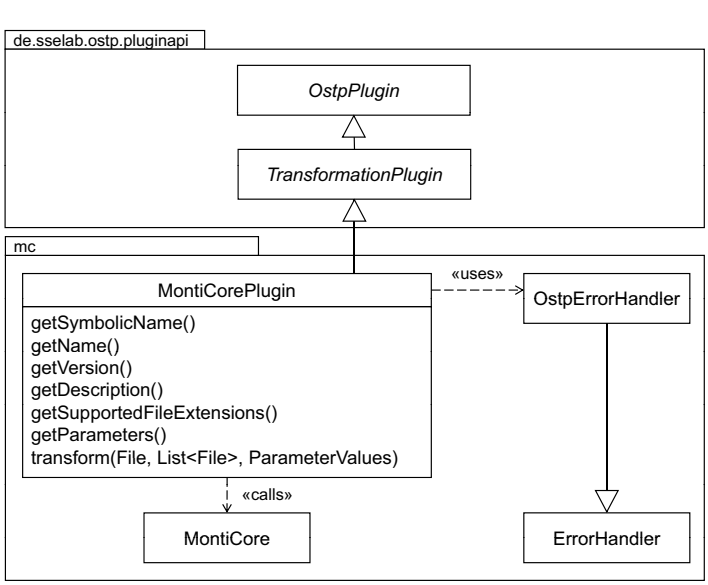


Abbildung 6.16: Oberklassen und realisierte Methoden des MontiCore-Plug-ins sowie einige MontiCore-Klassen

Die Klasse `MontiCorePlugin` erbt von `TransformationPlugin` aus der Ostp-Plug-in-API und überschreibt deren Methoden. Im MontiCore-Plug-in werden der symbolische Name, der für Benutzer sichtbare Name, die Version und die Beschreibung des Services definiert. Zusätzlich werden die Dateinamenserweiterungen der unterstützten Eingabeformate bestimmt. Die Parametrisierungsmöglichkeiten von MontiCore werden über die Implementierung der Methode `getParameters` gesteuert. In der Methode werden zwei Parameter erzeugt. In Abbildung 6.17 sind diese in einem Objektdiagramm dargestellt.

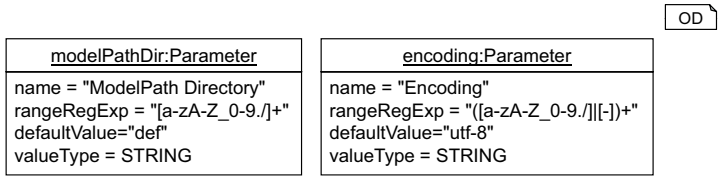


Abbildung 6.17: Parameter des MontiCore-Plug-ins

Über die `Parameter`-Objekte kann der Endnutzer einerseits den Modelpath, also den Pfad in dem sich Eingabemodelle befinden, und andererseits das Encoding beim Aufruf von MontiCore bestimmen. Der Aufruf von MontiCore selbst erfolgt in der Implementierung der `trans-`

`form()`-Methode im Plug-in. In Quellcode 6.18 ist die vollständige Implementierung der Methode gezeigt, um ein Gefühl für den Umfang der Plug-in-Realisierung zu vermitteln.

```
1  public ExecutionMessages transform(
2      File projectRootDir, List<File> inputFiles, ParameterValues parameters) throws
           Exception {
3
4    String outputDirectory = projectRootDir.getAbsolutePath();
5    String modelpathDirectory = parameters.getValue(MCG.PARAM_MODEL_PATH_DIR_NAME);
6    String encoding = parameters.getValue(MCG.PARAM_ENCODING_NAME);
7
8    ArrayList<String> argsList = new ArrayList<String>();
9    argsList.add("-o");
10   argsList.add(outputDirectory);
11   argsList.add("-conf");
12   argsList.add(outputDirectory + "/" + Constants.CONFIGURATIONFILE_NAME);
13   argsList.add("-mp");
14   argsList.add(outputDirectory + "/" + modelpathDirectory);
15   argsList.add("-encoding");
16   argsList.add(encoding);
17   for (File inputFile : getGeneratorFiles(inputFiles, projectRootDir)) {
18     argsList.add(inputFile.getCanonicalPath());
19   }
20   String[] args = argsList.toArray(new String[argsList.size()]);
21
22   OSTPErrorHandler errorHandler = new OSTPErrorHandler();
23   ExecutionMessages messages = new ExecutionMessages();
24
25   MontiCore m = new MontiCore(args);
26   m.addErrorHandler(errorHandler);
27   m.run();
28   messages.addMessages(errorHandler.getMessages());
29   return messages;
30 }
```

Quellcode 6.18: Implementierung der `transform()`-Methode im MontiCore-Plugin

In der Implementierung der Methode werden zunächst die Parameter für den Aufruf von MontiCore auf Grundlage der übergebenen Parameterwerte aufbereitet. Danach werden alle Eingabedateien für MontiCore aus der übertragenen Verzeichnishierarchie bestimmt und schließlich MontiCore mit diesen Argumenten aufgerufen. Danach werden die von MontiCore bei der Ausführung produzierten Meldungen gespeichert und schließlich zurückgegeben. Die Verarbeitung der generierten Dateien erfolgt automatisch durch das Ostp-Backend.

MontiCore ist eine Eigenentwicklung des Lehrstuhls, wurde jedoch im Gegensatz zum Feedback-System bereits vor der Erstellung des SSELab-Frameworks realisiert. Die Integration in das Framework wurde also erst nachträglich umgesetzt. Von Vorteil war dabei, dass durch die Wahl von Eclipse als zu unterstützende IDE alle Komponenten von MontiCore bereits als kompatible Plug-ins verfügbar waren. Außerdem konnte durch das modulare Design von MontiCore eine minimale Menge von Komponenten auf der Serverseite installiert, diese auch auf der Clientseite wiederverwendet und insgesamt eine vollständig transparente Nutzung des Ostp-Services erreicht werden [Kra10], so dass die Praxistauglichkeit der Konzepte zur servicebasierten Nutzung und IDE-Integration dateiverarbeitender Werkzeuge gezeigt werden konnte.

6.2.6 WikiBot

Der WikiBot-Service ist ebenfalls eine Eigenentwicklung des Lehrstuhls für Software Engineering. Dieser Service besteht aus zwei Ostp-Plug-ins und einem Kommandozeilen-Client, der den Zugriff auf die Funktionalität beider Plug-ins ermöglicht. Mit Hilfe des Clients können Wiki-Artikel aus einem MediaWiki einer SSELab-Instanz als Textdateien heruntergeladen werden. Anschließend können diese Dateien beliebig geändert und wieder hochgeladen werden, um die Artikel zu aktualisieren. Außerdem können die verlinkten Dateien eines Artikels, wie beispielsweise Bilder, über den Client ebenfalls verwaltet werden.

Installation des Werkzeugs. Die Installation der entwickelten API für beide Plug-ins wird analog zu der Integration von MontiCore direkt in die Ostp-Backends installiert. Diese API wurde von Anfang an für diese Art der Integration vorbereitet.

Zugriff auf das Werkzeug ermöglichen. Die API kann dann über die Mechanismen des OSGi-Containers von den beiden Service-Plug-ins geladen und aufgerufen werden. Die Funktionalität beider Plug-ins wird im Folgenden beschrieben.

Das Service-Plug-in zum Herunterladen von Wiki-Artikeln besteht aus der Klasse `WikiBotExtractorPlugin` und ist als Extraction-Service realisiert. Das Plug-in zum Hochladen von Wiki-Artikeln und Dateien enthält die Klasse `WikiBotPublisherPlugin` und wurde als Distribution-Service entwickelt. Abbildung 6.19 zeigt diese beiden Klassen in einem Klassendiagramm mit ihren Oberklassen.

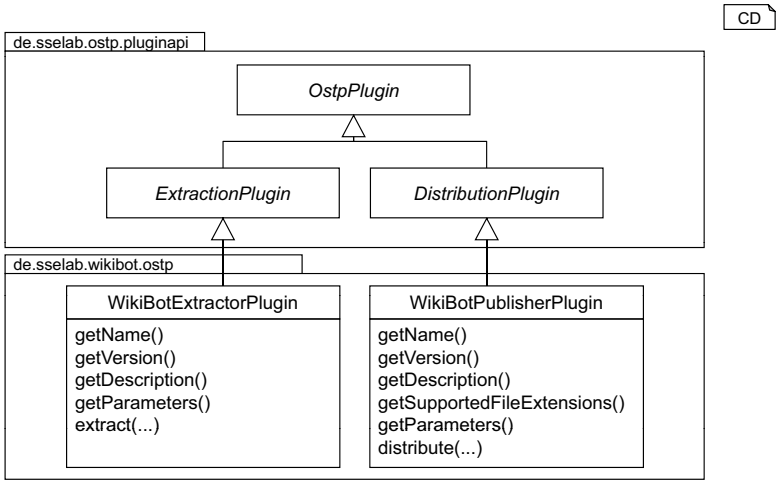


Abbildung 6.19: Implementierte Methoden der WikiBot-Plug-ins

Beide Klassen implementieren die Methoden zur Definition des Namens, der Versionsnummer und der Beschreibung des Services. Da die Klasse `WikiBotExtractorPlugin` keine

Eingabedateien verarbeitet, überschreibt sie auch nicht die Methode `getSupportedFileExtensions()`. Die Klasse `WikiBotPublisherPlugin` gibt dagegen über diese Methode die Dateinamenserweiterungen der unterstützten Dateiformate zurück. Von beiden Klassen werden Parameter definiert, mit deren Hilfe der Zugriff auf ein MediaWiki einer Framework-Instanz ermöglicht wird. Abbildung 6.20 zeigt diese Parameter in einem Objektdiagramm.

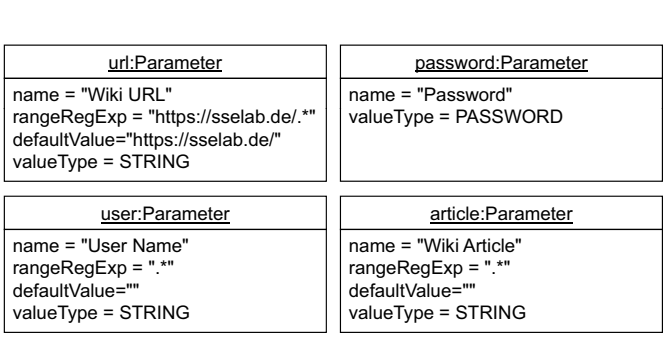


Abbildung 6.20: Parameter der WikiBot-Plug-ins

Beide Plug-ins verwenden die Objekte `url`, `user` und `password`, über die die URL auf ein MediaWiki sowie der Benutzername zum Login und das zugehörige Passwort angegeben werden können. Das Plug-in zum Herunterladen von Artikeln definiert zusätzlich den `article`-Parameter, über den der Name des Artikels bestimmt werden kann. Die Implementierung der beiden Methoden `extract()` und `distribute()` im jeweiligen Plug-in basiert auf dem Aufruf einer gemeinsam genutzten API für den Zugriff auf Wikis. Quellcode 6.21 zeigt zur Veranschaulichung die Implementierung der Methode `extract()`.

```
1  public ExecutionResult extract(ParameterValues parameterValues, File outputDir)
2      throws Exception {
3    String wikiUrl = parameterValues.getValue(WIKI_URL);
4    String userName = parameterValues.getValue(USER_NAME);
5    String password = parameterValues.getValue(PASSWORD);
6    String articleName = parameterValues.getValue(WIKI_ARTICLE);
7
8    MediaWikiConnector connector = new MediaWikiConnector(wikiUrl, userName, password);
9    String articleText = connector.getArticleText(articleName);
10
11   File outputFile = FileUtils.createNewFile(outputDir, articleName + ".txt");
12   FileUtils.writeStringToFile(outputFile, articleText);
13
14   ExecutionResult result = new ExecutionResult();
15   result.addOutputFiles(outputDir);
16   return result;
17 }
```

Quellcode 6.21: Implementierung der `extract()`-Methode im WikiBot-Extractor-Plug-in

In der Methode werden die Parameterwerte für die Instanziierung eines Konnektors genutzt, über den dann der Inhalt des Artikels mit dem übergebenen Namen abgerufen wird. Anschließend wird der Inhalt in eine Textdatei mit dem Namen des Artikels geschrieben und zurückgegeben. In der hier nicht gezeigten Implementierung der `distribute()`-Methode wird ebenfalls ein Konnektor erzeugt, dann aber zum Speichern des übergebenen Artikels oder der verlinkten Dateien genutzt.

Für die beiden Service-Plug-ins wurde ein Kommandozeilen-Client entwickelt, der deren Funktionalität bereitstellt. Dazu wurde die in Abschnitt 5.3.1 beschriebene User-Client-API genutzt. Abbildung 6.22 zeigt diese und die weiteren Komponenten, aus denen der Client aufgebaut ist.

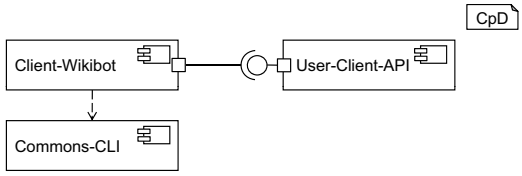


Abbildung 6.22: Komponenten des Clients der WikiBot-Plug-ins

Die Komponente User-Client-API ermöglicht den Zugriff auf eine Framework-Instanz und wird von der Komponente Client-WikiBot verwendet, die zusätzlich noch Commons-CLI zur Erzeugung der Kommandozeilenschnittstelle nutzt. In Abbildung 6.23 ist der Aufruf des Clients ohne Argumente über die Kommandozeile und die resultierende Ausgabe eines Hilfetextes gezeigt.

```
sselab-desktop:~# java -jar sselab-client-wikibot-0.9.1.jar
Please specify one of the extract or publish options.

usage: java -jar ostp-client-wikibot-0.9.1.jar [OPTION]... [[FILE]...|[ARTICLE]]
 -extract              extract an article from a SSELab MediaWiki
 -help                 print this message
 -o <output folder>    the folder where the output files should be stored
 -p <password>         the password for the SSELab account
 -publish              publish the content of txt files to a SSELab MediaWiki
 -u <user name>        the SSELab account user name to use
 -w <wiki url>         the url to the SSELab MediaWiki
```

Quellcode 6.23: Ausgabe eines Hilfetextes nach Aufruf des WikiBot-Clients ohne Parameter

Der WikiBot-Service ist eine Eigenentwicklung des Lehrstuhls, die in kurzer Zeit auf Grundlage des Frameworks und verschiedener APIs erstellt wurde. Hervorzuheben ist dabei die geringe Menge an Code, die für die Bereitstellung der Funktionalität auf Grundlage der APIs und integrierten Anwendungen notwendig war. Schwierigkeiten ergaben sich nur durch die teilweise mangelhafte Dokumentation der verwendeten externen APIs.

6.2.7 Google

Zuletzt wird in diesem Abschnitt die Entwicklung eines Social-Plug-ins für die OpenSocial-Container von Google (z.B. iGoogle, Gmail) behandelt. Google erlaubt die Registrierung von Anwendungen für den Zugriff auf die Container mit Hilfe eines gültigen Google-Accounts. Nach der Registrierung wird ein sogenannter Consumer-Key sowie ein Consumer-Secret erzeugt, die bei der Authentifizierung bei den Containern mittels OAuth verwendet werden müssen. Da die Social-Plug-in-API bereits Unterstützung für OpenSocial und OAuth anbietet, hat der Code des Service-Plug-ins für einen entsprechend geringen Umfang. Abbildung 6.24 zeigt die Klasse `GooglePlugin` sowie die Oberklassen der Social-Plug-in-API.

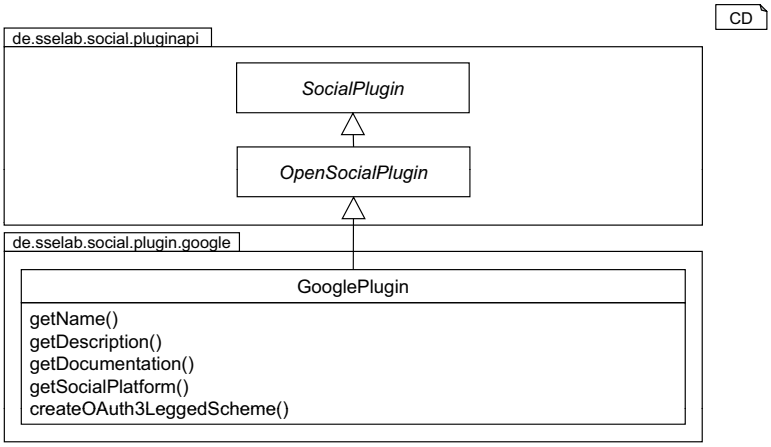


Abbildung 6.24: Oberklassen und realisierte Methoden des Google-Plug-ins

In der Klasse werden die Methoden zur Bestimmung des Namens und der Beschreibung des Services implementiert sowie Dokumentation bereitgestellt. Zusätzlich wird über die Methode `getSocialPlatform()` der Name der Plattform gesetzt, von der Profilinformationen abgerufen werden – in diesem Fall „Google Profiles“. Schließlich wird die Methode zur Authentifizierung auf Grundlage der OpenSocial API implementiert. Quellcode 6.25 zeigt die Implementierung dieser Methode, in deren Rumpf lediglich ein entsprechender Provider erzeugt und zusammen mit dem generierten Consumer-Key und dem zugehörigen Consumer-Secret zur Erzeugung des OAuth-Verfahrens verwendet wird.

```
1 protected OAuth3LeggedScheme createOAuth3LeggedScheme() {
2   return new OAuth3LeggedScheme(new GoogleProvider(), GOOGLE_KEY, GOOGLE_SECRET);
3 }
```

Quellcode 6.25: Implementierung der `createOAuth3LeggedScheme()`-Methode im Google-Plug-in

Wie durch den gezeigten Ausschnitt aus dem Code des Plug-ins deutlich wird, konnte der Google-Service sehr schnell realisiert werden. Dies ist vorwiegend auf die Nutzung der OpenSocial-API und die bereits vorhandene Implementierung in der Social-Plug-in-API zurückzuführen.

6.3 Aktualisierung der integrierten Werkzeuge

Bei der Integration der verschiedenen Werkzeuge wurde darauf geachtet, dass deren Aktualisierung zur Bereitstellung neuer Funktionen oder zur Behebung von Sicherheitslücken möglichst effektiv unterstützt werden kann. Sowohl die eingesetzten Strategien als auch einige allgemeine Richtlinien zur Aktualisierung der Werkzeuge jeder Service-Kategorie werden in diesem Abschnitt skizziert.

6.3.1 Base-Services

Base-Services sind server- oder webbasierte Anwendungen, die für jedes Projekt Daten auf den Servern einer SSELab-Instanz speichern, so dass der Aktualisierungsaufwand für Werkzeuge dieser Kategorie vergleichsweise hoch ist. Bei der Integration von Werkzeugen dieser Kategorie wurden daher einige Prinzipien angewandt, die deren Aktualisierung vereinfachen.

Die Installation und Konfiguration eines Werkzeugs der Base-Service-Kategorie wurde gemäß der Methodik zur Integration (vgl. Abschnitt 5.2.2) jeweils automatisiert. Im Zuge der Automatisierung wurde einerseits darauf geachtet, dass die Skripte zur Installation und Konfiguration unabhängig voneinander ausgeführt werden können. Dadurch kann die Aktualisierung eines Werkzeugs zunächst durch die Installation einer neuen Version erfolgen und im Anschluss die Konfiguration durchgeführt werden. Andererseits wurden möglichst stabile Schnittstellen der Werkzeuge zur Konfiguration genutzt, so dass die entsprechenden Skripte relativ selten angepasst werden müssen. Nach der Installation einer neuen Version muss jedoch immer geprüft werden, ob sich die zur Konfiguration verwendeten Schnittstellen geändert haben. Ist dies nicht der Fall, können die Skripte unverändert genutzt werden.

Insgesamt kann dadurch die Aktualisierung eines Großteils der als Base-Service integrierten Anwendungen effizient erfolgen. Die Aktualisierung der Services Subversion, Git, Storage, Web-Space, Mailing-Listen, Feedback-System, Jenkins und Nexus kann beispielsweise ohne großen Aufwand durchgeführt werden. Bei einigen integrierten Anwendungen, beispielsweise bei Trac, MediaWiki und Wordpress, wurden zusätzliche Erweiterungen in die Anwendungen installiert, die beispielsweise eine Integration mit den im SSELab-Framework genutzten Authentifizierungs- und Autorisierungsmechanismen möglich machen. Vor einer Aktualisierung dieser Anwendungen muss dementsprechend geprüft werden, ob diese Erweiterungen noch verfügbar und kompatibel sind oder eventuell nicht mehr benötigt werden, da die Funktionalität durch die Anwendung selbst zur Verfügung gestellt wird. Bei den Services Trac, MediaWiki und Wordpress mussten darüber hinaus einige Skripte zur Installation und Konfiguration auf Grundlage interner APIs realisiert werden, um eine vollständige Automatisierung erreichen zu können. Falls sich diese APIs bei einer neueren Version der Anwendungen geändert haben, müssen diese Skripte entsprechend angepasst werden.

Da die Anwendungen der Base-Service-Kategorie üblicherweise persistente Daten verwalten, muss bei deren Aktualisierung auch auf geänderte Datenformate geachtet und gegebenenfalls eine Schema- und Datenmigration durchgeführt werden. Diese Migration wird üblicherweise von den Entwicklern der Anwendungen berücksichtigt, so dass entsprechende Anleitungen zur Durchführung der Migration in den meisten Fällen in der Dokumentation der neuen Version verfügbar sind.

6.3.2 Ostp-Services

Ostp-Services sind dateiverarbeitende Werkzeuge, die nach der Integration in das SSELab-Framework in einer Serverumgebung ausgeführt werden. Da Ostp-Services neben den vom Framework bereitgestellten Caching-Mechanismen üblicherweise keine persistenten Daten verwalten, entspricht die Aktualisierung eines Ostp-Services der ursprünglichen Installation. Daher können die in Abschnitt 5.2.3 beschriebenen Schritte der Methodik zur Installation für eine Aktualisierung unverändert angewandt werden.

Ostp-Services können darüber hinaus innerhalb der Backends auch gleichzeitig in mehreren Versionen installiert werden. Beispielsweise sind die als Ostp-Services integrierten Werkzeuge MontiCore und MontiArc jeweils in mehreren Versionen in der im Rahmen dieser Arbeit gebildeten SSELab-Instanz gleichzeitig verfügbar. Die Aktualisierung eines Ostp-Services ist daher im Allgemeinen nur zur Behebung von Fehlern notwendig.

6.3.3 Social-Services

Social-Services werden im SSELab-Framework zur Verarbeitung von Informationen aus sozialen Netzwerken bzw. anderen Projektportalen im Kontext von Benutzerprofilen eingesetzt. Eine Aktualisierung der zugehörigen Service-Plug-ins muss dann erfolgen, wenn die Anbieter des entsprechenden Portals die genutzte API ändern oder aber ein Fehler innerhalb des Plug-ins existiert. Änderungen der API werden in vielen Fällen frühzeitig von den Betreibern angekündigt, so dass eine Aktualisierung rechtzeitig erfolgen kann. Die Aktualisierung entspricht auch bei dieser Service-Kategorie der Installation, so dass die in Abschnitt 5.2.4 beschriebenen Schritte durchgeführt werden können.

6.4 Anforderungen an Werkzeuge zur vereinfachten Integration

Im Rahmen dieser Arbeit konnten durch die Integration vielfältiger Werkzeuge in das Framework einige Erfahrungen gesammelt werden, die teilweise schon in den vorigen Abschnitten aufgegriffen worden sind. Insgesamt lässt sich auf Grundlage der Erfahrungen feststellen, dass viele Werkzeuge entwickelt werden, ohne dass eine Integration in Portale, Plattformen oder Frameworks ausreichend berücksichtigt wird.

Das SSELab-Framework ist auf eine leichtgewichtige Integration ausgelegt, bei der existierende Werkzeuge minimal angepasst werden müssen und insbesondere keine Konformität der Werkzeuge zu dem Framework vorausgesetzt wird. Dennoch konnten auf Grundlage der gesammelten Erfahrungen einige Anforderungen an Werkzeuge der drei Service-Kategorien iden-

tifiziert werden, deren Umsetzung eine Integration in Frameworks wie das SSELab deutlich vereinfachen würde. In den folgenden Abschnitten werden diese Anforderungen jeweils nach einer Zusammenfassung der gesammelten Erfahrungen aufgelistet.

6.4.1 Anforderungen an Base-Services

Die Integration von Werkzeugen der Base-Service-Kategorie in das Framework erfolgte anhand der in Abschnitt 5.2.2 vorgestellten Methodik. Abbildung 6.26 zeigt noch einmal die dabei durchzuführenden Schritte.

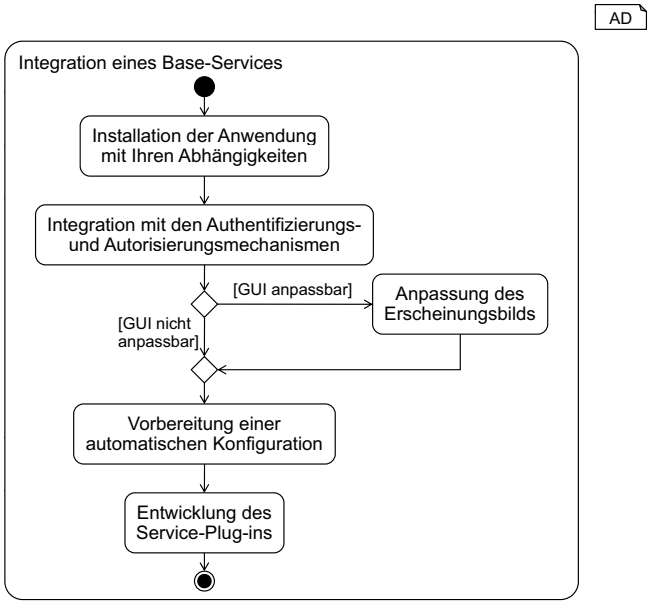


Abbildung 6.26: Schritte zur Integration von Anwendungen als Base-Services

Die Installation der Anwendungen und der Abhängigkeiten sowie der Laufzeitumgebungen war bei den integrierten Werkzeugen ohne Komplikationen möglich, da die meisten Anwendungen weit-verbreitete Web-, Application- und Datenbankserver verwenden. Dies ergibt sich aber auch dadurch, dass diese Eigenschaft ein Auswahlkriterium war, falls mehrere alternative Werkzeuge zur Verfügung standen.

Die Integration mit den Authentifizierungs- und Autorisierungsmechanismen des SSELab-Frameworks war im Gegensatz dazu nicht immer einfach möglich. HTTP Basic Authentication wird zwar von den meisten der Anwendungen direkt oder durch installierbare Erweiterungen unterstützt, jedoch sind die verschiedenen Implementierungen der Zugriffsrechte ein relativ großes Hindernis für eine effiziente Integration. Es ist verständlich, dass einige Anwendungen

ein komplexes Rechtemanagement realisieren müssen, um alle ihre Anforderungen abdecken zu können. Aber ohne eine Möglichkeit zur automatischen Konfiguration der Benutzer sowie ihrer Rechte und Rollen zu bieten, wird eine Integration dieser Anwendungen erheblich erschwert.

Die Anpassung des Erscheinungsbilds war in den meisten Fällen mit vertretbarem Aufwand möglich. Die dafür eingesetzten Mechanismen sind naturgemäß abhängig von der jeweiligen Anwendung und den zur Realisierung genutzten Technologien. Jenkins und Nexus sind hier als Ausnahmen zu nennen, da beide nur eine unzureichende Anpassung des Erscheinungsbilds erlauben.

Die Unterstützung einer automatischen Konfiguration ist bei den integrierten Anwendungen unterschiedlich ausgeprägt. Wie zuvor diskutiert, basiert die Konfiguration von Subversion und Trac beispielsweise auf Administrationswerkzeugen und textbasierten Dateien, so dass sie optimal über Skripte automatisiert verwaltet werden können. Im Gegensatz dazu bieten beispielsweise die integrierten Versionen von MediaWiki und Wordpress nur formularbasierte Webseiten für eine manuelle Installation. Die Automatisierung der Konfiguration dieser Anwendungen war daher unnötig komplex.

Insgesamt ergeben sich aus den Erfahrungen bei der Integration der Anwendungen dieser Kategorie die folgenden Anforderungen.

BA1 *Unterstützung verbreiteter Server*

Eine server- oder webbasierte Anwendung sollte weit verbreitete Web-, Application- und Datenbank-Server unterstützen. Im Idealfall werden dabei auch mehrere Alternativen zur Auswahl angeboten.

BA2 *Unterstützung mehrerer Authentifizierungsarten*

Eine Anwendung sollte gängige und auf Standards basierende Authentifizierungsmechanismen unterstützen. Wünschenswert ist insbesondere, dass dabei mehrere Arten angeboten werden, so dass bei der Integration eine passende Art ausgewählt werden kann.

BA3 *Konfigurierbare und feingranulare Zugriffsrechte*

Eine Anwendung sollte eine automatische Konfiguration der Zugriffsrechte von Benutzern über eine geeignete Schnittstelle ermöglichen. Darüber hinaus sollten diese Rechte in ausreichend feiner Granularität vergeben werden können.

BA4 *Automatische Konfiguration*

Eine einfach automatisierbare Installation und Konfiguration aller wesentlichen Eigenschaften und Funktionen einer Anwendung ist essentiell für deren effektive Integration und daher eine wichtige Anforderung [Zel07]. Die Berücksichtigung dieser Anforderung hat in vielen Fällen auch positive Auswirkungen auf die Architektur einer Anwendung, da zusätzliche Schnittstellen geschaffen werden, die bestimmte Funktionalität kapseln.

BA5 *Trennung der Funktionalität von der graphischen Oberfläche*

Eine zu der vorigen verwandte Anforderung ist die Trennung der Funktionalität der

Anwendung von der Benutzeroberfläche [Zel07]. Dadurch ist es möglich, die Funktionalität beispielsweise als Service zu nutzen, ohne auf die graphische Oberfläche angewiesen zu sein.

BA6 *Partitionierung der Daten und Konfiguration*

Falls eine Anwendung mehrere Projekte unterstützt, sollten sowohl die Daten als auch die zugehörige Konfigurationen der Projekte strikt getrennt werden, so dass die Mandantentrennung für Projekte unterstützt werden kann. Beispielsweise unterstützt Subversion mehrere Repositories, die im Dateisystem des Servers unabhängig angelegt werden können. Im Kontext des SSELabs wird ein solches Repository einem Projekt zugeordnet. Die Konfiguration der Zugriffsrechte für alle Repositories erfolgt jedoch über eine zentrale Datei. Besser wäre hier auch eine Trennung der Konfiguration der Repositories.

BA7 *Anpassung des Erscheinungsbilds*

Bei wiederverwendbaren Web-Anwendungen ist eine Anpassung des Erscheinungsbilds wichtig für eine Integration mit anderen Anwendungen. Die Anpassung sollte dabei möglichst einfach sein, beispielsweise durch die Verwendung von Template-Engines.

6.4.2 Anforderungen an Ostp-Services

Die Methodik zur Integration von Werkzeugen als Ostp-Service wurde in Abschnitt 5.2.3 erläutert. Die Schritte sind noch einmal in Abbildung 6.27 zusammengefasst.

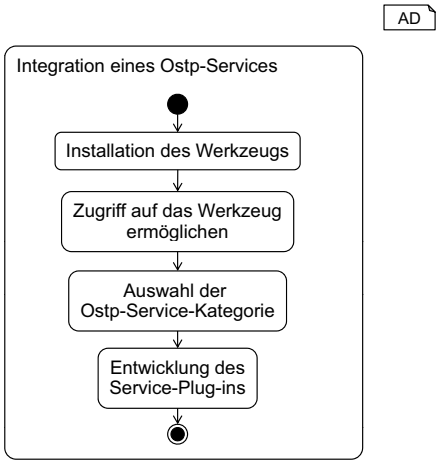


Abbildung 6.27: Schritte zur Integration von Werkzeugen als Ostp-Services

Viele der im Rahmen dieser Arbeit als Ostp-Services integrierten Werkzeuge waren Eigenentwicklungen des Lehrstuhls und basieren daher auf relativ einheitlichen Technologien. Die Installation dieser Werkzeuge war demzufolge ohne Weiteres möglich. Darüber hinaus hat auch die Installation externer Werkzeuge keine nennenswerten Probleme bereitet.

Den Zugriff auf die Werkzeuge von den Backend-Instanzen zu ermöglichen, war aus den aufgeführten Gründen ebenfalls leicht möglich. Bei einigen Werkzeugen, wie beispielsweise ppt2eps, mussten für eine Integration zusätzliche Infrastruktur, beispielsweise in Form einer Web-Service-Schnittstelle geschaffen werden.

Einige Schwierigkeiten ergaben sich bei der Integration dadurch, dass die Werkzeuge als Desktop-Anwendungen konzipiert wurden, so dass eine Integration in eine serverbasierte Umgebung üblicherweise nicht berücksichtigt worden war. Insgesamt lassen sich auf Grundlage der gesammelten Erfahrungen die folgenden Anforderungen aufstellen.

OA1 *Erweiterbarkeit unterstützen*

Anwendungen sollten erweiterbar entwickelt werden [Zel07], damit fehlende Funktionalität bei Bedarf ergänzt werden kann. Dies gilt insbesondere auch für Funktionalität, die bei der Integration der Anwendung relevant ist.

OA2 *Trennung der Anwendungslogik von der Benutzeroberfläche*

Bei dieser Art von Werkzeugen ist es ebenfalls wichtig, dass die Funktionalität auch unabhängig von einer graphischen Oberfläche über eine Schnittstelle genutzt werden kann, wie auch in [Zel07] gefordert wird. Dies ermöglicht erst den effektiven Einsatz eines Werkzeugs in einer serverbasierten Umgebung.

OA3 *Minimale Anzahl von Laufzeitabhängigkeiten*

Bei der Entwicklung von Anwendungen sollte immer auf die Unterscheidung der Abhängigkeiten geachtet werden (beispielsweise Laufzeit-, Compile-Zeit-, Testabhängigkeiten). Darüber hinaus erleichtert die Auslieferung von Anwendungen mit einer minimalen Menge von Laufzeitabhängigkeiten deren Integration.

OA4 *Betrieb in Multi-Prozess- und Multi-Thread-Umgebungen*

In einer Serverumgebung wird ein Werkzeug üblicherweise in einer Mutli-Thread-Umgebung ausgeführt oder auch mehrere Instanzen des Werkzeugs gleichzeitig betrieben. Bei der Entwicklung sollte dies unter anderem beim Zugriff auf programminterne und -externe Ressourcen berücksichtigt werden.

6.4.3 Anforderungen an Social-Services

Die Integration von Funktionen sozialer Netzwerke als Social-Services basiert auf der in Abschnitt 5.2.4 beschriebenen Methodik, die noch einmal durch das Aktivitätsdiagramm in Abbildung 6.28 illustriert wird.

Das Social-Backend wurde erst gegen Ende dieser Arbeit realisiert und in das Framework integriert, so dass noch nicht viele Plug-ins entwickelt und dementsprechend vergleichsweise

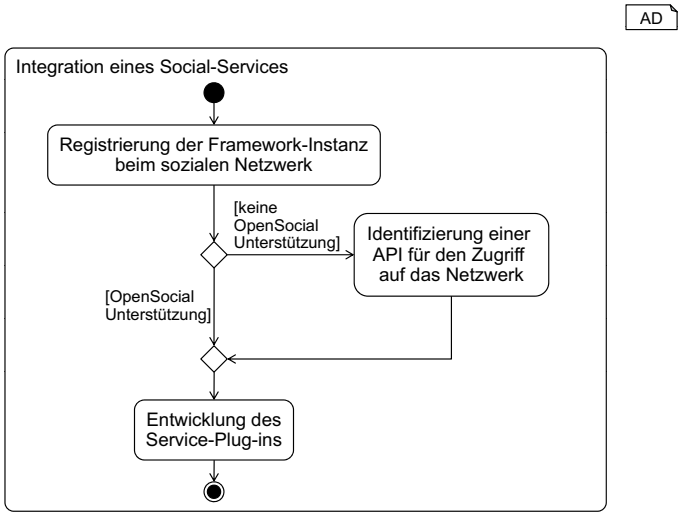


Abbildung 6.28: Schritte zur Integration von Funktionen sozialer Netzwerke als Social-Services

wenig Erfahrungen gesammelt werden konnten. Die wichtigsten Anforderungen auf Grundlage der bisherigen Erfahrungen sind in der folgenden Auflistung zusammengefasst.

SA1 *Unterstützung gängiger Authentifizierungsarten*

Portale und soziale Netzwerke sollten standardisiert und verbreitete Protokolle zur Authentifizierung unterstützen, damit eine Anmeldung über externe Anwendungen möglich ist.

SA2 *Dokumentierte und stabile API*

Darüber hinaus ist die Bereitstellung dokumentierter und auch stabiler Schnittstellen wichtig, damit Informationen oder Services eines Portals bzw. Netzwerks effizient genutzt werden können. Im Idealfall werden auch direkt Client-Komponenten für die Kommunikation über die Schnittstellen angeboten.

6.5 Zusammenfassung

In diesem Kapitel wurde aufbauend auf der Beschreibung der Erweiterungsmechanismen des Frameworks in Kapitel 5 die Integration konkreter Werkzeuge behandelt. Zunächst wurde dazu eine Übersicht über alle Werkzeuge gegeben, die im Rahmen dieser Arbeit in das Framework integriert worden sind. Im Anschluss daran wurden einige Details der Methodiken zur Integration anhand ausgewählter Werkzeuge erläutert. Neben den vorbereitenden Schritten der Integration wurde dabei auch detailliert auf einige Aspekte der Service-Plug-ins eingegangen. Im Anschluss daran

wurden die Aktualisierung der integrierten Werkzeuge beschrieben sowie einige Anforderungen aufgestellt, deren Erfüllung zu einer vereinfachten Integration von Werkzeugen in Framework, Portale oder Plattformen führt. Die Anforderungen basieren auf den gesammelten Erfahrungen bei der Integration der Werkzeuge und der Erkenntnis, dass die Integration von Werkzeugen in Umgebungen wie das SSELab noch immer nicht ausreichend berücksichtigt wird.

Kapitel 7

Methodik zur Bildung einer Instanz des Frameworks

Das Deployment großer Softwaresysteme ist ein komplexer und herausfordernder Aufgabenbereich in Software Engineering Projekten [Dea07]. Gemäß der Spezifikation *Deployment and Configuration of Component-based Distributed Applications* [OMG06] der OMG ist Deployment ein Prozess, der sowohl nach der Entwicklung und der Auslieferung als auch nach der Akquise eines Softwareprodukts durch den Besitzer selbst durchgeführt wird. In anderen Definitionen werden das Release der Software sowie die Aktivitäten, die nach der Installation ausgeführt werden, beispielsweise die Aktualisierung, Überwachung und Deinstallation, auch als Deployment-Aktivitäten bezeichnet [Dea07, CFH+98]. Im Rahmen dieses Kapitels werden diese umfassenderen Definitionen des Deployment-Prozesses zugrunde gelegt.

Beim Deployment von Software as a Service ist zusätzlich hervorzuheben, dass alle Aktivitäten ausschließlich auf der Serverinfrastruktur durchgeführt werden, auf der ein Produkt laufen soll. Für das SSELab-Framework bedeutet dies insgesamt, dass die Bildung einer Instanz durch die Konfiguration und die Instanziierung der Framework-Komponenten und deren Installation in der zugehörigen Laufzeitumgebung auf einer Serverinfrastruktur erfolgt. Nachdem eine Instanz des Frameworks erzeugt worden ist, können jedoch zunächst nur die in Abschnitt 4.4 beschriebenen Verwaltungsfunktionen des Frontends genutzt werden. Die Durchführung konkreter Projekte wird erst durch ein Deployment von Service-Plug-ins der verschiedenen Kategorien und die Bereitstellung der zugehörigen Clients ermöglicht. Daraus folgt auch, dass die installierten Services den Einsatzzweck einer Instanz des Frameworks prägen und die Projektarten bestimmen, die durch diese effektiv unterstützt werden können.

In diesem Kapitel wird aufbauend auf den vorigen Kapiteln sowohl das Deployment aller Komponenten des SSELab-Frameworks als auch von Services und Clients behandelt. Als Grundlage für das Verständnis des Deployment-Prozesses wird in den folgenden Abschnitten zunächst die Verteilung der Komponenten und darauf aufbauend verschiedene Betriebsarten einer SSELab-Instanz sowie die a posteriori Integration von Werkzeugen diskutiert. Im Anschluss daran werden die Laufzeitumgebungen der Backend- und Frontend-Komponenten eingeführt und abschließend der Deployment-Prozess beschrieben.

7.1 Verteilung der Komponenten des Frameworks

Zur Umsetzung der beiden Anforderungen **FA47** (Entwicklung als webbasiertes Framework) und **NFA11** (Flexibles Deployment) wurde das SSELab-Framework so konzipiert, dass es ver-

schiedene Verteilungsvarianten unterstützt. Insbesondere die Entkopplung des Frontends von den verschiedenen Backends und den Clients durch den Einsatz entsprechender Schnittstellen ermöglicht neben der Erweiterbarkeit des Frameworks auch eine flexible Verteilung der einzelnen Komponenten. In Abbildung 7.1 ist zur Veranschaulichung eine exemplarische Verteilungsvariante gezeigt, bei der eine Frontend-Instanz zur Verwaltung von drei Backend-Instanzen eingesetzt wird. Zwei davon sind Base-Backend-Instanzen und eine ist eine Ostp-Backend-Instanz, die jeweils auf einem eigenständigen Server ausgeführt werden. Diese Verteilung der Komponenten entspricht weitestgehend der im Rahmen dieser Arbeit erzeugten Framework-Instanz in der Version 1.0, die unter `http://sselab.de` zur Verfügung gestellt worden ist. Weitere Details zu dieser Instanz werden im folgenden Kapitel 8 beschrieben.

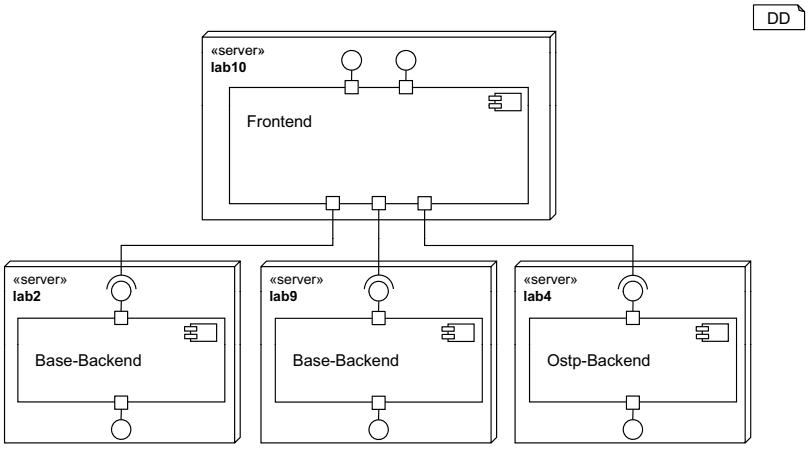


Abbildung 7.1: Verteilungsdiagramm mit einer Frontend- und drei Backend-Instanzen

Neben der gezeigten Verteilungsvariante sind auch noch weitere Varianten möglich, bei denen die Instanzen des Frontends und der Backends auf einer geringeren Anzahl von Servern ausgeführt werden. Beispielsweise werden für die Entwicklung des SSELab-Frameworks virtuelle Maschinen eingesetzt, auf denen jeweils eine Frontend-Instanz, zwei Base- und Ostp-Backend-Instanzen sowie eine Social-Backend-Instanz gleichzeitig betrieben werden. Um eine möglichst einfache Wartbarkeit und eine optimale Skalierbarkeit zu erreichen, sollten jedoch die Frontend-Instanz und die Backend-Instanzen jeweils auf einem dedizierten Server ausgeführt werden. Dadurch wird auch sichergestellt, dass die Administrationsoberfläche von den Daten der Services getrennt betrieben werden kann, was eine Voraussetzung für die nachfolgend beschriebenen Betriebsarten ist. Unabhängig von der gewählten Verteilungsvariante können gemäß Anforderung **FA47** sowohl herkömmliche Server (reale oder virtuelle Maschinen) als auch Cloud-Infrastrukturen [AFG+10] für das Hosting einer Framework-Instanz eingesetzt werden.

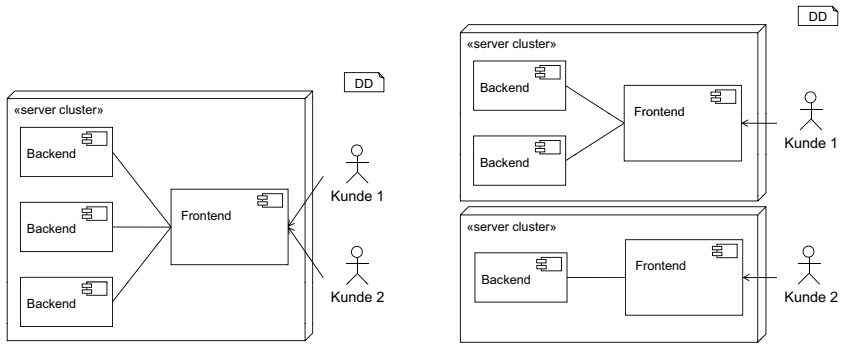
Durch die beschriebenen Verteilungsvarianten können einerseits verschiedene Betriebsarten einer Framework-Instanz und andererseits die a posteriori Integration von Werkzeugen effektiv unterstützt werden. Beide Aspekte werden in den folgenden Abschnitten behandelt.

7.1.1 Betriebsarten einer Framework-Instanz

Bei der Nutzung von Software as a Service ergeben sich immer Fragestellungen in Bezug auf die Datensicherheit und den Datenschutz [GTHM01]. Bei Software Engineering Projekten sind insbesondere die Entwicklungsartefakte sensibel, da sie zum geistigen Eigentum des Softwareproduzenten gehören. Die Sicherheit und der Schutz der Artefakte muss daher sowohl vertraglich durch Service Level Agreements (SLAs) als auch technisch gewährleistet werden. Darüber hinaus ist es für den Erfolg eines Anbieters von SaaS-Lösungen essentiell, dass sowohl der Anbieter selbst als auch dessen Partner als vertrauenswürdig angesehen werden und eine gute Reputation besitzen [GTHM01]. Auf SLAs und den Aufbau eines guten Ansehens wird in diesem Kapitel nicht weiter eingegangen, sondern die Mechanismen des SSELab-Frameworks zur Mandantentrennung vorgestellt, die durch die Verteilungsvarianten der Framework-Komponenten entstehen. Die Möglichkeiten, die sich bei der Bildung einer Instanz des Frameworks ergeben, lassen sich wie folgt zusammenfassen.

- Gemeinsames oder getrenntes Hosting der Daten und der Administrationsoberfläche.
- Betrieb einer SSELab-Instanz für mehrere oder für einzelne Kunden.

Durch die Kombination dieser Möglichkeiten ergeben sich die vier grundlegenden Betriebsarten einer SSELab-Instanz. Die zwei Varianten bei denen die Daten und die Administrationsoberfläche gemeinsam gehostet werden, sind in Abbildung 7.2 schematisch dargestellt.



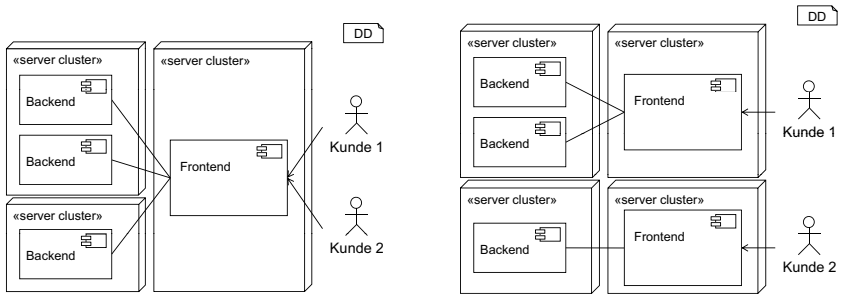
(a) Betrieb einer Instanz für mehrere Kunden (b) Betrieb einer Instanz für einzelne Kunden

Abbildung 7.2: Gemeinsames Hosting der Daten und der Administrationsoberfläche einer SSELab-Instanz

Abbildung 7.2a zeigt den Betrieb einer Instanz für mehrere Kunden. In diesem Fall wird eine Frontend-Instanz zur Verwaltung der Projekte mehrerer Kunden verwendet. Die in Abschnitt 4.4 beschriebenen Verwaltungsmechanismen stellen dabei sicher, dass ein Kunde ausschließlich die eigenen Projekte nutzen und administrieren kann und nicht die der anderen Kunden – also die Mandantenfähigkeit gewährleistet ist. Bei dieser Variante werden alle Komponenten durch

den Anbieter selbst gehostet, so dass diese dem klassischen SaaS-Modell entspricht. Falls eine striktere Trennung der Projekte einzelner Kunden notwendig sein sollte, kann jeweils eine dedizierte Instanz des Frameworks gebildet werden, durch die die Projekte der jeweiligen Kunden exklusiv gehostet werden. Abbildung 7.2b illustriert diese Betriebsart. Bei dieser Variante ist es möglich, dass der Anbieter das Hosting selbst übernimmt oder aber die Komponenten innerhalb der Serverinfrastruktur des Kunden installiert werden. Letzteres ist insbesondere eine geeignete Lösung für SaaS-Anbieter, die noch nicht etabliert sind und daher keine ausreichende Reputation aufbauen konnten [GTHM01].

Abbildung 7.3 zeigt die beiden Betriebsarten bei denen die Daten und die Administrationsoberfläche auf getrennten Serverinfrastrukturen gehostet werden. Abbildung 7.3a illustriert die Variante, bei der eine Frontend-Instanz zur Verwaltung aller Backend-Instanzen der Kunden verwendet wird. Bei dieser Variante wird eine Zuordnung der jeweiligen Kunden zu einer festen Menge von Backend-Instanzen genutzt, so dass bei der Erzeugung neuer Projekte für einen Kunden eine der zugehörigen Backend-Instanzen ausgewählt wird. Die Backend-Instanzen und damit die Daten können bei dieser Variante bei Bedarf wiederum bei den Kunden gehostet werden. Das Frontend kann in diesem Fall ausschließlich durch den Anbieter selbst betrieben werden. Dadurch wird die Wartung und Aktualisierung sowie die kundenübergreifenden Konfiguration vereinfacht. Abbildung 7.3b zeigt schließlich die entsprechende Variante bei der eine eigenständige Instanz pro Kunde eingesetzt wird, um eine striktere Trennung zwischen den Kunden zu erreichen. Dabei betreibt der Anbieter eine unabhängige Frontend-Instanz pro Kunde und ermöglicht die Installation der Backend-Instanzen bei den Kunden.



(a) Betrieb einer Instanz für mehrere Kunden (b) Betrieb einer Instanz für einen Kunden

Abbildung 7.3: Getrenntes Hosting der Daten und der Administrationsoberfläche einer SSELab-Instanz

Insgesamt ermöglichen die Verteilungsvarianten des Frameworks eine Auswahl und auch eine Kombination von Betriebsarten, um auf die Bedürfnisse einzelner Kunden eingehen zu können. Bei den Varianten, bei denen Framework-Komponenten auf der Infrastruktur eines Kunden gehostet werden, müssen jedoch der erhöhte Aufwand und die benötigten Rechte zur Durchführung der Wartung durch den Anbieter berücksichtigt und vertraglich geregelt werden [GTHM01].

7.1.2 A posteriori Integration von Werkzeugen

Bei der Realisierung des SSELab-Frameworks wurde frühzeitig eine Instanz gebildet und produktiv eingesetzt. Dies hat sowohl die Erprobung der Konzepte als auch schnelles Feedback durch die Benutzer der Instanz ermöglicht. Die Instanz wurde dabei von Grund auf neu erzeugt und inkrementell um neue Funktionen und Services ergänzt. Auch die Projekte wurden erst nach der Bildung der Instanz angelegt. Insgesamt ist daher das Deployment einer SSELab-Instanz auf einer ungenutzten Serverinfrastruktur das primär unterstützte Szenario. Darüber hinaus ist es jedoch auch möglich, eine Framework-Instanz zu erzeugen, durch die nachträglich bereits installierte, konfigurierte und genutzte Anwendungen inklusive ihren gespeicherten Daten verwaltet werden können. Dieser Einsatzzweck wird in diesem Abschnitt anhand der drei Service-Kategorien skizziert.

Ein Beispiel für die a posteriori Integration von *Base-Services* ist die Verwaltung existierender Subversion-Repositories mit Hilfe eines Base-Backends und des Subversion-Plug-ins. In diesem Szenario existieren bereits ein oder mehrere Subversion-Repositories, deren Konfiguration beispielsweise nicht mehr manuell, sondern mit Hilfe einer SSELab-Instanz verwaltet werden sollen. Durch die Vielzahl der Konfigurationsmöglichkeiten eines Werkzeugs wie Subversion ist jedoch davon auszugehen, dass die Konfiguration der existierenden Subversion-Repositories von der durch das Subversion-Plug-in erwarteten Konfiguration abweicht. Zur Angleichung der erwarteten und der tatsächlichen Konfiguration existieren die folgenden drei Strategien.

- Anpassung des Service-Plug-ins an die tatsächliche Konfiguration.
- Anpassung der tatsächlichen Konfiguration an die vom Plug-in erwartete (Konfigurationsmigration).
- Erstellung neuer Repositories mit der Konfiguration des Service-Plug-ins und Import der Daten (Datenmigration).

Bei der ersten Strategie wird das Service-Plug-in individuell an die tatsächliche Konfiguration angepasst und kann dadurch ausschließlich für diese genutzt werden. In diesem Fall kann ein unterbrechungsfreier Betrieb der Anwendung sichergestellt werden, was bei den übrigen Strategien gegebenenfalls nicht möglich ist. Bei diesen Strategien wird jedoch eine Diversifizierung der Konfigurationen vermieden, was langfristig einen geringeren Wartungsaufwand zur Folge hat. Insgesamt lässt sich feststellen, dass von Fall zu Fall entscheiden werden muss, welche Strategie optimal bezüglich des Kosten-Nutzen-Verhältnisses ist. Die wichtigsten Einflussfaktoren bei der Entscheidung für eine der Strategien sind unter anderem die Anzahl der Benutzer und ob bzw. welcher Aufwand für sie entsteht und ob eine Unterbrechung des Betriebs der Server und damit der Anwendungen überhaupt möglich ist.

Die a posteriori Integration von *Ostp-Services* ist im Gegensatz dazu deutlich einfacher, da diese Werkzeuge keine persistenten Daten speichern und damit zustandslos verwendet werden. Daher muss zur Integration eines bereits als Service verwendeten Werkzeugs nur ein entsprechendes Service-Plug-in bereitgestellt werden. In diesem Fall erfolgt die Realisierung eines Service-Plug-ins für ein bereits installiertes Werkzeug in Übereinstimmung mit der in Kapitel 5 vorgestellten Methodik zur Integration von Ostp-Services. Der Fokus liegt in diesem Fall auf

der Integration einer Legacy-Anwendung durch die Erzeugung eines Wrappers. Der Zugriff der Nutzer auf das Werkzeug muss in diesem Fall anschließend entweder über das Frontend der SSELab-Instanz oder über geeignete Clients erfolgen. Die Clients können in einigen Fällen so entwickelt werden, dass der Übergang von einer lokalen zu einer serverbasierten Installation des Werkzeugs für die Nutzer transparent erfolgen kann. Dies wurde im Rahmen dieser Arbeit anhand der Integration von MontiCore in das SSELab erfolgreich demonstriert (vgl. Abschnitt 6.2.5).

Die Integration von *Social-Services* entspricht in den meisten Fällen der in diesem Abschnitt beschriebenen a posteriori Integration in Übereinstimmung mit der in Abschnitt 5.2.4 behandelten Methodik, da die zugehörigen Netzwerke oder Portale bereits mit Produktivdaten betrieben werden. Die Herausforderung bei Social-Services ist daher primär die Nutzung der APIs der Netzwerke zu Testzwecken bei der Entwicklung der zugehörigen Service-Plug-ins.

7.2 Laufzeitumgebungen der Komponenten des Frameworks

Die Kenntnis der Laufzeitumgebungen der Komponenten des Frameworks ist eine Voraussetzung für die Durchführung des Deployments. In diesem Abschnitt werden daher die wichtigsten Aspekte der Laufzeitumgebungen von Backend- und Frontend-Instanzen beschrieben, bevor im folgenden Abschnitt die Methodik des Deployments diskutiert wird. Hier wird die Laufzeitumgebung erläutert, die aktuell für den Betrieb der Instanz des Frameworks unter `http://sselab.de` eingesetzt wird. Viele Bestandteile der Infrastruktur könnten jedoch auch ersetzt werden, so dass im Folgenden auch jeweils alternative Technologien genannt werden.

7.2.1 Laufzeitumgebung von Backend-Instanzen

Aus der Anforderung **FA49** (Zentralisierung der integrierten Werkzeuge als Services (Hosted Services)) ergibt sich unter anderem, dass die Backend-Instanzen als Server laufen und dauerhaft verfügbar sein müssen. Die Laufzeitumgebung, die aktuell zur Umsetzung dieser Anforderung eingesetzt wird, ist in Abbildung 7.4 in einem Verteilungsdiagramm gezeigt. Die Eigenschaften der Laufzeitumgebung aller Backend-Arten werden hier wie zuvor anhand eines Base-Backends erläutert, gelten jedoch für die anderen Arten entsprechend.

Aus den in Abschnitt 7.1 erwähnten Gründen wird jede Backend-Instanz auf einem dedizierten Server ausgeführt. Aktuell werden virtuelle Maschinen auf einem VMWare ESXi-Cluster mit Ubuntu Server als Betriebssystem eingesetzt. Eine Verwendung von Cloud-Infrastrukturen, anderer Linux-Distributionen oder Unix-Derivaten ist jedoch auch möglich, falls die Softwarekomponenten, die von einer Backend-Art und deren Services benötigt werden, verfügbar sind.

Ein zentrales Konzept des SSELab-Frameworks ist der durchgängige Einsatz von Plug-in-Systemen in allen Backend-Arten. Die Plug-in-Systeme der Realisierung basieren auf der OSGi-Spezifikation [OSG09a], so dass sowohl die Komponenten einer Backend-Instanz und deren Abhängigkeiten als auch die in Abbildung 7.4 als Beispiel gezeigten Plug-ins für Subversion und Trac in einem OSGi-Container laufen. Aktuell wird Apache Felix [Géd10] als OSGi-Container verwendet. Andere Implementierungen der OSGi-Spezifikation könnten jedoch ebenfalls genutzt werden, wie beispielsweise Eclipse Equinox [WHKL08], das im Verlauf dieser Arbeit vorüber-

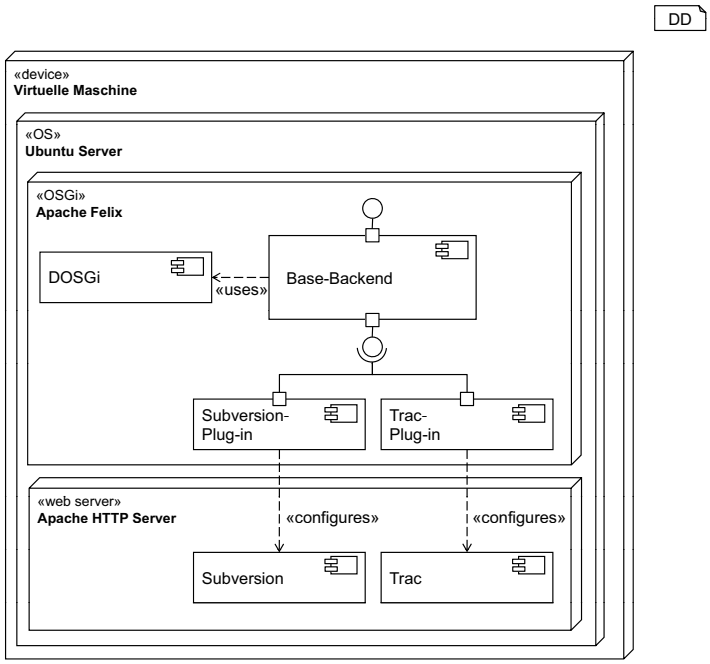


Abbildung 7.4: Laufzeitumgebung eines Base-Backends mit installierten Service-Plug-ins und den zugehörigen Anwendungen

gehend auch als Container eingesetzt worden ist. Für die Ausführung eines OSGi-Containers wird lediglich ein Java Runtime Environment (JRE) benötigt.

Ein weiterer wesentlicher Bestandteil der Laufzeitumgebung aller Backend-Instanzen sind die Komponenten von Apache CXF Distributed OSGi [DOSG12]. Das DOSGi-Projekt stellt eine Referenzimplementierung für die Komponente Distribution Provider der OSGi-Spezifikation [OSG09b] zur Verfügung. Die Implementierung nutzt Web-Services auf Grundlage von JAX-WS und Jetty als eingebetteten Servlet-Container und HTTP-Server. Der Einsatz der DOSGi-Komponenten ermöglicht die Verwendung des OSGi-Containers als Server und die Veröffentlichung einer Web-Service-Schnittstelle für jede Backend-Instanz. In Abbildung 7.4 sind der Übersichtlichkeit halber nicht alle Komponenten explizit angegeben, sondern werden durch eine einzelne Komponente mit dem Namen *DOSGi* repräsentiert. Alternativ zu der Einbettung eines Servlet-Containers in einen OSGi-Container könnte auch die Einbettung anders herum erfolgen, also ein OSGi-Container in einen Servlet-Container eingebettet werden, wie beispielsweise in [KD08] beschrieben wird.

Neben der bisher vorgestellten Laufzeitumgebung für die Backend-Instanzen benötigen die Services ebenfalls eine bestimmte Infrastruktur zur Ausführung. Welche Infrastruktur das im

Einzelnen ist, variiert entsprechend der Anforderungen eines konkreten Services. In Abbildung 7.4 ist exemplarisch Apache HTTPD [LL02] als Web-Server für die beiden Services Subversion und Trac gezeigt. Auf die benötigte Infrastruktur weiterer Services wird hier nicht im Detail eingegangen. Bei der Integration aller Services wurde jedoch darauf geachtet, dass möglichst die bevorzugte Laufzeitumgebung der Anwendungen genutzt wird.

7.2.2 Laufzeitumgebung von Frontend-Instanzen

Das SSELab wurde als verteiltes webbasiertes Framework entwickelt (vgl. Anforderung **FA47**). Genau wie die Backend-Instanzen wird daher für die Ausführung einer Frontend-Instanz auch eine Serverinfrastruktur zur Laufzeit benötigt. Die wesentlichen Bestandteile dieser Laufzeitumgebung sind in Abbildung 7.5 in einem Verteilungsdiagramm zusammengefasst.

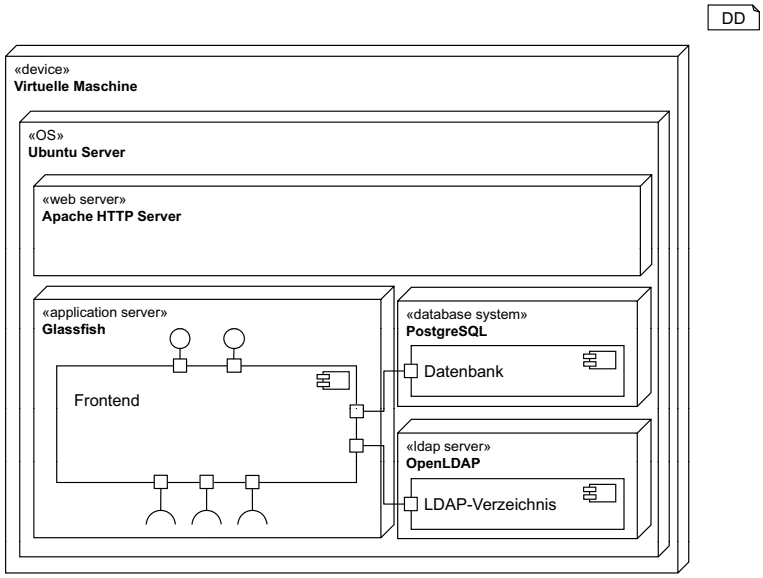


Abbildung 7.5: Laufzeitumgebung einer Frontend-Instanz

Genau wie bei den Backend-Instanzen wird auch für eine Frontend-Instanz aktuell eine virtuelle Maschine mit einem Ubuntu Linux als Betriebssystem verwendet. Das Frontend selbst ist als Enterprise Java Anwendung (Java EE 5 [JEE06]) realisiert und benötigt daher einen Application Server. Aktuell wird GlassFish [Gla12] eingesetzt. Andere kompatible Application Server könnten jedoch auch genutzt werden, wie beispielsweise JBoss. Zur Verwaltung der persistenten Daten werden einerseits PostgreSQL als Datenbanksystem und andererseits OpenLDAP als LDAP-Server verwendet. Über die Datenbank werden die Objekte aller Domänenklassen verwaltet und mit Hilfe des LDAP-Servers können Anwendungen Anfragen zur Authentifizierung

und Autorisierung durchführen. Hier könnten alternativ auch jede andere relationale Datenbank sowie ein anderer LDAP-Server eingesetzt werden.

Neben dem Application Server wird auf der Maschine auch ein Apache Web-Server betrieben. Dieser Server wird als sogenannter Integration Reverse Proxy [Som03] eingesetzt, so dass sowohl die Webseiten und Web-Services der Frontend-Instanz als auch die Webseiten der integrierten Base-Services aller Base-Backend-Instanzen darüber verfügbar gemacht werden. Zusätzlich wird dieser Apache für die Verwaltung von Zertifikaten sowie für die Umsetzung von Single-Sign-On auf Grundlage von HTTP Basic Authentication eingesetzt.

7.2.3 Sicherheitsaspekte

Bei dem Betrieb von Software as a Service ergeben sich immer Fragestellungen in Bezug auf Sicherheitsaspekte. In diesem Abschnitt werden daher einige dieser Aspekte auf Grundlage der in Abschnitt 3.3 angegebenen Anforderungen und der zuvor beschriebenen Laufzeitumgebung von SSELab-Instanzen behandelt.

In den Anforderungen **NFA1** und **NFA9** wird gefordert, dass Benutzern und Administratoren der Zugriff auf die geschützten Seiten einer SSELab-Instanz nur nach erfolgreicher Authentifizierung gestattet werden soll. Die Authentifizierung wird in einer SSELab-Instanz unter Verwendung von HTTP Basic Authentication [FHBH+99] durch den Integration Reverse Proxy durchgeführt. Dadurch kann Single-Sign-On auf einfache Weise unterstützt werden, um Anforderung **FA33** zu erfüllen. Da HTTP Basic Authentication aus Sicherheitsgründen jedoch nur über eine verschlüsselte Verbindung genutzt werden sollte, wurde der Proxy gemäß Anforderung **NFA2** so konfiguriert, dass für die Kommunikation mit einem Web-Browser und den Framework-Clients HTTPS unter Verwendung eines signierten Zertifikats genutzt wird, so dass die Kommunikation auf Ebene der Transportschicht gesichert ist. Innerhalb des Frameworks werden gemäß Anforderung **NFA12** ebenfalls TLS verschlüsselte Verbindungen, in diesem Fall mit beidseitigem Handshake unter Verwendung von Client-Zertifikaten, genutzt. Dadurch ist die Kommunikation mit den Backend-Instanzen ausschließlich durch die zugehörige Frontend-Instanz möglich.

Die Autorisierung innerhalb einer SSELab-Instanz erfolgt auf Grundlage der Standardmechanismen der verwendeten Web- und Application-Server, so dass hier auf bewährte Konzepte zurückgegriffen werden konnte. Insbesondere wurde darauf geachtet, dass ein möglichst einfaches Rechte- und Rollenkonzept genutzt wird, dass es durch eine kompakte, leicht nachvollziehbare Konfiguration umgesetzt werden kann, damit Konfigurationsfehler ausgeschlossen werden können. Darüber hinaus wurde bei der Integration von Anwendungen darauf geachtet, dass deren persistente Daten strikt voneinander getrennt sind, so dass die Mandantentrennung für Projekte gewährleistet werden kann, wie in Anforderung **NFA4** gefordert und in Abschnitt 7.1.1 beschrieben.

Bei dem Betrieb der im Rahmen dieser Arbeit erzeugten SSELab-Instanzen wurde darauf geachtet, dass alle genutzten Server- und Software-Komponenten auf dem aktuellen Stand sind. Die Aktualisierung der integrierten Werkzeuge wurde bereits in Abschnitt 6.3 diskutiert. Die Aktualisierung der Laufzeitumgebung einer SSELab-Instanz wurde über das Paketverwaltungssystem des Betriebssystems vorgenommen, so dass neue Versionen und Security-Updates auf einfache Weise und zeitnah installiert werden konnten.

Im Rahmen dieser Arbeit war immer ein Vertrauensverhältnis zwischen den Service-Entwicklern und den Administratoren vorhanden, da alle Rollen durch Mitglieder des Lehrstuhls für Software Engineering eingenommen wurden. Dementsprechend gibt es aktuell keine speziellen Mechanismen, um Service-Plug-ins oder die zugehörigen Werkzeuge gemäß Abschnitt 4.3.2 auf schadhaften Code zu überprüfen. Um fehlerhaften Code zu eliminieren, wurden alle Service-Plug-ins vor der Installation in eine Instanz ausführlich getestet und nach der Installation zunächst für ausgewählte Projekte aktiviert und erst später freigeschaltet. Dadurch konnten Fehler in den Service-Plug-ins im Allgemeinen effizient erkannt und behoben werden. Falls jedoch zukünftig Service-Plug-ins auch von Dritten entwickelt werden sollen, müssten darüber hinaus geeignete Testbetrieb- und Sandboxing-Verfahren methodisch und technisch etabliert werden, da sonst Service-Plug-ins erheblichen Schaden in den Backend-Instanzen und den zugehörigen Server-Maschinen anrichten könnten.

7.3 Methodik des Deployments

In diesem Abschnitt wird der Deployment-Prozess einer SSELab-Instanz inklusive der Services und der Clients auf technischer Ebene beschrieben. Dabei wird zunächst das Deployment von Backend-Instanzen, danach das von Frontend-Instanzen und schließlich von Services und der Clients behandelt. In Anlehnung an die Definition aus [Dea07] und [CFH+98] werden dabei jeweils die Aktivitäten Release, Konfiguration, Installation und Aktivierung sowie die Aktualisierung genauer betrachtet. Weitere Aktivitäten wie die Überwachung und die Deinstallation werden hier nicht behandelt. Alle erläuterten Schritte sind dabei unabhängig von der gewählten Betriebsart gemäß Abschnitt 7.1.1.

Im den folgenden Abschnitten wird auf einige Skripte Bezug genommen, die in Entwicklungs- bzw. Source-Code-Projekten des SSELabs enthalten sind. Insbesondere die beiden Projekte `sselab-build` und `sselab-env` werden dabei referenziert.

7.3.1 Deployment von Backend-Instanzen

Für die in diesem Abschnitt beschriebene Methodik des Deployments von Backend-Instanzen wird vorausgesetzt, dass eine Servermaschine mit einem installierten Ubuntu Server oder einem vergleichbaren Betriebssystem bereits vorhanden ist. Auf die Installation und Konfiguration einer entsprechenden Maschine wird daher nicht weiter eingegangen. Der Deployment-Prozess einer Backend-Instanz umfasst mehrere zusammenhängende Schritte. In Abbildung 7.6 sind sowohl die Schritte für die erstmalige Instanziierung als auch die Aktualisierung einer Backend-Instanz zusammengefasst, die im Folgenden gemeinsam behandelt werden.

Aktualisierung der Zertifikate. Die Kommunikation zwischen einer Frontend- und einer Backend-Instanz erfolgt über eine TLS-verschlüsselte Verbindung mit beidseitigem Handshake, um Anforderung **NFA12** zu erfüllen. Die Verwaltung der dafür benötigten Zertifikate geschieht über die Komponenten der Backend-Instanzen sowie über den Application Server der Frontend-Instanz. Vor einem Release der Backend-Komponenten muss daher durch

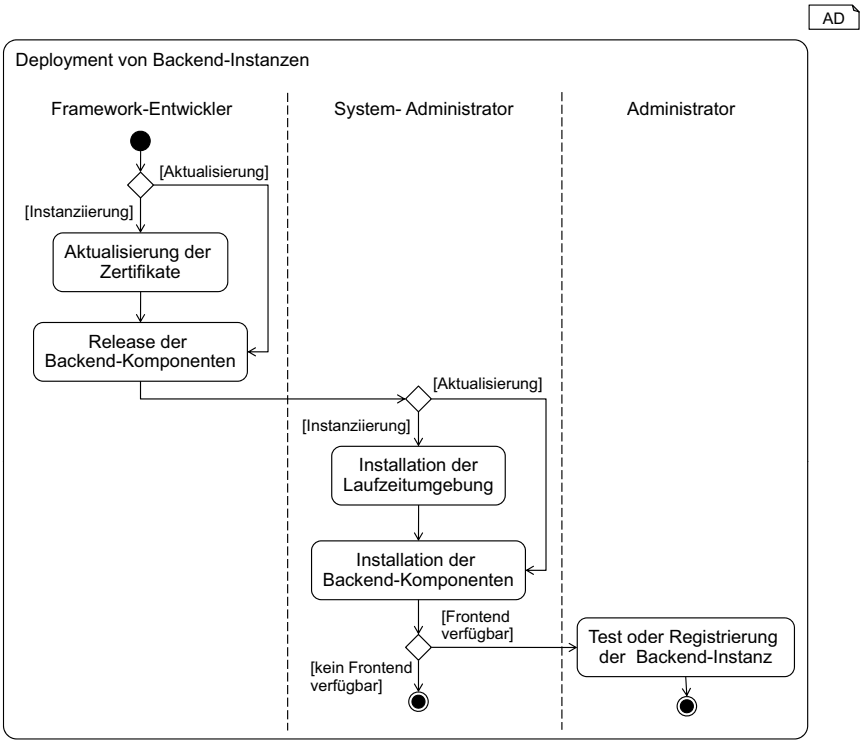


Abbildung 7.6: Methodik des Deployments von Backend-Instanzen

einen Framework-Entwickler sichergestellt werden, dass die Zertifikatskette für das Produktivsystem korrekt konfiguriert ist. Dafür existieren entsprechende Skripte in der Entwicklungsprojekten der Backend-Komponenten sowie des Projekts zur Konfiguration einer Frontend-Instanz, die bei Bedarf vor dem Release ausgeführt werden müssen.

Release der Backend-Komponenten. Durch das Release der Backend-Komponenten werden diese in einer bestimmten Version als ausführbare Java-Archive von einem Framework-Entwickler zur Verfügung gestellt. Die Erzeugung der Archive ist durch die Build-Skripte des SSELabs mit Hilfe von Apache Ant [LH07] automatisiert. Durch einen Aufruf des Targets `backend-all-publish` des zentralen Ant-Skripts (`sselab-build/build-sselab.xml`) werden die Archive aller Backends mit der richtigen Konfiguration erzeugt, so dass sie in den Backend-Instanzen installiert werden können.

Installation der Laufzeitumgebung. Die für die initiale Bildung einer Backend-Instanz notwendige Installation der in Abschnitt 7.2.1 beschriebenen Laufzeitumgebung ist ebenfalls au-

tomatisiert und kann durch einen Aufruf des Shell-Skripts `servers/felix/create-instance.sh` aus dem Entwicklungsprojekt `sselab-env` von einem System-Administrator durchgeführt werden. Der Aufruf muss dabei auf dem Produktivsystem erfolgen. Durch das Skript werden sowohl Apache Felix als auch die Komponenten des DOSGi-Projekts installiert. Darüber hinaus werden eine Komponente für das Hot-Deployment von OSGi-Bundles aus dem Dateisystem und das Logging-Framework konfiguriert sowie Skripte zum Starten und Stoppen des Servers erzeugt. Nach einem erfolgreichen Durchlauf des Skripts ist die Laufzeitumgebung einsatzbereit.

Installation der Backend-Komponenten. Die im vorigen Schritt erwähnte Komponente für das Hot-Deployment von OSGi-Bundles ermöglicht die Verwaltung von Bundles über das Dateisystem. Diese Komponente scannt ein Verzeichnis (per Default das `load`-Verzeichnis der Felix-Installation) und installiert automatisch alle Bundles, die in dieses Verzeichnis kopiert werden. Auch die Aktualisierung und das Entfernen von Bundles ist durch ein Überschreiben bzw. Entfernen der entsprechenden Dateien möglich. Die Installation oder Aktualisierung der Backend-Komponenten erfolgt daher, indem ein System-Administrator die Java-Archive in das `load`-Verzeichnis der zugehörigen Felix-Instanz kopiert. Anschließend wird der Web-Service der Backend-Instanz durch die DOSGi-Komponenten automatisch geladen und unter der zugehörigen URL veröffentlicht.

Test oder Registrierung der Backend-Instanz. Nachdem der Web-Service einer Backend-Instanz verfügbar ist, kann diese durch einen Administrator in einer laufenden Frontend-Instanz getestet bzw. registriert werden. Nach einer Aktualisierung muss nur ein automatisierter Testaufruf über das Frontend ausgeführt werden. Eine Registrierung ist im Gegensatz dazu nach der Instanziierung einer neuen Backend-Instanz notwendig. Dies geschieht wie in Abschnitt 4.4.2 beschrieben durch Angabe der zugehörigen Web-Service-URL. Bei der Registrierung wird dann automatisch der Testaufruf gestartet, so dass die Verbindung zu der neuen Backend-Instanz überprüft wird. Im Erfolgsfall werden anschließend alle Services synchronisiert, wie ebenfalls in Abschnitt 4.4.2 beschrieben. Falls noch keine Frontend-Instanz verfügbar ist, entfällt dieser Schritt.

7.3.2 Deployment von Frontend-Instanzen

Wie zuvor wird auch für das Deployment einer Frontend-Instanz eine existierende Servermaschine mit Ubuntu Server oder einem vergleichbaren Betriebssystem vorausgesetzt. Die nachfolgend behandelten Schritte zur Instanziierung und zur Aktualisierung einer Frontend-Instanz sind im Aktivitätsdiagramm in Abbildung 7.7 gezeigt.

Release der Frontend-Komponenten. Beim Release der Frontend-Komponenten durch einen Framework-Entwickler wird ein Java Enterprise Archive (EAR) erzeugt, das in einem Application Server installiert werden kann. Das EAR enthält alle Frontend-Komponenten als Web- oder als Java-Archive, die zur Laufzeit benötigt werden. Die Erzeugung des EARs ist ebenfalls durch Ant-Build-Skripte automatisiert und kann durch den Aufruf des Targets `frontend-publish` des zentralen Skripts im Entwicklungsprojekt `sselab-build` von einem Framework-Entwickler durchgeführt werden.

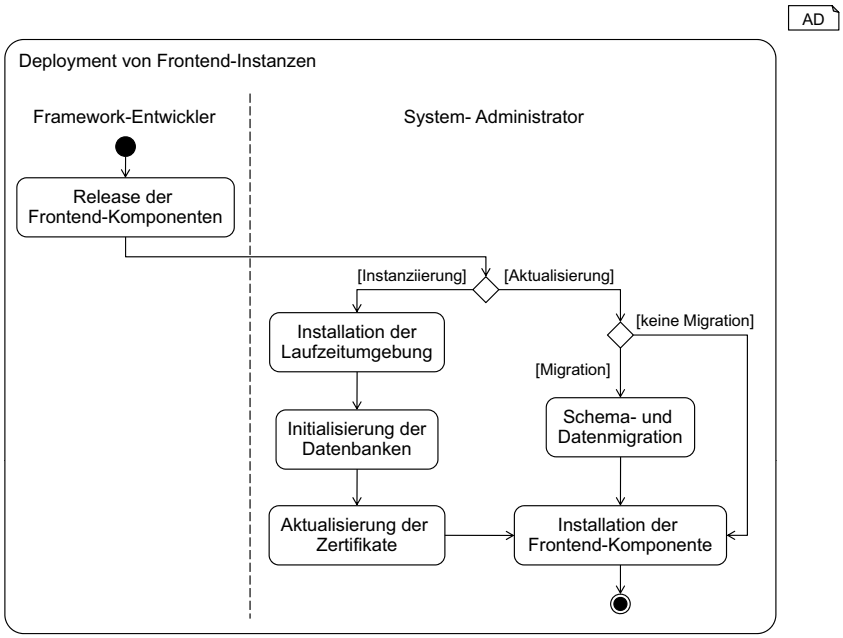


Abbildung 7.7: Methodik des Deployments von Frontend-Instanzen

Installation der Laufzeitumgebung. Die Installation der Laufzeitumgebung einer Frontend-Instanz ist über mehrere Shell-Skripte automatisiert. Diese Skripte befinden sich in dem Entwicklungsprojekt `sselab-env` und können von einem System-Administrator aufgerufen werden, um die Installation und Konfiguration von GlassFish, Apache HTTPD, OpenLDAP und PostgreSQL durchzuführen. Nach dem Aufruf der Skripte sind alle Server konfiguriert und einsatzbereit.

Initialisierung der Datenbanken. Bei der Erzeugung einer neuen Frontend-Instanz müssen die Datenbanken initialisiert werden. Die Schemata für die SQL- und die LDAP-Datenbanken sind in den entsprechenden Verzeichnissen des Projekts `sselab-env` enthalten und können über Skripte von einem System-Administrator zur Initialisierung der Datenbanken verwendet werden.

Aktualisierung der Zertifikate. Über die Laufzeitumgebung des Frontends werden einerseits die Zertifikate für die Kommunikation mit den Backend-Instanzen und andererseits die Zertifikate für den Zugriff auf die Webseiten und die Web-Services der Frontend-Instanz verwaltet. Der erste Fall wurde bereits im vorigen Abschnitt im Kontext des Deployments von Backend-Instanzen beschrieben. Da Apache als Integration Reverse Proxy eingesetzt wird, erfolgt die Verwaltung der Zertifikate für den zweiten Fall über die Apache-Konfiguration,

in der die Zertifikatskette konfiguriert und dann für den Aufbau von HTTPS-Verbindungen verwendet wird. Die Erzeugung dieser Konfiguration ist ebenfalls automatisiert und kann durch einen System-Administrator auf dem Produktivsystem durchgeführt werden.

Schema- und Datenmigration. Falls bei der Weiterentwicklung des Frontends die Datenbankschemata verändert worden sind, müssen für eine Aktualisierung zunächst die Datenbanken migriert werden. Für jede Änderung an dem Datenbankschema werden durch die Framework-Entwickler im Entwicklungsprozess entsprechende Skripte erzeugt, durch die die Schema- und Datenmigration automatisiert durchgeführt werden können. Vor jeder Änderung der Produktivdatenbanken sollte jedoch zunächst eine zusätzliche Sicherungskopie durch einen System-Administrator erzeugt werden, so dass bei Problemen schnell ein lauffähiges System wiederhergestellt werden kann.

Installation der Frontend-Komponente. Der GlassFish Application Server unterstützt ein Hot-Deployment von Enterprise Archiven. Daher muss zur Installation oder Aktualisierung nur das Archiv der Frontend-Komponente von einem System-Administrator in dessen Verzeichnis `autodeploy` kopiert werden. Die Anwendung wird dann automatisch durch den Server neu geladen und kann anschließend über einen Browser aufgerufen und getestet werden.

7.3.3 Deployment von Services

Nachdem ein Werkzeug gemäß Abschnitt 5.2 als Service in das Framework integriert worden ist, kann es über eine SSELab-Instanz den Benutzern zur Verfügung gestellt werden. Die Schritte, die für das Deployment eines Services durchgeführt werden müssen, sind in Abbildung 7.8 zusammengefasst.

Release des Service-Plug-ins. Service-Plug-ins bestehen aus einem Java-Archiv und benötigen gegebenenfalls weitere Archive, die als Laufzeitabhängigkeiten in einer Backend-Instanz installiert sein müssen. Die Erzeugung der Archive erfolgt automatisiert über die Build-Skripte des jeweiligen Entwicklungsprojekts durch einen Service-Entwickler. Über das zentrale Skript des SSELabs können durch den Aufruf des Targets `plugins-all-publish` alle in einer Entwicklungsumgebung verfügbaren Service-Plug-ins auf einmal erzeugt werden.

Installation und Konfiguration der Anwendung. Falls ein Base- oder ein Ostp-Service zum ersten mal instanziiert wird, muss die zugehörige Anwendung bzw. das Werkzeug installiert und konfiguriert werden. Dies geschieht mit Hilfe von Skripten, die für jeden Service bei dessen Integration gemäß Abschnitt 5.2 erzeugt worden sind. Nach der Ausführung dieser Skripte auf der Servermaschine der zugehörigen Backend-Instanz durch einen System-Administrator können die Instanzen der Anwendungen bzw. der Werkzeuge durch das zugehörige Service-Plug-in verwaltet werden.

Upload der Artefakte. Die Instanziierung bzw. Aktualisierung eines Services wird durch den Upload der entsprechenden Java-Archive über eine Frontend-Instanz abgeschlossen. Dies

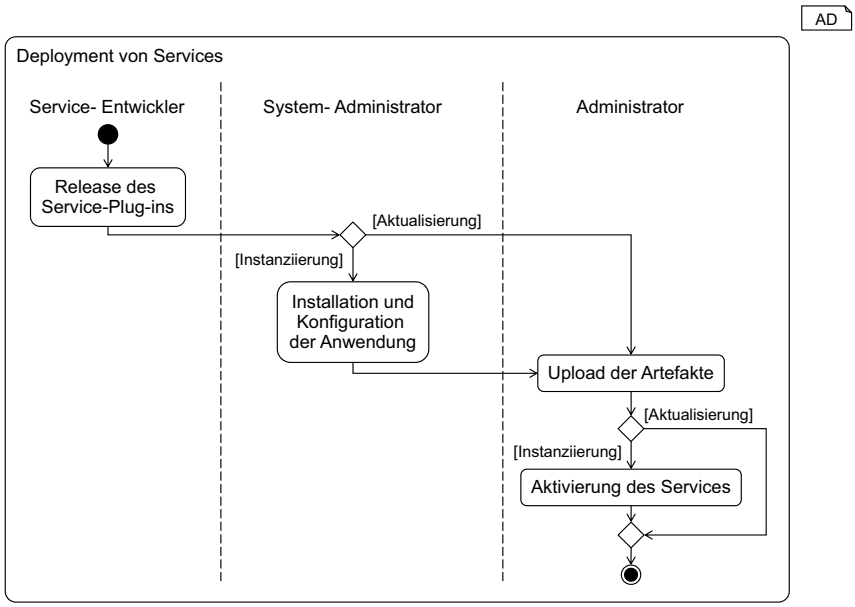


Abbildung 7.8: Methodik des Deployments von Services

kann durch einen Administrator entweder über die Webseiten der Frontends oder mit Hilfe des in Abschnitt 4.5.2 beschriebenen Admin-Clients durchgeführt werden.

Aktivierung des Services. Nach der initialen Installation eines Service-Plug-ins durch einen Administrator ist der zugehörige Service noch nicht aktiviert und kann daher von den Benutzern der SSELab-Instanz noch nicht ohne Weiteres genutzt werden. Im Anschluss an eine erfolgreiche Testphase, in dem der Service beispielsweise in ausgewählten Projekten genutzt wird, sollte ein Administrator den Service aktivieren und damit zur allgemeinen Nutzung freigeben.

7.3.4 Deployment von Clients

Die Clients des SSELab-Frameworks können über das Frontend verwaltet werden, wie in Abschnitt 4.4.5 beschrieben. Nach der Entwicklung eines Clients durch einen Client-Entwickler kann der in Abbildung 7.9 gezeigt Deployment-Prozess durchlaufen werden, um diesen über eine SSELab-Instanz verfügbar zu machen.

Release des Clients. Die Verwaltungsmechanismen im Frontend sind darauf ausgelegt, das ein Client aus einer einzelnen Datei besteht. Dabei kann es sich beispielsweise um ein Java- oder ein Zip-Archiv handeln. Die Build-Skripte der aktuell verfügbaren Client-Projekte

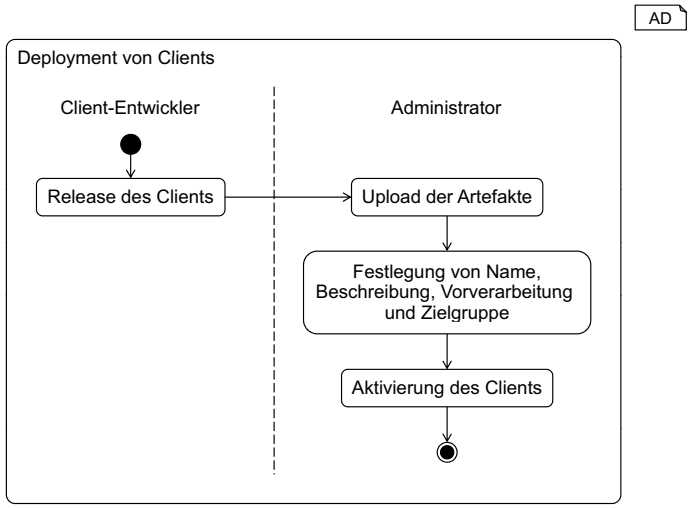


Abbildung 7.9: Methodik des Deployments von Clients

erlauben jeweils die Erzeugung eines entsprechenden Archivs. Durch das zentrale Build-Skript können alle Clients durch den Aufruf des Targets `clients-all-publish` durch einen Client-Entwickler erzeugt werden.

Upload der Artefakte. Sobald ein Client-Archiv verfügbar ist, kann es durch einen Administrator über die Webseiten einer Frontend-Instanz hochgeladen werden. Dabei werden die Archive im Dateisystem des Frontends zunächst temporär gespeichert, wie in Abschnitt 4.4.5 beschrieben.

Festlegung von Name, Beschreibung, Vorverarbeitung und Zielgruppe. Zusätzlich zum Archiv selbst müssen noch einige Informationen über den Client nach dessen Upload durch einen Administrator angegeben werden. Dazu gehören der Name des Clients, eine Beschreibung, die Art der Vorverarbeitung sowie die Zielgruppe (Administratoren oder Benutzer). In Abhängigkeit von der gewählten Vorverarbeitungsstrategie wird dann der Client noch gegebenenfalls zur Verwendung von Bot-Accounts konfiguriert, wie in Abschnitt 4.4.5 diskutiert.

Aktivierung des Clients. Im Anschluss daran kann ein Client noch nicht unmittelbar von Personen der Zielgruppe heruntergeladen werden. Erst nach der Aktivierung des Clients durch einen Administrator wird dieser endgültig verfügbar gemacht und kann genutzt werden. Die Aktivierung erfolgt ebenfalls über die Webseiten der Fronten-Instanz.

7.4 Zusammenfassung

In diesem Kapitel wurde die Methodik zur Bildung von Instanzen des SSELab-Frameworks beschrieben. Zunächst wurden auf Grundlage der Verteilungsmöglichkeiten der Framework-Komponenten die verschiedenen Betriebsarten vorgestellt, die den Betrieb einer SSELab-Instanz für Kunden in Abhängigkeit von deren Anforderungen an die Datensicherheit und den Datenschutz ermöglichen. Zusätzlich wurde auf die a posteriori Integration von Werkzeugen eingegangen, bei der eine SSELab-Instanz zur Verwaltung existierender und bereits genutzter (Legacy-) Anwendungen eingesetzt wird.

Im Anschluss daran wurden die Laufzeitumgebungen der Komponenten des Frameworks kompakt beschrieben und darauf aufbauend die Methodik des Deployments von Backend- und Frontend-Komponenten sowie von Services und Clients behandelt. Hervorzuheben ist dabei die Automatisierung der Deployment-Prozesse, die eine einfache und schnelle Instanziierung und Aktualisierung der Komponenten ermöglicht. Diese Eigenschaft ist insbesondere im Bereich von Software as a Service von großer Bedeutung, da es sich dabei um Produktivsysteme handelt, die dauerhaft verfügbar sein müssen. Fehler in einem Deployment-Prozess führen in vielen Fällen zu einem unerwünschten temporären Ausfall.

Kapitel 8

Nutzungs- und Betriebskonzept der sselab.de-Instanz

Im Laufe der Entwicklung des SSELab-Frameworks wurden mehrere Instanzen gebildet und erfolgreich für die Durchführung diverser Projekte genutzt. In diesem Kapitel wird eine dieser Instanzen genauer behandelt. Diese wurde im Laufe des Jahres 2009 erzeugt und unter der URL `http://sselab.de` verfügbar gemacht, so dass sie im Folgenden auch als sselab.de-Instanz bezeichnet wird. Seit der Erzeugung wurden kontinuierlich Projekte in der Lehre, der Forschung und zur Kooperation mit industriellen Partnern mit Hilfe der Instanz durchgeführt. Zum Zeitpunkt der Erstellung dieser Ausarbeitung existieren über 1400 Benutzerkonten und über 600 aktivierte Projekte. In Abbildung 8.1 ist sowohl die Zunahme der Benutzerkonten als auch der aktivierten Projekte über die Zeit gezeigt.

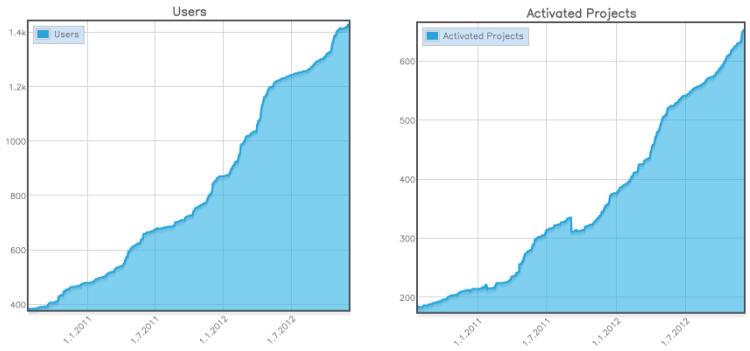
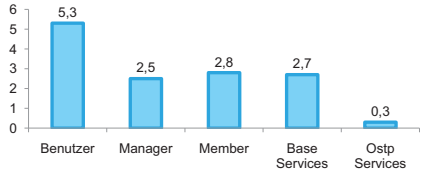


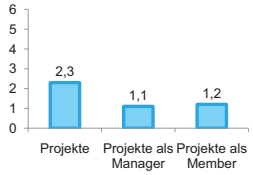
Abbildung 8.1: Anzahl der Benutzerkonten und der aktivierten Projekte (Stand: 07.12.2012)

Einige weitere Statistiken zu der Nutzung der Instanz sind in den folgenden Abbildungen zusammengefasst. Abbildung 8.2a zeigt die durchschnittliche Anzahl von Benutzern und Services pro Projekt. Der Wert von 5,3 Benutzern pro Projekt verdeutlicht, dass die Instanz nicht nur für umfangreichere Projekte, sondern auch für kleine Projekte genutzt wird. Beispielsweise werden viele studentische Arbeiten mit Hilfe der Instanz durchgeführt. In den zugehörigen Projekten sind dann oft nur ein Studierender und die Betreuer enthalten. Der geringe Wert bei den Ostp-Services pro Projekt ergibt sich daraus, dass Ostp-Services auch unabhängig von Projekten genutzt werden können. Abbildung 8.2b zeigt die durchschnittliche Anzahl von Projekten pro

Benutzer. Diese Werte verdeutlichen, dass neben Benutzern, die die Instanz für mehrere Projekte nutzen, auch Benutzer existieren, die nur wenige Projekte damit durchführen. Dies ergibt sich ebenfalls aus dem Einsatz in der Lehre, beispielsweise in Praktika oder in Übungen.



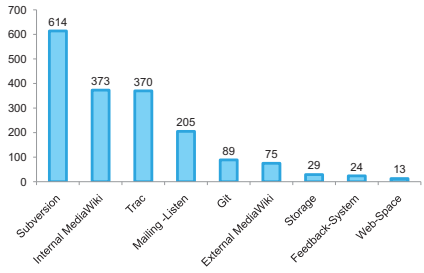
(a) Durchschnittliche Anzahl von Benutzern und Services pro Projekt



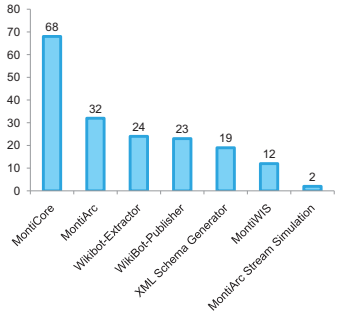
(b) Durchschnittliche Anzahl von Projekten pro Benutzer

Abbildung 8.2: Statistiken zu Projekten, Services und Benutzern (Stand: 13.02.2013)

Abbildung 8.3 illustriert, wie oft die Services der Instanz genutzt werden. In Abbildung 8.3a ist die Nutzung der Base-Services dargestellt. Fast alle Projekte nutzen Subversion als Versionsverwaltungssystem, da dieser Service von Beginn an verfügbar war. Git wurde im Gegensatz dazu deutlich später in die Instanz integriert und freigeschaltet. Die geringe Zahl bei den Services Storage, Feedback-System und Web-Space ist damit zu erklären, dass diese auch erst kürzlich in die Instanz integriert worden sind und bis zum Zeitpunkt der Erstellung dieser Ausarbeitung noch nicht zur allgemeinen Verwendung freigeschaltet worden sind. Abbildung 8.3b zeigt die Anzahl der Projekte, in denen die Ostp-Services aktiviert sind. Hier ist die geringe Anzahl ebenfalls damit zu erklären, dass diese Services auch unabhängig von Projekten genutzt werden können.



(a) Nutzung von Base-Services in Projekten



(b) Nutzung von Ostp-Services in Projekten

Abbildung 8.3: Nutzung von Services der Instanz in Projekten (Stand: 13.02.2013)

In den folgenden Abschnitten werden einige weitere Aspekte der sselab.de-Instanz diskutiert. Zunächst werden die primär unterstützen Projektarten anhand einiger repräsentativer Projekte beschrieben. Darauf folgend wird eine Einführung in das Nutzungskonzept gegeben. Abschließend wird aufbauend auf dem vorigen Kapitel 7 das Betriebskonzept der Instanz skizziert.

8.1 Unterstützte Projektarten

Wie in Kapitel 7 beschrieben, prägen die Services einer Framework-Instanz deren primären Einsatzzweck. Die sselab.de-Instanz wurde gemäß Anforderung **FA19** (Verschiedene Projektarten) durch die installierten Services hauptsächlich für die Unterstützung von Software Engineering Projekten ausgelegt. Darüber hinaus wurde sie jedoch auch erfolgreich für die Durchführung von Projekten anderer Art eingesetzt. Beispiele dafür sind die Erstellung wissenschaftlicher Beiträge – beispielsweise Papiere, Dissertationen und Bücher, oft mit international verteilten Autorengruppen – und die Verwaltung von Literaturdatenbanken sowie die Organisation und Durchführung von Lehrveranstaltungen und studentischen Abschlussarbeiten. Darüber hinaus wurden mit Hilfe der Instanz auch Journale, Konferenzen und Workshops organisiert – beispielsweise das internationale Journal *Software and Systems Modeling (SoSyM)*, die Multikonferenz *Software Engineering 2013* sowie die Workshops *Modeling in Software Engineering (MiSE)* 2009 und 2013 und *Entwicklung und Evolution von Forschungssoftware (EEFSW)* 2011.

In den folgenden Abschnitten werden zur Veranschaulichung des primären Einsatzzwecks einige konkrete Forschungsprojekte des Lehrstuhls für Software Engineering kompakt beschrieben, bei denen die Projektarbeit durch die Services der sselab.de-Instanz unterstützt wird.

8.1.1 GloSE

Im Forschungsprojekt GloSE (Global Software Engineering) [BBH+09] kooperieren die Software Engineering Lehrstühle der RWTH Aachen, Leibniz Universität Hannover, TU München und TU Clausthal, um Herausforderungen im Kontext global verteilter Software-Entwicklungsprojekte zu behandeln. Dabei werden sowohl methodische als auch technische Lösungen erarbeitet, die die Projektorganisation, die Prozesse und Informationsflüsse sowie die Dokumente und Informationsartefakte gleichermaßen berücksichtigen.

Das SSELab-Framework wurde im Rahmen des GloSE-Projekts ebenfalls weiterentwickelt und an die Bedürfnisse verteilter Projekte angepasst sowie um entsprechende Services ergänzt. Beispielsweise wurde das Feedback-System im Kontext dieses Projekts entwickelt und unter anderem für die Evaluation verteilter werkzeuggestützter Abnahmetests genutzt [LHK+12]. Darüber hinaus wurde die sselab.de-Instanz im Rahmen des Projekts bei der Erstellung des Antrags zur Verwaltung der Dokumente, Literatur und Protokolle eingesetzt. Für die Durchführung verteilter Praktika mit Studierenden aller Partneruniversitäten [DHH+11] wurden die Services der Instanz für das Projektmanagement und die Entwicklung genutzt. Auch bei der Erarbeitung der in diesem Abschnitt referenzierten wissenschaftlichen Beiträge kam diese Instanz zum Einsatz.

8.1.2 Energie Navigator

Der Energie Navigator [KPLR12, FPPR12, FLP+11, FKP+10] ist eine Plattform zur Planung und Überwachung von Gebäuden. Mit Hilfe der Plattform kann die Funktionsweise der technischen Anlagen eines Gebäudes regelbasiert spezifiziert und im späteren Betrieb automatisch überprüft werden. Dazu werden die Sensordaten des Gebäudes erfasst und aufbereitet, so dass Analysen durchgeführt und die Einhaltung der Spezifikation geprüft werden können. Bei einer Verletzung der Spezifikation werden entsprechende Berichte erzeugt. Das Ziel der Plattform ist, sowohl den

Energieverbrauch als auch den Nutzungskomfort von Gebäuden zu optimieren. Die Plattform wird im Rahmen mehrerer Forschungsprojekte von der synavision GmbH in Kooperation mit dem Lehrstuhl für Software Engineering, dem Institut für Gebäude- und Solartechnik der TU Braunschweig und energydesign braunschweig GmbH entwickelt.

Die sselab.de-Instanz wird einerseits für die Entwicklung der Plattform verwendet und andererseits für das Management der verschiedenen Entwicklergruppen, die an den einzelnen Teilprojekten arbeiten. Bei der Entwicklung wird ein agiler iterativer Prozess mit testgetriebener Entwicklung und Continuous Integration eingesetzt. Im Kontext des Energie Navigator Projekts wurden außerdem mehrere Lehrveranstaltungen durchgeführt und Publikationen verfasst, für die jeweils auch die Instanz verwendet worden ist.

8.1.3 SensorCloud

Das SensorCloud-Projekt [SC12] hat das Ziel, eine Cloud-basierte Plattform für die Anbindung von Sensoren und Aktoren sowie die Verarbeitung der zugehörigen Massendaten aus verschiedenen Domänen (unter anderem Verkehr, Energie, Umwelt, Mobilität) zu entwickeln. Insbesondere die sichere und vertrauenswürdige Nutzung der gesammelten und aggregierten Daten durch Dritte steht dabei im Vordergrund, damit innovative Anwendungen auf Grundlage der SensorCloud-Daten realisiert werden können.

Im Rahmen des SensorCloud-Projekts wird die sselab.de-Instanz für das Projektmanagement eingesetzt. Die einzelnen Arbeitspakete werden in jeweils eigenen Projekten organisiert. Dabei wird ein iterativer Prozess zur Kooperation der Projektpartner eingesetzt. Da die Partner an mehreren Standorten gleichzeitig an dem Projekt arbeiten, werden auch hier Aspekte der verteilten Kooperation und Kollaboration durch die Instanz unterstützt.

8.1.4 Weitere Projekte

Neben den vorgestellten Forschungsprojekten, bei denen die Kooperation mit externen Partnern im Vordergrund steht, werden auch interne Forschungs- und Entwicklungsprojekte mit Hilfe der Instanz durchgeführt. Einige Beispiele dafür sind die in Abschnitt 6.1 bereits beschriebenen Werkzeuge MontiCore, MontiArc und MontiWIS sowie das SSELab selbst. All diese Werkzeuge wurden bzw. werden noch immer im Kontext mehrerer Promotionen realisiert. Die Instanz wurde dabei einerseits für deren Entwicklung und andererseits für die Verfassung der schriftlichen Ausarbeitung verwendet.

8.2 Einführung in das Nutzungskonzept der Instanz

In diesem Abschnitt wird eine Einführung in die Nutzung der sselab.de-Instanz anhand der Web-Oberfläche des Frontends gegeben. Dabei werden einige Aspekte und Richtlinien bei der Nutzung der Instanz durch einen neuen Benutzer hervorgehoben. Die Einführung erfolgt auf Grundlage der zum Zeitpunkt dieser Ausarbeitung unter `http://sselab.de` laufenden Instanz, die der aktuellsten Version des 1.0-Entwicklungszweigs entspricht. In diesem Entwicklungszweig sind jedoch nicht alle in dieser Arbeit beschriebenen Funktionen enthalten, da es sich um den letzten

stabilen Zweig handelt, auf dessen Grundlage ein Deployment der Komponenten des Frameworks durchgeführt worden ist. Alle beschriebenen Funktionen sind im Entwicklungszweig der Version 1.1 enthalten, die jedoch noch nicht in den produktiven Einsatz gebracht worden ist.

Im nachfolgenden Abschnitt wird die Nutzung des Frontends der sselab.de-Instanz für Gäste und Benutzer behandelt. Anschließend wird ein Überblick über die Verwaltungsfunktionen für Administratoren gegeben und die Nutzung der Services durch externe Werkzeuge behandelt.

8.2.1 Erste Schritte der Nutzung für Gäste und Benutzer

Abbildung 8.4 zeigt die Seite, die nach einem Aufruf der URL `http://sselab.de` angezeigt wird. Darauf sind einige allgemeine Informationen zum SSELab und Links zu Seiten mit ausführlicher Dokumentation der Funktionen und der verfügbaren Services enthalten.

Abbildung 8.4: Startseite der sselab.de-Instanz

Über die Startseite können außerdem neue Projekte beantragt, Benutzerkonten erstellt und bei Bedarf ein neues Passwort angefordert werden. Für die Nutzung aller weiteren Funktionen und

der Services der Instanz muss ein Benutzerkonto angelegt werden. Dabei werden einige grundlegende Information vom dem Benutzer erfragt, wie im Screenshot in Abbildung 8.5 gezeigt ist. Dazu gehören der Vor- und der Nachname sowie eine gültige E-Mail-Adresse. Außerdem muss der Benutzer einen Kontonamen wählen, über den er innerhalb der sselab.de-Instanz identifiziert wird. Bei der Wahl eines Kontonamens sollte ein sprechender Name gewählt werden, da in einigen Services dieser Name als einzige Information zur Identifikation eines Benutzers verwendet wird. Die angegebene E-Mail-Adresse wird von der sselab.de-Instanz sowie von einigen der integrierten Services verwendet, um dem Benutzer Nachrichten zu senden. Nach der Angabe eines Passworts und dessen Bestätigung kann schließlich das Konto durch den Benutzer erstellt werden. Im Anschluss daran ist ein Login über die Startseite mit Hilfe des Kontonamens und des Passworts möglich.

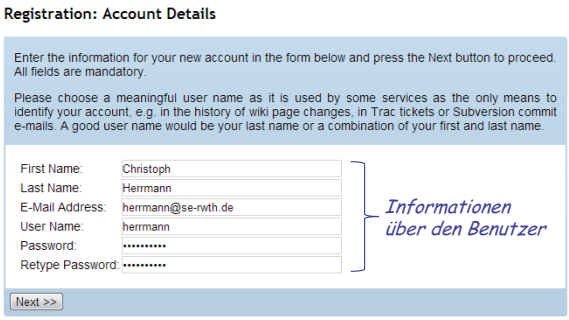


Abbildung 8.5: Ausschnitt der Webseite zur Erstellung eines neuen Benutzerkontos

Nach dem erfolgreichen Login eines Benutzers wird zunächst die Übersichtsseite angezeigt, die in Abbildung 8.6 dargestellt ist. Über diese Seite kann ein Benutzer auf die wichtigsten Funktionen der Instanz zugreifen. Dazu gehören die Anzeige und die Verwaltung der eigenen Projekte sowie die Nutzung der verfügbaren Services und Clients. Weiterhin sind die Seiten zur Verwaltung des eigenen Kontos sowie Seiten zur Kontaktaufnahme mit den Administratoren und weiterführende Dokumentation verlinkt.

Die Beantragung neuer Projekte ist ebenfalls von der Übersichtsseite aus möglich. Dabei werden einige Informationen von dem Benutzer abgefragt. Dazu gehören unter anderem Informationen über das neu zu erstellende Projekt und welche Services genutzt werden sollen. In Abbildung 8.7 sind Ausschnitte aus der Web-Oberfläche gezeigt, durch die diese Informationen erfragt werden. Die grundlegenden Informationen über ein Projekt sind dessen Name, eine Beschreibung und optional ein Ablaufdatum des Projekts. Projekte werden innerhalb des SSE-Labs über ihren Namen identifiziert, so dass jeder Projektname eindeutig gewählt werden muss. Dies wird bei der Angabe des Namens überprüft und bei der Wahl eines bereits vergebenen Namens, eine entsprechende Fehlermeldung angezeigt. Die Beschreibung eines Projekts wird unter anderem auch in die nachfolgend behandelten Einladungs-Mails eingefügt. Durch sie sollte daher eine kompakte und verständliche Übersicht über das Projekt gegeben werden. Die Wahl eines Ablaufdatums ist beispielsweise für kurzlebige Projekte sinnvoll. Wenn das Datum erreicht

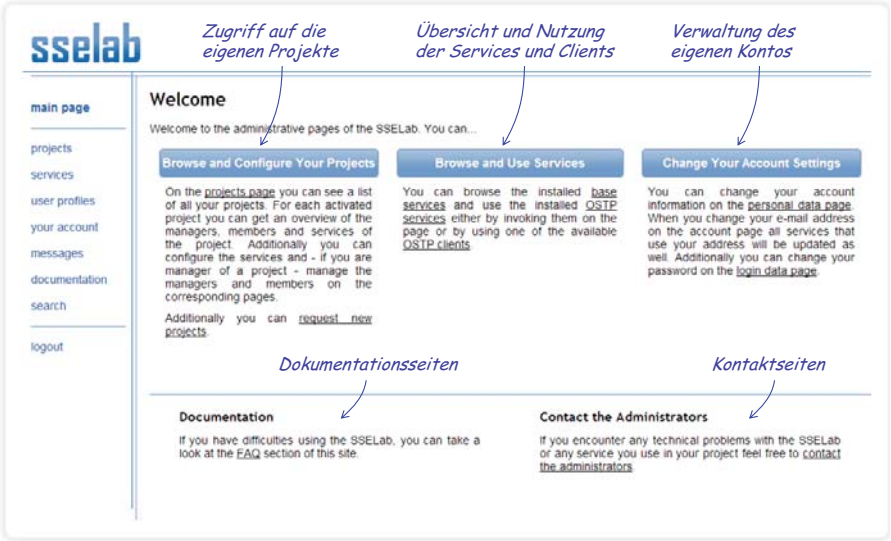


Abbildung 8.6: Überblicksseite nach erfolgreichem Login

wird, sendet die Instanz automatisch eine Erinnerung an die Projektverantwortlichen sowie die Administratoren, dass das Projekt eventuell nicht mehr benötigt wird und gegebenenfalls deaktiviert werden kann. Eine automatische Deaktivierung von Projekten könnte bei Bedarf ebenfalls angeboten werden.

Nach der Angabe der grundlegenden Informationen über ein Projekt können im nächsten Schritt die gewünschten Services ausgewählt und danach schließlich das neue Projekt angelegt werden. Dabei wird es jedoch noch nicht direkt aktiviert, sondern muss erst durch einen Administrator der Instanz freigeschaltet werden. Die Administratoren werden daher bei jeder Beantragung eines neuen Projekts benachrichtigt.

Nachdem ein beantragtes Projekt von einem Administrator aktiviert worden ist, kann es genutzt werden. Der Benutzer, der das Projekt beantragt hat, wird dem Projekt automatisch als Manager zugeordnet und kann direkt die aktivierten Services des Projekts nutzen sowie weitere Mitglieder zu dem Projekt einladen. Diese Funktionen sind über die Projektseite verfügbar. Abbildung 8.8 zeigt als Beispiel die Seite, für das in Abbildung 8.7 angelegte Projekt. Darüber können die zuvor festgelegte Beschreibung und das Ablaufdatum des Projekts geändert sowie ein Logo hochgeladen werden. Weiterhin ist eine Liste der für das Projekt aktivierten Services auf der Seite enthalten, über die der Zugriff darauf möglich ist. Im Fall von Base-Services werden hier die projektspezifischen Webseiten der Service-Instanzen verlinkt. Bei Ostp-Services ist dagegen der Web-Client des Frontend-Instanz verlinkt. Außerdem existiert auf der Seite eine Liste aller Teilnehmer des Projekts. Über die Einträge können die zugehörigen Profilseiten aufgerufen werden.

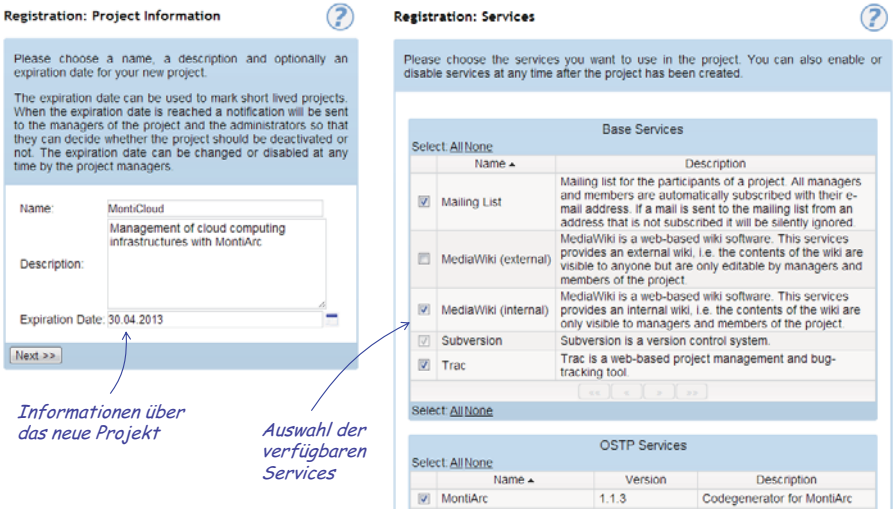


Abbildung 8.7: Ausschnitt der Webseiten zur Beantragung eines neuen Projekts

Abbildung 8.8: Projektseite des neuen Projekts

Die Manager eines Projekts können sowohl die Mitglieder als auch die Services des Projekts verwalten. Abbildung 8.9 zeigt die Webseite mit deren Hilfe die Gruppe der Manager eines Projekts administriert werden können. Auf der Seite kann die Rolle eines Managers auf Member

geändert oder auch Manager aus dem Projekt entfernt werden. Das Hinzufügen neuer Mitglieder zu einem Projekt erfolgt über den Einladungsmechanismus. In Abbildung 8.9 ist ein Ausschnitt aus dem Formular zur Versendung entsprechender Einladungen gezeigt. In dem Formular können eine oder mehrere E-Mail-Adressen der Personen eingetragen werden, die zu dem Projekt hinzugefügt werden sollen. Zusätzlich kann eine Nachricht an die Eingeladenen geschrieben werden. Das zugehörige Textfeld ist nicht in der Abbildung gezeigt. Nachdem die Einladung durch den Manager abgeschickt worden ist, erhalten die Eingeladenen eine E-Mail mit Informationen darüber, wer die Einladung versendet hat und zu welchem Projekt die Einladung gehört. Weiterhin ist ein Link auf eine Seite enthalten, auf der die Einladung angenommen oder auch abgelehnt werden kann. Bei der Annahme einer Einladung ist es möglich, entweder ein bereits existierendes Benutzerkonto zu verwenden oder ein neues zu erstellen.

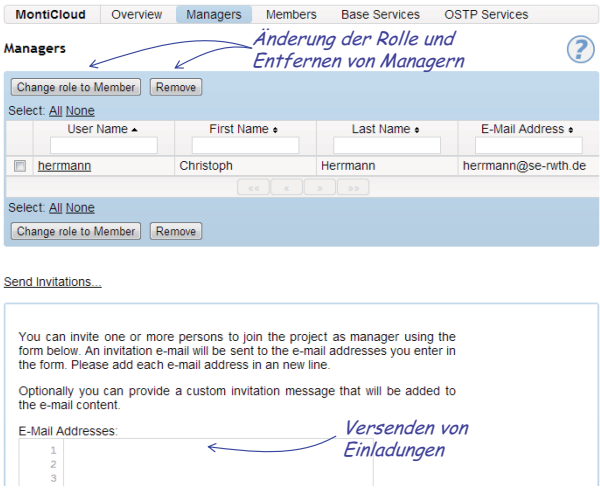


Abbildung 8.9: Seite zur Verwaltung der Manager eines Projekts

Die Verwaltung der Base- und Ostp-Services eines Projekts erfolgt jeweils über eine eigenständige Seite. Abbildung 8.10 zeigt einen Ausschnitt aus der Seite zur Verwaltung der Base-Services. Darauf sind sowohl die bereits genutzten als auch die ungenutzten Services des Projekts mit ihrer Beschreibung aufgelistet. Die verfügbaren Services können durch einen Manager jederzeit für das jeweilige Projekt aktiviert oder deaktiviert werden. Weiterhin können über diese Seite auch die Seiten zur Konfiguration der Services erreicht werden. In Abbildung 8.10 ist beispielsweise die Seite zur Konfiguration von Subversion verlinkt, auf der die in Abschnitt 6.2.1 beschriebenen Einstellungen zum Verhalten der projektweiten und benutzerspezifischen Commit-Mails sowie die Zugriffsrechte des Subversion-Repositories des Projekts geändert werden können. Weiterhin sind in Abbildung 8.10 noch die Links zu der Dokumentation der Services gezeigt, auf denen jeweils eine Übersicht über die Services gegeben wird und weiterführende Dokumentation referenziert ist.

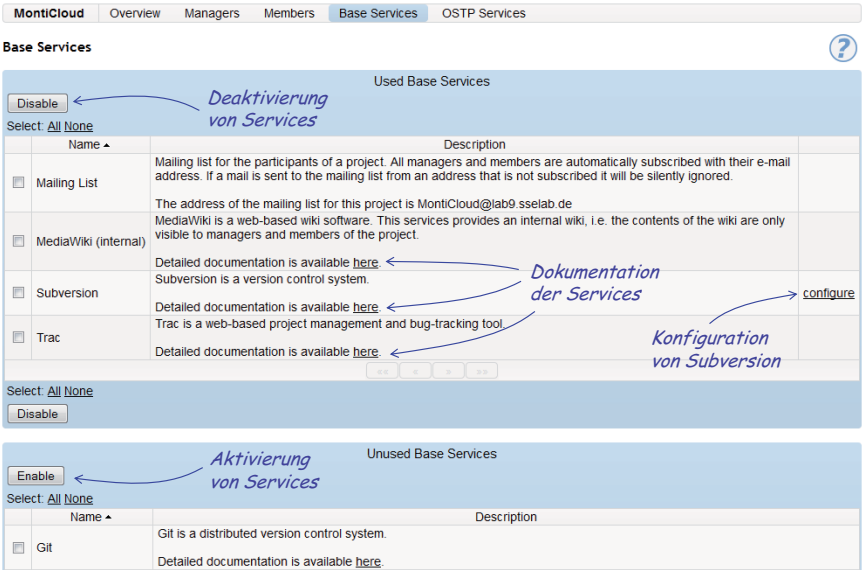


Abbildung 8.10: Seite zur Verwaltung der Base-Services eines Projekts

Auf die weiteren Funktionen der Web-Oberfläche für Benutzer wird hier nicht genauer eingegangen. Auf allen Seiten ist jedoch eine Beschreibung der enthaltenen Funktionen verfügbar. Diese kann über das Fragezeichensymbol geöffnet werden. Darüber hinaus ist weitere Dokumentation auf der Startseite sowie im FAQ-Abschnitt der SSELab-Instanz vorhanden.

8.2.2 Überblick der Funktionen für Administratoren

Die Web-Oberfläche für Administratoren ermöglicht die Verwaltung der gesamten sselab.de-Instanz. Auf alle verfügbaren Funktionen detailliert einzugehen, würde den Rahmen dieses Kapitels sprengen, so dass in diesem Abschnitt nur ein Überblick über die Nutzung der Web-Oberfläche für Administratoren gegeben wird. Abbildung 8.11 zeigt dazu die Startseite nach einem erfolgreichen Login eines Administrators.

Die einzelnen Funktionen des Frontends können über Einträge der Navigationsleiste auf der linken Seite aufgerufen werden. In der folgenden Auflistung werden diese Funktionen zusammengefasst, ohne dabei auf die zugehörigen Seiten im Detail einzugehen. Eine ausführliche Beschreibung vieler der folgenden Funktionen ist in Abschnitt 4.4 im Kontext der Architekturbeschreibung des Frontends enthalten.

admin accounts Auf den Seiten zur Verwaltung der Administratorkonten können neue Konten erzeugt sowie bestehende editiert oder gelöscht werden. Darüber hinaus kann hier das Passwort des eingeloggten Administrators geändert werden. Damit in der sselab.de-Instanz

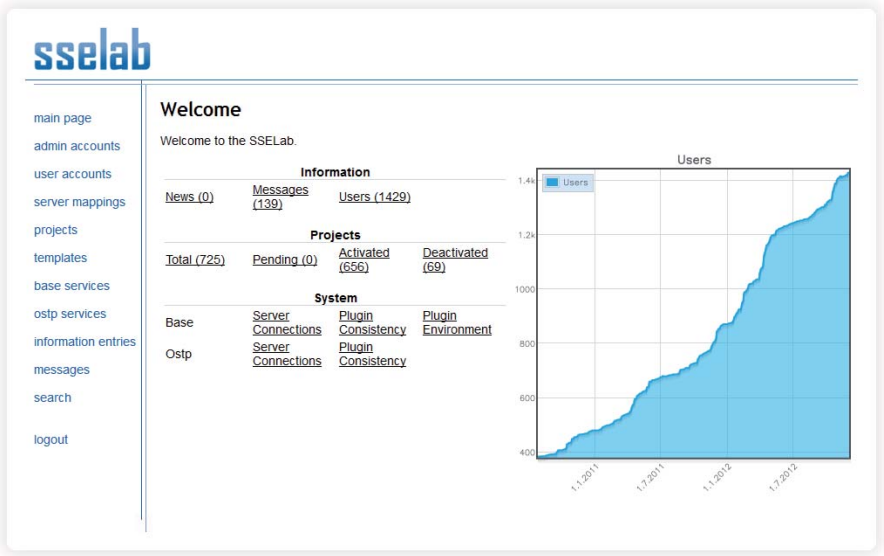


Abbildung 8.11: Startseite nach dem Login eines Administrators

immer mindestens ein Administrator existiert, gibt es ein geschütztes Administratorkonto, das nicht gelöscht werden darf.

user accounts Administratoren können ebenfalls alle Benutzerkonten der sselab.de-Instanz verwalten. Zusätzlich zu grundlegenden Funktionen wie das Erzeugen oder Löschen von Konten können auch mehrere Konten zusammengefasst werden. Dabei müssen alle überflüssigen Konten und ein Zielkonto identifiziert werden. Die überflüssigen Konten werden dann gelöscht und das Zielkonto zu allen Projekten in der richtigen Rolle hinzugefügt, denen diese Konten zugeordnet waren.

server mappings Die Verwaltung der Backend-Instanzen erfolgt über diese Seiten. Hier können neue Instanzen durch das Erstellen von Server-Mappings zu der sselab.de-Instanz hinzugefügt werden. Bereits registrierte Backend-Instanzen können hier ebenfalls durch das Löschen der zugehörigen Server-Mappings entfernt werden.

projects Die Verwaltung aller Projekte der Instanz ist über diese Administrationsseiten möglich. Hier können sowohl neue Projekte angelegt als auch existierende konfiguriert werden. Dazu gehört unter anderem die Verwaltung aller Teilnehmer sowie der genutzten Services jedes Projekts.

templates Projekttemplates ermöglichen die Gruppierung mehrerer Services, die dann bei der Erzeugung von Projekten anstatt einzelner Services ausgewählt werden können. Über

diese Seiten können neue Templates angelegt und existierende konfiguriert oder gelöscht werden.

services Auf den Seiten der jeweiligen Service-Kategorie können die Services der sselab.de-Instanz verwaltet werden. Dazu gehört die Installation und Aktualisierung durch das Hochladen von Service-Plug-ins und deren Abhängigkeiten sowie die Deinstallation von Services und deren Aktivierung oder Deaktivierung innerhalb der sselab.de-Instanz. Ein Link zur Verwaltung von Social-Services fehlt in Abbildung 8.11, da diese erst nach dem Release der Version 1.0 in das Framework integriert worden sind. In der Web-Oberfläche des Entwicklungszweigs der Version 1.1 sind die entsprechenden Seiten enthalten.

information entries Über diese Seiten können mehrere Arten von Bekanntmachungen durch die Administratoren verwaltet werden. Dazu gehören interne Bekanntmachungen, die nach der Authentifizierung den Benutzern angezeigt werden. Beispielsweise kann die Ankündigung von Wartungsarbeiten darüber erfolgen. Weiterhin können öffentliche Bekanntmachungen angelegt werden, die auf den frei zugänglichen Seiten angezeigt werden, sowie Einträge im FAQ-Abschnitt für Benutzer erzeugt werden.

messages Über diese Seiten können die Nachrichten verwaltet werden, die von der Instanz oder von den Benutzern an die Administratoren geschickt worden sind. Auf Anfragen von Benutzern kann hier direkt geantwortet werden. Zusätzlich können hier auch Nachrichten an verschiedene Nutzergruppen verfasst und gesendet werden. Beispielsweise an andere Administratoren, an die Benutzer bestimmter Projekte oder an einzelne Benutzer.

search Auf dieser Seite können Administratoren nach Benutzern, Projekten oder Services suchen. In den Resultaten der Suche sind dann die zugehörigen Profil-, Projekt- oder Serviceseiten verlinkt.

Neben diesen Funktionen können über die in Abbildung 8.11 gezeigte Startseite noch die zu Beginn diese Kapitels beschriebenen Statistiken angezeigt sowie die in Kapitel 7 behandelten Testaufrufe aller registrierten Backend-Instanzen ausgeführt und die Konsistenz der installierten Service-Plug-ins geprüft werden.

8.2.3 Nutzung der Services durch externe Werkzeuge

In diesem Abschnitt wird die Nutzung externer Werkzeuge in Kombination mit den Services der sselab.de-Instanz skizziert. Die Unterstützung externer Werkzeuge und existierender Clients basiert auf einigen Anforderungen an das SSELab-Framework, die in Kapitel 3 definiert worden sind. In Anforderung **FA24** wird beispielsweise gefordert, dass auch nach der Integration eines Werkzeugs in das Framework ein Zugriff durch existierende Clients möglich sein soll. Anforderung **NFA3** fordert weiterhin, dass die genutzten Authentifizierungs- und Autorisierungsmechanismen auf Standards basieren sollen, so dass ein authentifizierter Zugriff durch externe Werkzeuge und Clients ohne Weiteres möglich ist. Anforderung **FA25** beschreibt außerdem die Nutzung externer Werkzeuge in Kombination mit den Services einer SSELab-Instanz. Auf Grundlage dieser Anforderungen wurde sowohl das Framework als auch die Integration der

Services so umgesetzt, dass eine Nutzung durch externe Werkzeuge und existierende Clients unterstützt wird.

Durch den Einsatz standardisierter Authentifizierungs- und Autorisierungsverfahren können die Base-Services einer SSELab-Instanz mit existierenden Clients und externen Werkzeugen genutzt werden. Falls eine Authentifizierung vor der Verwendung eines Services notwendig ist, muss der Name und das zugehörige Passwort eines gültigen und autorisierten Benutzerkontos angegeben werden. Darüber hinaus fordert Anforderung **FA13** die Unterstützung eines automatisierten Zugriffs auf Services durch Bot-Accounts. Diese Accounts können im Kontext eines Projekts durch dessen Manager erzeugt und ebenfalls für die in Abschnitt 4.4.3 beschriebene Werkzeug-Service-Interaktion genutzt werden. Innerhalb eines existierenden Clients oder eines externen Werkzeugs kann dann ein Bot-Account genau wie ein Benutzerkonto verwendet werden.

Insgesamt können dadurch externe Werkzeuge in Kombination mit den Base-Services der sselab.de-Instanz effektiv genutzt werden, wie im Rahmen dieser Arbeit mehrfach erfolgreich gezeigt werden konnte. Beispiele dafür sind die Nutzung von Subversion-Repositories der Instanz durch verfügbare Subversion-Clients oder der Import der RSS-Feeds von Trac und Jenkins in gängige Feedreader. Weitere Beispiele sind die Verwendung von Trac mit Hilfe von Mylyn aus der Eclipse IDE heraus und der Einsatz von Continuous Integration Servern zur Überwachung von Subversion- oder Git-Repositories. Bei der Kopplung externer Werkzeuge mit den Base-Services der sselab.de-Instanz konnte jeweils die Konfiguration anhand der verfügbaren Dokumentation des jeweiligen Werkzeugs erfolgreich durchgeführt werden. Dementsprechend wird hier für die Methodik zur Kopplung externer Werkzeuge auf deren Dokumentation verwiesen.

Im Gegensatz zu Base-Services werden Ostp-Services üblicherweise nicht über eine Serverinfrastruktur angeboten, so dass für deren Kopplung mit externen Werkzeugen im Rahmen dieser Arbeit die in Abschnitt 4.5 beschrieben Framework-Clients entwickelt worden sind. Diese können genutzt werden, um die Ostp-Services der sselab.de-Instanz durch ein desktop- oder serverbasiertes Werkzeug aufzurufen. Ein Beispiel dafür ist die in Kapitel 6 beschriebene Integration von MontiCore in das Framework, durch die ein Aufruf des MontiCore-Generators aus der Eclipse IDE und von einer Kommandozeile aus möglich ist. Bei dem Aufruf eines Ostp-Services kann ebenfalls entweder ein gültiger Benutzername und das zugehörige Passwort oder ein Bot-Account wie in Abschnitt 4.4.5 genutzt werden.

Für Social-Services wurden im Rahmen dieser Arbeit keine speziellen Clients entwickelt, da einerseits deren Nutzung über einen Web-Browser effizient möglich ist und andererseits diese im Kontext von Benutzerprofilen zur Verarbeitung von Informationen sozialer Netzwerke genutzt werden. Eine Kopplung externer Werkzeuge mit diesen Services ist daher nicht sinnvoll und wird dementsprechend in der aktuellen Fassung des SSELab-Frameworks nicht unterstützt.

8.3 Betriebskonzept der Instanz

Im vorigen Kapitel 7 wurden die Verteilungsvarianten der Framework-Komponenten sowie die möglichen Betriebsarten von Instanzen des Frameworks behandelt. Die konkrete Verteilungsvariante und Betriebsart, die für die Bildung der sselab.de-Instanz ausgewählt worden sind, werden in diesem Abschnitt beschrieben. Abbildung 8.12 zeigt dazu in einem Verteilungsdiagramm die gesamte Serverinfrastruktur der Instanz sowie die Verteilung der Komponenten auf die einzelnen

Server. Alle gezeigten Server sind virtuelle Maschinen auf einem VMWare ESXi-Cluster mit Ubuntu-Server als Betriebssystem. Die Frontend-Instanz wird auf dem Server *lab10* betrieben. Zusätzlich zu der benötigten Laufzeitumgebung des Frontends gemäß Abschnitt 7.2.2 ist dort auch die Datenbank zur Verwaltung der persistenten Daten installiert.

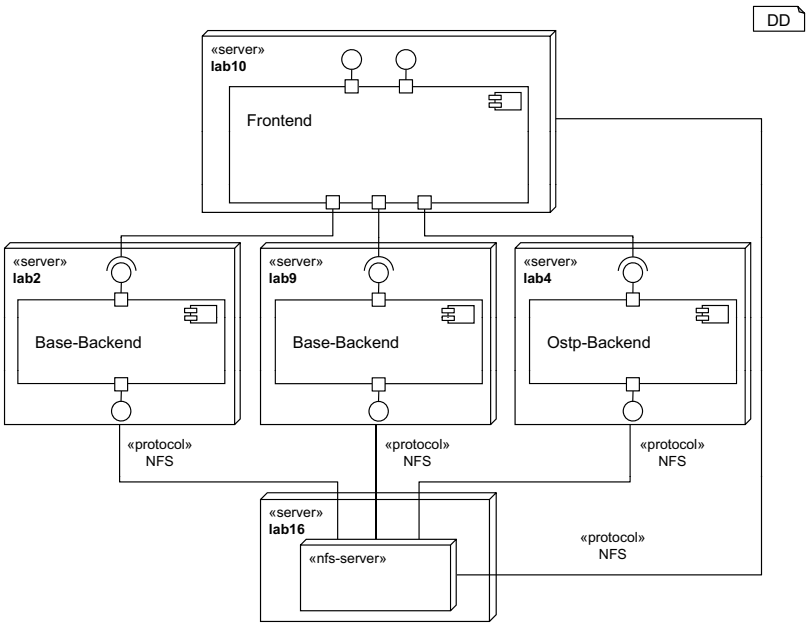


Abbildung 8.12: Verteilung der Komponenten der sselab.de-Instanz

Auf den beiden Servern *lab2* und *lab9* ist jeweils eine Base-Backend-Instanz mit der erforderlichen Laufzeitumgebung gemäß Abschnitt 7.2.1 installiert. Zusätzlich sind dort die Laufzeitumgebungen aller über die Instanz angebotenen Services vorhanden und konfiguriert. Der Server *lab4* wird für die Bereitstellung einer Ostp-Backend-Instanz mit allen benötigten Laufzeitkomponenten genutzt.

Der Server *lab16* wird als Storage-System für alle anderen Server verwendet, so dass alle Daten von Projekten zentrale darauf gespeichert werden können. Dazu gehören unter anderem die Daten der Services der Backend-Server – beispielsweise Subversion-Repositories oder Trac-Instanzen mit ihren Datenbanken – sowie die Datenbank der Frontend-Instanz. Die Kommunikation mit diesem Server erfolgt über das Network File System (NFS) Protokoll [SCR+03], so dass eine transparente Nutzung des Storage-Systems über das Dateisystem der anderen Server möglich ist. Die auf `lab16` gespeicherten Daten werden darüber hinaus über einen Backup-Mechanismus in regelmäßigen Abständen gesichert.

Neben dem Lehrstuhl für Software Engineering der RWTH Aachen und dessen Projektpartner nutzen noch weitere Lehrstühle die sselab.de-Instanz für die Durchführung von Projekten und

Lehrveranstaltungen. Die anfallenden Daten aller Projekte werden dabei auf die zuvor beschriebene Serverinfrastruktur des Lehrstuhls für Software Engineering gespeichert. Insgesamt ergibt sich dadurch, dass die Instanz gemäß Abschnitt 7.1 als Instanz für mehrere Kunden mit einem gemeinsamen Hosting der Daten und der Administrationsoberfläche betrieben wird. Die Wahl dieser Betriebsart ist möglich, da einerseits die in Abschnitt 7.1 erwähnte Vertrauensbasis durch die Beziehungen der Lehrstühle und deren Projektpartner gegeben ist. Außerdem ist diese Betriebsart in diesem Fall auch sinnvoll, da so der Administrationsaufwand des Lehrstuhls für den Betrieb der Instanz minimiert werden kann.

8.4 Zusammenfassung

In diesem Kapitel wurde ein Überblick über die Instanz des SSELab-Frameworks gegeben, die im Rahmen dieser Arbeit unter der URL `http://sselab.de` gebildet worden ist. Zunächst wurde anhand einiger Statistiken und der Beschreibung repräsentativer Projekte die Nutzung und der primäre Einsatzzweck der Instanz vorgestellt. Anschließend wurde eine Einführung in das Nutzungskonzept der Instanz für Gäste, Benutzer und Administratoren anhand der Web-Oberfläche des Frontends gegeben sowie im Anschluss daran auf die Nutzung der Services durch externe Werkzeuge eingegangen. Zuletzt wurden auf Grundlage von Kapitel 7 die für diese Instanz gewählte Verteilungsvariante und die Betriebsart beschrieben.

Kapitel 9

Verwandte Arbeiten

Mittlerweile existieren einige populäre webbasierte Projektportale, wie beispielsweise SourceForge, GitHub, Google Code, Assembla und JavaForge. Da die meisten dieser Portale kommerziell sind, gibt es üblicherweise nur wenige Informationen über deren Architektur, so dass ein Vergleich mit dem SSELab-Framework auf dieser Ebene nur schwer durchgeführt werden kann. In den folgenden Abschnitten werden daher die wichtigsten CDEs und Projektportale diskutiert, über die ausreichend Informationen in Form von wissenschaftlichen Beiträgen und technischer Dokumentation verfügbar sind. Dabei bilden die Diskussion aktueller Trends und Herausforderungen aus Kapitel 1 und die Anforderungen aus Abschnitt 3.3.4 die Grundlage für den Vergleich mit den Konzepten des SSELab-Frameworks.

9.1 SourceForge, GForge, FusionForge und Davenport

SourceForge [SF12] war eines der ersten Projektportale und wurde im Jahr 1999 von der Firma VA Linux Systems (später umbenannt in VA Software und schließlich in SourceForge, Inc.) entwickelt und unter `http://sourceforge.net/` in Betrieb genommen. Noch heute ist das Projektportal unter dieser Adresse verfügbar und bietet Open Source Projekten einen kostenlosen Hosting-Service an.

Die SourceForge Plattform wurde auf Grundlage einer Analyse der von Open Source Projekten verwendeten Methoden und Werkzeuge entworfen [ABS02]. Die Ziele bei der Entwicklung der Plattform waren unter anderem, die Erzeugung und die Administration verteilter Projekte und der genutzten Werkzeuge zu erleichtern, die Kommunikation und Kollaboration innerhalb von Projekten zu fördern und Wissen effektiv zu verwalten und es auch projektübergreifend nutzbar zu machen [ABS02]. Zur Erreichung dieser Ziele wurden mehrere webbasierte Werkzeuge in die Plattform integriert – sowohl Eigenentwicklungen als auch existierende Werkzeuge wie CVS und Subversion. Im Laufe der Zeit wurde SourceForge weiter in Richtung eines Framework-Ansatzes ausgebaut, indem sogenannte „Hosted Apps“ – freie oder Open Source Anwendungen – den Benutzern zur automatischen Installation angeboten wurden [SF12, KK09, CW09]. Die Hosted Apps sind daher vergleichbar mit den Services des SSELab-Frameworks. Im Gegensatz zur SourceForge Plattform wurde beim SSELab jedoch von Anfang an die Architektur auf die Integration existierender Werkzeuge ausgelegt.

Gemäß einem Interview mit Verantwortlichen von SourceForge wird die Plattform hauptsächlich für das Durchführen von Releases, die Nutzung von Versionsverwaltungssystemen und die Verwaltung von Mailing-Listen genutzt [CW09]. Darüber hinaus werden auch Werkzeuge zum

Projektmanagement wie beispielsweise Bug-Tracker eingesetzt. Die Plattform gibt insgesamt keinen spezifischen Entwicklungsprozess vor [CW09], unterstützt jedoch agile Prozesse, die in vielen Open Source Projekten genutzt werden [ABS02]. Da Open Source Projekte üblicherweise auch global verteilt durchgeführt werden, ist dieser Ansatz vergleichbar mit dem der sselab.de-Instanz, die ebenfalls keinen konkreten Prozess vorgibt, jedoch durch die integrierten Werkzeuge auf agile und verteilte Projekte zugeschnitten ist.

Nachdem zunächst der Quellcode der SourceForge Plattform frei zugänglich war, wurde im Jahr 2001 SourceForge.net Enterprise Edition als kommerzielle Version angeboten [ABS02] und gleichzeitig der Code durch eine Lizenzänderung geschlossen. Dies führte dazu, dass die letzte Open Source Version des SourceForge Codes unabhängig von VA Software weiterentwickelt und schließlich unter dem Namen GForge unter `http://gforge.org` durch das Unternehmen GForge Group verfügbar gemacht wurde [Lan09, GF12]. Neben der Open Source Variante wurde nach einiger Zeit ebenfalls eine kommerzielle Version unter dem Namen GForge Advanced Server angeboten. Als sich das Unternehmen zunehmend auf die kommerzielle Version konzentrierte, wurde im Jahr 2009 die frei verfügbare Variante von GForge von Open Source Entwicklern erneut umbenannt und unter `http://fusionforge.org/` weiterentwickelt. Die GForge Group hat darauf die kommerzielle Version von Grund auf neu entwickelt, jedoch den Quellcode seitdem nicht mehr frei zugänglich gemacht.

Die frei verfügbaren Varianten des Quellcodes von SourceForge bzw. GForge wurden von vielen Organisationen im industriellen und akademischen Umfeld verwendet, um eigene Projektportal-Instanzen zu installieren und zu nutzen [Lan09]. Im akademischen Bereich wurde GForge auch als Grundlage für die Entwicklung und die Erforschung von Projektportalen genutzt. In [KPW04] wird beispielsweise eine Erweiterung von GForge mit dem Namen Davenport vorgestellt. Die Autoren haben GForge mit dem Ziel erweitert, die diversen Protokolle (SSH/SFTP, FTP, CVS pserver, etc.) durch WebDAV/HTTP zu ersetzen. Außerdem haben sie einige andere Erweiterungen vorgenommen sowie das integrierte CVS durch Subversion ersetzt. Auf Grundlage ihrer Erfahrungen mit GForge bzw. Davenport fordern sie, dass CDEs eine erweiterbare („fully pluggable“) Architektur haben sollten, damit neue Entwicklungswerkzeuge einfach integriert werden können. Dies war jedoch keine Eigenschaft der Architektur von GForge. Bei der Nutzung und Weiterentwicklung von SourceForge bzw. GForge im akademischen Umfeld wurde darüber hinaus auch auf die schlechte Qualität und Modularität des Codes und der Dokumentation hingewiesen [COPT02, HLLL04]. Auch einer der ursprünglichen Entwickler von SourceForge, der später das Unternehmen GForge Group gegründet hat, bestätigt die schlechte Qualität der Codebasis, bis schließlich GForge Advanced Server von Grund auf neu entwickelt worden ist [Per13]. Bei der Entwicklung des SSELab-Frameworks wurde daher nicht auf GForge aufgesetzt, sondern die Architektur und der Code von Grund auf neu entwickelt und auf die Konzeption einer erweiterbaren Framework-Architektur geachtet.

9.2 Jazz

Jazz [Jaz12] ist eine kommerzielle CDE bzw. eine erweiterbare Technologieplattform von IBM und basiert unter anderem auf den Erfahrungen von IBM mit Eclipse, WebSphere und anderen Open Source Projekten [Fro07, Ric10]. Der Fokus von Jazz ist die nahtlose Integration von

Entwicklungs- und Kollaborationswerkzeugen unter Berücksichtigung der eingesetzten Entwicklungsprozesse. Jazz wird von IBM schon seit ungefähr zehn Jahren entwickelt [Fro07]. Im Laufe dieser Zeit haben sich die Architektur und der Funktionsumfang von Jazz mehrfach grundlegend geändert. In den folgenden Abschnitten werden die wichtigsten Meilensteine diskutiert.

Jazz als Forschungsprototyp

In den Jahren 2003 und 2004 haben Mitarbeiter von IBM Research die ersten wissenschaftlichen Beiträge zum Thema Jazz veröffentlicht [CHRP03, CdSH+03, CHR+03, HCRP04]. Darin wurden die ursprünglichen Ziele und erste Ergebnisse des Jazz-Forschungsprojekts beschrieben. Jazz wurde damals auf Grundlage der Vision von Booch und Brown für CDEs [BB03] und des Konzepts der kontextuellen Kollaboration [Fon03] entwickelt. Dabei werden Kollaborationsmechanismen in eine Basisanwendung integriert [HCRP04]. Im Fall von Jazz wurde Eclipse als Basisanwendung verwendet und entsprechend um mehrere Kollaborationswerkzeuge ergänzt. Das Ziel bei der Integration dieser Werkzeuge war, die Produktivität kleiner informeller Entwicklerteams zu verbessern [CHRP03, HCRP04].

In den Veröffentlichungen bezüglich des Forschungsprototyps von Jazz werden nicht die Architektur, sondern ausschließlich die Erweiterungen von Eclipse beschrieben. Die wichtigsten Funktionen, die von den Autoren realisiert worden sind, waren synchrone und asynchrone Kommunikationsmechanismen über eine Chat-Funktion und ein Diskussionsoberfläche sowie eine Screen-Sharing-Anwendung [CHRP03]. Diese Funktionen konnten aus Eclipse heraus über eine View aufgerufen werden, mit deren Hilfe Teams in Jazz verwaltet werden können. Zusätzlich haben die Autoren Eclipse so erweitert, dass für die Mitglieder eines Teams ersichtlich ist, welche Dateien gerade von anderen Mitgliedern des Teams bearbeitet werden [CHRP03].

Der erste Forschungsprototyp von Jazz basiert auf einer engen Integration von Kollaborationswerkzeugen in die IDE [CdSH+03]. Für die beschriebenen Chat- und Screen-Sharing-Funktionen wurden auch entsprechende Serverkomponenten in Jazz benötigt, die jedoch in den ersten Publikationen nicht weiter diskutiert werden. Im Gegensatz zu diesem ersten Ansatz von Jazz basiert das SSELab-Framework auf einer anderen Vision. In das Framework werden alle Werkzeuge, also auch Kollaborationswerkzeuge, zunächst als Hosted Services auf der Serverseite integriert. Eine Integration in die IDE erfolgt gegebenenfalls dann erst in einem nachfolgenden Schritt. Darüber hinaus ist die Integration von Werkzeugen in das SSELab-Framework eher leichtgewichtig. Obwohl laut den Autoren Jazz gemäß der Vision von [BB03] für CDEs entwickelt worden ist, widerspricht der Ansatz von Jazz den Aussagen von Booch und Brown, die behaupten, dass aus einer IDE keine CDE werde, wenn einzelne Kollaborationswerkzeuge integriert würden.

Jazz-Plattform Version 0.6

Im Jahr 2005 wurde das Jazz-Forschungsprojekt zu einem offiziellen Projekt von IBM [Ric10, Jaz12]. Die neue Vision für das Projekt war umfangreicher als die in [CdSH+03, CHRP03, CHR+03, HCRP04] beschriebene Vision [Fro07]. Dies spiegelt sich auch in der Zusammensetzung der Projektmitglieder wider, die nicht mehr nur von IBM Research, sondern auch aus dem IBM Eclipse-Team und von Rational Software kamen [Jaz12]. Im Jahr 2008 wurde schließlich die Jazz-Plattform in der Version 0.6 und Rational Team Concert in der Version 1.0 als Client für

die Plattform veröffentlicht [Jaz12]. In [JPTO12] wird diese Version von Jazz als eine erweiterbare Kollaborationsplattform für Teams beschrieben, die eine nahtlose Integration von Aufgaben bzw. Tätigkeiten in allen Phasen des Software-Lebenszyklus erlaubt. Jazz war nun nicht mehr nur auf kleine Teams ausgerichtet, sondern skalierte auch für den Einsatz in großen Teams [JPTO12].

Diese Version der Jazz-Plattform basiert auf einer Client-Server-Architektur [JPTO12]. Der Server ist eine Java-Web-Anwendung, die in einem Java EE 1.4 kompatiblen Application Server läuft, auf einer OSGi-Laufzeitumgebung basiert und eine Menge von Services bereitstellt, die über SOAP und HTTP aufgerufen werden können. Für den Zugriff auf den Server können unterschiedliche Clients genutzt werden: entweder Eclipse, Kommandozeilenwerkzeuge oder Ant-Skripte sowie Web-Browser für die Web-Oberfläche des Servers. Sowohl der Server als auch die Java-basierten Clients bauen auf Eclipse-Technologien auf und verwenden dementsprechend Plug-in-Systeme [JPTO12]. Insgesamt ist die Grobarchitektur von Jazz ähnlich zu der des SSELab-Frameworks, das ebenfalls mehrere Arten von Clients unterstützt und Plug-in-Systeme auf der Serverseite einsetzt. Unterschiede ergeben sich jedoch bei den jeweiligen Service-Konzepten. In Jazz werden Werkzeuge durch sogenannte Jazz-Komponenten bereitgestellt [JPTO12]. Jede Komponente besteht aus mehreren Plug-ins, die teilweise auf dem Server und teilweise in Eclipse installiert werden. Die serverseitigen Plug-ins definieren einen Web-Service über ein Java-Interface und ein zugehöriges clientseitiges Plug-in kapselt den Zugriff auf diesen Service [Ric10]. Im SSELab-Framework werden Services immer im Kontext eines Backends bereitgestellt. Jedes Backend definiert einen einzelnen Web-Service, über den alle Services des Backends aufgerufen werden können. Dadurch können durch das Framework generische Clients für alle Services einer Backend-Arten angeboten werden, so dass der Aufwand für die Integration eines neuen Werkzeugs verringert wird, da zunächst nur das Service-Plug-in realisiert werden muss. In einem nachgelagerten Schritt können dann bei Bedarf servicespezifische Clients auf Grundlage der generischen Clients entwickelt werden.

Die Jazz-Plattform in der Version 0.6 besteht aus den zwei Kernkomponenten *Repository* und *Team Process*, die den sogenannten Kernel bilden, und aus mehreren optionalen Komponenten [JPTO12]:

- Die Komponente *Repository* basiert auf einer relationalen Datenbank, deren Tabellen und Schemata zur Laufzeit durch andere Komponenten erweitert werden können. Jede Jazz-Komponente kann dazu mit Hilfe von Java-Klassen und eines Ecore Modells des Eclipse Modeling Frameworks (EMF) [SBPM08] das benötigte Datenmodell deklarieren. Der Server erzeugt auf Grundlage dieser Modelle die benötigten relationalen Tabellen und die zugehörigen SQL-Anfragen [Ric10]. Das SSELab-Framework besitzt momentan keinen solchen Mechanismus. Bei der Erzeugung einer neuen Backend-Art muss dass Frontend mit den zugehörigen Datenbanktabellen manuell erweitert werden. Ein entsprechender Mechanismus wäre für das Framework wünschenswert, ist jedoch nicht essentiell, da erfahrungsgemäß neue Backend-Arten nur selten erzeugt werden [HKR12b].

- Die Komponente *Team Process* ist die zweite Kernkomponente von Jazz. Durch diese Komponente wird die Prozessunterstützung von Jazz realisiert. Ein Prozess wird in diesem Kontext als die Menge der Methoden, Regeln, Richtlinien und Konventionen eines Teams definiert [JPTO12]. Der Prozess eines Teams, z.B. ein agiler oder ein RUP-basierter Pro-

zess, kann mit Hilfe dieser Komponente regelbasierte definiert werden [YEC+08]. Andere Komponenten können dann auf den definierten Prozess in der Komponente Team Process zugreifen und das Team entsprechend unterstützen. Im Kern des SSELab-Frameworks wurde keine Prozessunterstützung integriert, da der Kern unabhängig von einem konkreten Prozess bleiben soll und stattdessen die integrierten Werkzeuge für die Prozessunterstützung zuständig sein sollen.

- Zusätzlich zu den beiden Kernkomponenten enthält die Jazz-Plattform noch die folgenden optionalen Komponenten. *Work Item* ist die Jazz-Komponente für die Verarbeitung von Tickets, Bugs und Aufgaben in Jazz. *Source Control* realisiert ein Versionsverwaltungssystem. *Team Build* integriert ein Build-Server. *Team Reports* stellt eine Reportsystem und *Agile Planning* leichtgewichtiges Planungs-Wiki zur Verfügung. In das SSELab-Framework wurden vergleichbare Werkzeuge integriert. Dabei wurden jedoch keine Eigenentwicklungen, sondern existierende Werkzeuge verwendet.

Die Jazz-Plattform wurde von IBM ab 2006 für die Entwicklung von Jazz selbst genutzt [Fro07]. Sie wurde jedoch nicht nur von IBM, sondern auch in verschiedenen Forschungsprojekten eingesetzt, weiterentwickelt oder analysiert. Beispielsweise hat IBM die Inhalte des Repositories dieser Instanz einigen ausgewählten Forschern zur Verfügung gestellt [YEC+08]. Auf Grundlage der Repository-Daten konnten viele Eigenschaften des Entwicklungsprozesses von Jazz ausgewertet werden. Das Repository wurde dementsprechend von einigen Forschern analysiert [NSD08, HZ09, FCP11]. Ein weiteres Beispiel sind Arbeiten, um Informationen aus sozialen Netzwerken in Jazz zu integrieren. In [CGL09] beschreiben die Autoren, wie sie den Mashup-Service [YBCD08] FriendFeed in Jazz integriert haben. Mit Hilfe des FriendFeed-Services können Feeds verschiedener sozialer Netzwerke aggregiert werden. Durch die Integration in Jazz können die von den Projektmitgliedern täglich genutzten Dienste von sozialen Netzwerken dazu beitragen, dass persönliche Informationen ausgetauscht und diskutiert werden können, wodurch die Entwicklung von Beziehungen und der Aufbau eines gemeinsamen Kontextes ermöglicht wird [CGL09]. Ein weiteres Beispiel ist die Integration eines Werkzeugs für verteiltes Pair Programming in Jazz [DMH+08].

In der Literatur werden neben den positiven Aspekten, wie der guten Integration und der reichhaltigen Informationen im Repository, auch einige Einschränkungen von Jazz diskutiert. Jazz basiert beispielsweise auf einer bestimmten Vision der Kollaboration, die gegebenenfalls nicht zu den vorherrschenden Prozessen eines Teams passt. In diesem Fall kann die Nutzung von Jazz für ein Entwicklerteam einschränkend sein [Fro07]. Eine andere Einschränkung ergibt sich durch die enge Kopplung aller Werkzeuge. Dies produziert eine Herstellerabhängigkeit [Fro07], die ein großes Risiko bei der Wahl einer Integrationsplattform wie Jazz darstellt [PVEP10]. Auch das zentrale Repository hat nicht nur Vorteile. Gemäß [HZ09] ist die Datenbank von Jazz sehr komplex und die Datenmenge enorm, aber das Schema nur wenig dokumentiert [CCD+08].

Jazz-Integrationsarchitektur

In der Version 0.6 war die Jazz-Plattform hauptsächlich darauf ausgerichtet, dass alle Werkzeuge für die Plattform von Grund auf neu geschrieben werden [Ric10]. Diese Ausrichtung stellt einen

wesentlichen Unterschied zum SSELab-Framework dar, das von Beginn an auf die Integration existierender Werkzeuge ausgerichtet war und nicht deren Konformität zum Framework fordert. Der Vorteil von Eigenentwicklungen ist, dass ein hoher Grad der Integration erreicht werden kann. Ein entscheidender Nachteil ist jedoch der große Aufwand für die Entwicklung und Wartung der integrierten Werkzeuge [KK09] (vgl. auch Abschnitt 1.3).

Letztendlich führte dieser Nachteil, zusammen mit den zuvor beschriebenen Einschränkungen, zu einer Neuausrichtung von Jazz. Nach dem Release der Jazz-Plattform in der Version 0.6 wurde Telelogic von IBM im Jahr 2008 aufgekauft [Jaz12]. Unter anderem durch diese Übernahme mussten viele neue Werkzeuge in Jazz integriert werden. Die Entwickler von Jazz realisierten zu diesem Zeitpunkt, dass die Zahl der existierenden und zu integrierenden Werkzeuge immer die Zahl der Eigenentwicklungen bei Weitem übersteigen würde [Ric10]. Die Architektur von Jazz war zu diesem Zeitpunkt jedoch nicht auf die Integration existierender Werkzeuge ausgelegt. Insbesondere das Repository war nicht skalierbar genug, um alle Bedürfnisse der zu integrierenden Werkzeuge effizient erfüllen zu können [Ric10]. Durch die Neuausrichtung auf die Integration existierender Werkzeuge musste die Architektur von Jazz entsprechend angepasst werden [Ric10].

Um auch Werkzeuge externer Anbieter integrieren zu können, startete IBM im Jahr 2009 das Projekt „Open Services for Lifecycle Collaboration“ (OSLC) [OSLC12], um eine offene Spezifikation für die Integration von Werkzeugen zu definieren. Die wichtigsten Aspekte dieser Spezifikation wurden an die Architektur des Webs und dem Konzept „Linked Data“ angelehnt [ZW11]: Jede Ressource eines Werkzeugs – beispielsweise eine Anforderung, ein Testfall oder eine Quellcodedatei – wird durch eine eindeutige und langlebige URL identifiziert, über die auf die Ressource zu gegriffen werden kann, unabhängig davon durch welches Werkzeug sie erzeugt wurde. Das Format von Ressourcen wird zusätzlich über XML definiert. Die Formatbeschreibung enthält dabei jedoch nicht alle Details, sondern nur grundlegende gemeinsame Elemente und kann durch werkzeugspezifische Elemente erweitert werden. Zusätzlich stellen Werkzeuge sogenannte Services, wie beispielsweise Queries, bereit, um Ressourcen verarbeiten zu können.

In der Jazz-Plattform wurde die relationale Datenbank durch einen dateibasierten RDF-Speicher und SPARQL-Provider ausgetauscht. Die Integration erfolgt primär zwischen webbasierten Werkzeugen, kann aber auch Clients unterstützen [Ric10]. Im Jahr 2009 wurden die ersten Werkzeuge auf Grundlage der neuen Architektur veröffentlicht [Ric10]. Da das SSELab-Framework auf einer leichtgewichtigen Integration von Werkzeugen basiert und vorhandene Integrationsmechanismen berücksichtigt, könnten auch Werkzeuge integriert werden, die der OSLC-Spezifikation entsprechen.

9.3 Projektserver Proanvil

Die Autoren von [KK09] diskutieren den Einsatz von Collaborative Development Environments (CDEs) in Universitäten und wie dadurch sowohl Software-Entwicklungsprojekte im Allgemeinen, insbesondere aber die Ausbildung von Studierenden und die Arbeit von Universitätsangestellten verbessert werden kann. Sie betrachten dabei nicht nur die technischen Aspekte von CDEs, wie die Integration von Werkzeugen, sondern auch soziale Aspekte, wie beispielsweise die Gründe für die Akzeptanz oder die Ablehnung eines Projektportals durch die Benutzer.

Zusätzlich betonen sie den Einsatz von Social Software unter anderem zur Teambildung innerhalb von Projekten, zur Verbesserung der Motivation von Studierenden und der Akzeptanz der Benutzer.

Nach Aussage der Autoren wurden die zukünftigen Benutzer vom *Projektserver Proanvil* [PP12] bei der Entwicklung von Beginn an involviert. Zuerst wurde die aktuelle Situation am Lehrstuhl analysiert, danach die Anforderungen an das Projektportal definiert, dann existierende Systeme evaluiert und schließlich ein Prototyp des Systems entwickelt. Das Portal wurde von Beginn an zur Unterstützung von Projekten genutzt und auf Grundlage der gewonnenen Erfahrungen weiterentwickelt. Bei der Entwicklung des SSELab-Frameworks wurde ein ähnlicher Ansatz gewählt. Nach der Analyse der Anforderungen und der Entwicklung der ersten Version des Systems wurden unmittelbar reale Projekte mit Hilfe einer SSELab-Instanz durchgeführt und dann die Funktionen auf Grundlage des Benutzer-Feedbacks angepasst.

Die Autoren diskutieren drei Ansätze, um ein Projektportal zu erstellen. Der erste Ansatz, ein existierendes System zu erweitern, wurde verworfen, da zu der Zeit der Evaluation kein System existierte, das die beschriebenen Anforderungen abgedeckt oder mit angemessenem Aufwand hätte angepasst werden können. Der zweite Ansatz, bei dem das System vollständig selbst entwickelt wird, wurde ebenfalls nicht gewählt, da eine Eigenentwicklung (inklusive aller Werkzeuge) durch die Menge und die Komplexität der einzelnen Komponenten viel Zeit in Anspruch genommen hätte und das System nicht in einem angemessenen Zeitraum hätte fertiggestellt werden können. Außerdem wäre die Wartung und Erweiterung der Komponenten für den Lehrstuhl aus Kapazitätsgründen nicht möglich. Der dritte Ansatz, bei dem ein Framework für die Integration existierenden Komponenten entwickelt wird, wurde von den Autoren gewählt. Die Gründe dafür sind, dass viele frei verfügbare Anwendungen existieren, diese unabhängig voneinander weiterentwickelt werden und leicht durch den Framework-Ansatz geändert, hinzugefügt oder entfernt werden könnten und weil viele Benutzer schon mit diesen Anwendungen vertraut sind. Darüber hinaus diskutieren die Autoren auch einige Herausforderungen in Bezug auf den Framework-Ansatz. Einerseits müssen übergeordnete Funktionen entwickelt werden können, wie beispielsweise Single-Sign-On, Benachrichtigungssysteme oder Suche über den Inhalt der Komponenten. Außerdem müssen laut den Autoren Synergieeffekte bei der Integration der Anwendungen erzielt werden, um ein Mehrwert durch die integrierten Anwendungen zu erzielen. Das SSELab wurde ebenfalls als Framework entwickelt, in das existierenden Anwendungen integriert werden können. Die Gründe für die Wahl des Framework-Ansatzes sind die gleichen, die auch die Autoren von [KK09] angeben. Einige Unterschiede zwischen den Ansätzen ergeben sich jedoch bei der Architektur: Proanvil kann im Vergleich zum SSELab-Framework nicht über mehrere Server verteilt, sondern nur auf einem einzelnen virtuellen Server betrieben werden. Außerdem basiert Proanvil nicht auf einem Plug-in-Ansatz wie das SSELab-Framework.

Die Autoren vergleichen ihre Arbeit auch mit DrProject [RW07] (vgl. Abschnitt 9.4) und sagen, dass ihr System ähnliche Anforderungen erfüllt, auch für den universitären Einsatz gedacht ist und entsprechende Funktionalität besitzt. Sie grenzen sich jedoch durch die Betonung der sozialen Aspekte und durch den Framework-Ansatz bei der Entwicklung des Systems von [RW07] ab. Darüber hinaus erwähnen sie bei der Diskussion verwandter Arbeiten, dass auch SourceForge [SF12] durch die Integration von „Hosted Apps“ einen Ansatz zur Integration von existierenden Anwendungen verfolgt.

In [KK11] beschreiben die Autoren die Ergebnisse von zwei Umfragen unter Studierenden, die auf deren Erfahrungen mit und Anforderungen an ein Projektportal abzielen. Darüber hinaus fließen in die Ergebnisse die Erfahrungen aus dem Einsatz des Systems für Projekte innerhalb von zwei Jahren ein. In den Umfragen wurden einerseits die Erfahrungen der Studierenden mit Werkzeugen und Projektportalen und andererseits technische Aspekte abgefragt, beispielsweise welche Anwendungen die Studierenden nutzen möchten. Zusätzlich wurde gefragt, welche Art der Administration von den Studierenden bevorzugt wird, ob soziale Funktionen genutzt werden und ob der Inhalt von anderen Projekten zugänglich sein sollen. Die Umfrage wurde mit Hilfe eines Online-Umfragewerkzeugs durchgeführt.

Die Ergebnisse der Umfragen und der Erfahrungen mit den laufenden Projekten am Lehrstuhl ergeben einen Trend in Richtung ausgereifter Werkzeugunterstützung für Projekte. Alle Studierende, die an der Umfrage teilgenommen haben, möchten Werkzeuge in ihren Projekten nutzen und insbesondere Versionsverwaltungssysteme wie Subversion und Git. Darüber hinaus werden Wikis, Bug-Tracking-Systeme und Mailing-Listen als wichtig angesehen. Weiterhin wollen die Studierenden selbst Projekt administrieren und die Werkzeuge für ihre Projekte auswählen können. Die Ergebnisse der Umfrage stimmen mit den Erfahrungen mit dem Einsatz der SSELab-Instanzen überein, so dass auch die meisten geforderten Funktionen und Werkzeuge durch die Instanzen abgedeckt werden. Nur im Bereich der sozialen Aspekte ist eine Abweichung vorhanden. Aus den Umfragen ergibt sich, dass viele Studierende für einen Zugang zu fremden Projekten gestimmt haben und deren Inhalte sehen und Projekte gegenseitig kommentieren wollen. Eine Designentscheidung des SSELab-Frameworks ist jedoch, dass nur dann Projekte für einen Benutzer sichtbar sind, wenn er selbst ein Teilnehmer davon ist, um die Mandantenfähigkeit gewährleisten zu können. Der Grund für diese Entscheidung war, dass auch Projekte mit sensiblen Daten durchgeführt werden und der Charakter eines offenen Portals das Vertrauen in das SSELab bezüglich des Datenschutzes beeinträchtigen könnte.

9.4 DrProject

In [RW07] wird das webbasierte Projektportal *DrProject* vorgestellt. Dieses Portal wurde speziell für den Einsatz in der Lehre an Universitäten entwickelt und an die Bedürfnisse von Studierenden und deren Betreuer angepasst. Laut den Autoren ist das Ziel des Systems, den Studierenden den Umgang mit Projektportalen und der Zusammenarbeit in Software-Entwicklungsprojekten beizubringen.

Die Autoren beschreiben in [RW07] die Entwicklung von DrProject. In 2003 wurden zunächst drei Prototypen des Systems entwickelt. Zwei der Prototypen waren Eigenentwicklungen, einmal auf Grundlage von Java Server Pages (JSPs) und einmal unter Verwendung von Hibernate, Tapestry und anderen, damals aktuellen Java-Technologien. Der dritte Prototyp wurde als Erweiterung von GForge umgesetzt. Alle drei Prototypen sind laut Aussage der Autoren gescheitert. Die Gründe dafür waren, dass der Einarbeitungsaufwand bei den Prototypen auf Java-Basis so groß war und das System nicht von Studierenden hätte weiterentwickelt und gewartet werden können. Der dritte Prototyp wurde nicht zu einem Produkt ausgebaut, da GForge nach Aussage der Autoren zu viele Sicherheitslücken aufgewiesen hat. In 2005 wurde schließlich Trac als Grundlage für DrProject ausgewählt und an die Bedürfnisse der Autoren angepasst. Im Gegensatz zu DrProject

wurde das SSELab-Framework auf Grundlage eines dritten Ansatzes entwickelt. Das SSELab ist keine vollständige Eigenentwicklung und auch keine Adaption eines existierenden Portals, sondern ein Framework für die Integration vorhandener Anwendungen.

Die Anforderungen aus [RW07] an DrProject überlappen mit den Anforderungen an eine SSELab-Instanz für Software-Entwicklungsprojekte gemäß Abschnitt 3.3. Für jedes Projekt in DrProject werden ein Subversion-Repository, ein Wiki, eine vereinfachte Version der Trac-Ticket-Engine und eine Mailing-Liste bereitgestellt. Zusätzlich wurde eine Administrationsoberfläche entwickelt, die es den Betreuern ermöglicht, viele gleichartige Projekte auf einmal aufzusetzen und die Rollen und Rechte der Benutzer feingranular zu vergeben. Bei der Entwicklung des SSELab-Frameworks wurde auf die Verwendung feingranularer Rechte und Rollen innerhalb eines Projekts verzichtet, da der Administrationsaufwand erfahrungsgemäß den Nutzen übersteigt. Außerdem verfolgt das SSELab-Framework im Gegensatz zu DrProject nicht den Ansatz, den Funktionsumfang der einzelnen Komponenten zu reduzieren, da die Komponenten einerseits möglichst wenig angepasst werden sollten und andererseits die Studierenden möglichst früh durch die Nutzung einer SSELab-Instanz an den Umgang mit komplexen Werkzeugen herangeführt werden sollen, damit sie im Laufe ihres Studiums sowohl den Nutzen als auch die Herausforderungen komplexer Werkzeuge kennenlernen können.

Zum Zeitpunkt der Erstellung dieser Ausarbeitung wird DrProject nicht mehr weiterentwickelt [DP12]. Zwar wurde zwischenzeitlich ein Nachfolgeprojekt mit dem Namen Basie gestartet, das aber mittlerweile auch eingestellt worden ist.

9.5 ProGET und PicoLibre

In [BBH06] beschreiben die Autoren eine Plattform zur Kollaboration mit dem Namen ProGET. Das Ziel dieser Plattform ist die Unterstützung von Forschern und ihrer externen Partner bei der Durchführung von Projekten sowie die Überwachung und Steuerung der Projekte durch die Forschungseinrichtung. Der Fokus liegt dabei nicht auf Software Engineering Projekten, so dass in die Plattform ausschließlich Kollaborations- und Groupware-Werkzeuge und keine Entwicklungswerkzeuge integriert worden sind. Beispiele für solche Werkzeuge sind phpGroupWare, eine Mailing-Listen-Software, TWiki und ein WebDAV-basierter Storage Service. ProGET wurde auf Grundlage der Erfahrungen von PicoLibre (später umbenannt in PicoForge) entwickelt [BBH06]. PicoLibre ist ebenfalls eine Plattform zur Kollaboration, die jedoch speziell auf Software Engineering Projekte ausgerichtet ist [COPT02]. Darin wurden beispielsweise Werkzeuge wie CVS, Mailing Listen und Bug Tracker integriert und Studierenden für die Durchführung ihrer Entwicklungsprojekte zur Verfügung gestellt.

Einige der in [BBH06] beschriebenen Integrationsprinzipien von ProGET wurden auch bei der Entwicklung des SSELab-Frameworks zugrunde gelegt. Beispielsweise beschreiben die Autoren, dass sie davon abgesehen haben, die integrierten Werkzeuge anzupassen, um fehlende Funktionalität zu ergänzen. Anstatt dessen haben sie mehrere Werkzeuge unverändert integriert, obwohl dadurch gegebenenfalls Redundanz entsteht. Ein weiteres Beispiel ist die Partitionierung der Daten aller integrierten Werkzeuge, so dass die Projekte unabhängig voneinander sind und der Datenschutz sichergestellt werden kann. Diese beiden Prinzipien waren auch bei der Entwicklung des SSELab-Frameworks wichtige Anforderungen (vgl. Abschnitt 3.3). Insgesamt gibt

es daher einige konzeptionelle Ähnlichkeiten zwischen ProGET und dem SSELab-Framework. Auch einige der in [BBH06] beschriebenen Werkzeuge waren eine Inspiration für die Umsetzung entsprechender Services. Beispielsweise die WebDAV-basierten Services Storage und Web Space. Ein wesentlicher Unterschied ist jedoch, dass sowohl PicoLibre als auch ProGET nicht als Framework entwickelt worden sind, sondern auf Projekte bestimmter Domänen abzielen. Dies wurde in [BBH06] als eine einschränkende Eigenschaft bezeichnet. Bei der Entwicklung des SSELab-Frameworks wurde daher aufbauend auf diesen Erfahrungen auf die Trennung der integrierten Werkzeuge und deren Verwaltung geachtet und das Framework so realisiert, dass es nicht auf konkrete Projektarten abzielt.

Kapitel 10

Zusammenfassung und Ausblick

In diesem Kapitel werden die wichtigsten Ergebnisse dieser Arbeit noch einmal zusammengefasst. Darauf aufbauend werden in einem Ausblick einige Forschungsthemen und Erweiterungen skizziert, die in weiterführenden Arbeiten behandelt werden könnten.

10.1 Zusammenfassung

In der vorliegenden Arbeit wurde der servicebasierte Einsatz von Werkzeugen in Software Engineering Projekten behandelt. Das primäre Ziel war, existierende Werkzeuge als Software as a Service in eine Serverumgebung zu integrieren, um deren flexible Nutzung in Entwicklungsprojekten zu ermöglichen. Zur Erreichung dieses Ziels wurden Konzepte für ein Framework für webbasierte Projektportale und CDEs mit dem Namen SSELab entwickelt. Dieses Framework ist auf die Bereitstellung von Werkzeugen als Services sowie einen Zugriff auf deren Funktionen über desktop- und webbasierte Clients ausgelegt. Das Design des Frameworks wurde maßgeblich durch mehrere Trends und Herausforderungen beeinflusst, die im Rahmen dieser Arbeit im Kontext von Entwicklungsumgebungen identifiziert worden sind. Neben dem Trend hin zu Hosted Services wurden dabei auch die Integration von IDEs und CDEs, der Einsatz von Web 2.0 Anwendungen und Social Software sowie die Verwendung von Plug-in-Systemen in Entwicklungsumgebungen berücksichtigt. Als wichtigste Herausforderungen wurden die große Menge sowohl an neuen als auch an Legacy-Werkzeugen, vorhandene Integrationsmechanismen sowie die Abhängigkeit von Funktionsumfang und Langlebigkeit einer Integrationslösung hervorgehoben.

Jedes Framework enthält eine Repräsentation der Anwendungsdomäne sowie geeignete Abstraktionen, um eine effektive Erweiterbarkeit zu ermöglichen. Der Kern des Frameworks enthält dementsprechend ein Domänenmodell für Projektportale und CDEs, durch das alle für diese Arbeit relevanten Aspekte abgedeckt werden. Das Finden optimaler Abstraktionen ist im Allgemeinen eine schwierige Aufgabe und erfolgt üblicherweise durch Generalisierung mehrerer konkreter Beispiele [JF88]. Parallel zu der Konzipierung und Realisierung des Frameworks wurden daher Werkzeuge unter Verwendung der Erweiterungsmechanismen als Services integriert. Abstrahierend von den konkreten Werkzeugen wurden dabei drei Service-Kategorien identifiziert: Base-Services für server- und webbasierte Anwendungen, Ostp-Services für dateiverarbeitende Werkzeuge und Social-Services für Funktionen sozialer Netzwerke. Auf Grundlage dieser Kategorien konnten im Rahmen der Realisierung des Frameworks bereits vielfältige Werkzeuge integriert werden. Dies spricht einerseits für geeignet gewählte Abstraktionen bei der Definition

der Kategorien und andererseits für den gewählten Ansatz einer leichtgewichtigen Integration, bei der die Werkzeuge möglichst wenig angepasst werden und vorhandene Integrationsmechanismen erhalten bleiben. Dieser Ansatz erfordert auch, dass die Werkzeuge so realisiert sind, dass deren Integration in ein Framework wie das SSELab effizient möglich ist. Insgesamt wurde im Laufe dieser Arbeit jedoch festgestellt, dass noch immer viele Werkzeuge entwickelt werden, ohne dass eine Integration in Plattformen oder Frameworks ausreichend berücksichtigt wird. Diese Werkzeuge müssen daher mit erhöhtem Aufwand für eine Integration vorbereitet werden. Aus diesen Erfahrungen wurden mehrere Anforderungen abgeleitet, durch die eine effizientere Integration von Werkzeugen ermöglicht werden kann.

Das Framework ist als verteiltes webbasiertes System mit mehreren Komponenten konzipiert: Die Backends sind für die Konfiguration und Ausführung der eigentlichen Services verantwortlich. Für jede Service-Kategorie existiert dabei eine eigenständige Backend-Art. Das Frontend ermöglicht die Verwaltung der Backends mit ihren Services sowie der Benutzer und Projekte. Die Clients des Frameworks nutzen die Schnittstellen des Frontends und ermöglichen den Aufruf der angebotenen Funktionalität sowie von Services über eine Kommandozeile oder aus einer IDE heraus. Die Konzepte zur Erweiterbarkeit des Frameworks basieren einerseits auf der Entkopplung der Verwaltungsmechanismen im Frontend von der Konfiguration und Ausführung der eigentlichen Services in den Backends sowie der Verwendung der Services über die Clients. Andererseits erlaubt die Architektur der Backends eine Installation von Services zur Laufzeit einer Framework-Instanz. Auf Grundlage dieser Architektur und der Konzepte des Frameworks wurden in dieser Arbeit Methodiken zur Erweiterung des Frameworks diskutiert. Neben der Integration von Services aller Kategorien wurden auch die Erweiterung des Frameworks zur Unterstützung neuer Service-Kategorien sowie die Entwicklung von Clients behandelt.

Die Instanziierung des Frameworks erfolgt durch ein Deployment der Komponenten in einer geeigneten Serverumgebung. Durch die flexiblen Verteilungsmöglichkeiten der Framework-Komponenten ergeben sich dabei verschiedene Betriebsarten, von denen eine im Zuge des Deployment-Prozesses ausgewählt werden muss. Diese Betriebsarten resultieren daraus, dass einerseits die Daten der Services und die Administrationsoberfläche des Frontends über mehrere Server-Maschinen verteilt und andererseits Framework-Instanzen für einzelne oder für mehrere Kunden betrieben werden können. Darüber hinaus ist es möglich, mit Hilfe von Instanzen eine a posteriori Integration von Werkzeugen zu realisieren, so dass bereits installierte und im Einsatz befindliche Werkzeuge nachträglich integriert und verwaltet werden können. Sowohl die Strategien für diese Art der Integration als auch die Methodik zum Deployment einer vollständigen Instanz des Frameworks wurden im Rahmen dieser Arbeit diskutiert.

Die Praxisrelevanz und Nutzbarkeit der Ergebnisse wurde durch die Implementierung des Frameworks auf Grundlage der entwickelten Architektur und Konzepte und durch die Bildung einer Instanz des Frameworks unter der URL `http://sselab.de` und deren erfolgreicher Einsatz in diversen Projekten in der Forschung, der Lehre und zur Kooperation mit industriellen Partnern gezeigt. Im Vordergrund stand dabei die Unterstützung verteilter Software-Entwicklungsprojekte mit agilen Prozessen. Darüber hinaus wurde diese Instanz auch für andere Projektarten eingesetzt. Um einen Überblick über die Nutzung der Instanz zu geben, wurden mehrere Forschungsprojekte und die Verwendung der Services innerhalb dieser Projekte beschrieben. Zusätzlich wurde in dieser Arbeit eine Einführung in das Nutzungs- und Betriebskonzept der Instanz gegeben.

10.2 Ausblick

Aufbauend auf den zuvor beschriebenen Ergebnissen dieser Arbeit können einige weitere Forschungsthemen behandelt sowie einige Erweiterungen des geschaffenen Frameworks vorgenommen werden. In diesem Abschnitt wird eine Auswahl davon vorgestellt.

Web- bzw. cloudbasierte IDEs sind heutzutage noch nicht so weit verbreitet wie desktopbasierte IDEs, werden jedoch zunehmend erforscht und eingesetzt [DJCD11, vDMC+10, Lan08]. Das in dieser Arbeit geschaffene Framework wurde unter Berücksichtigung dieses Trends konzipiert und kann daher als Grundlage zur Realisierung solcher IDEs verwendet werden. Dies hat eine erste Arbeit in diese Richtung bestätigt, in der eine Web-Infrastruktur geschaffen wurde, über die Ostp-Services aufgerufen und dadurch auf einfache Weise webbasierte Demonstratoren für die integrierten Werkzeuge realisiert werden können [Wie12]. Über diesen Mechanismus wurde beispielsweise das Simulationsframework von MontiArc [HRR12] als Demonstrator online zur Verfügung gestellt. Für eine effektive Realisierung webbasierter IDEs mit umfangreicherer Funktionalität müsste jedoch eine neue Backend-Art für das Framework konzipiert werden, die eine serverseitige Installation von Entwicklungswerkzeugen wie Compilern, Debuggern und Testumgebungen sowie einen interaktiven Zugriff darauf ermöglicht. Der in [vDMC+10] beschriebene Ansatz zur Installation von Werkzeugen der Eclipse IDE in einer Serverinfrastruktur könnte hier als Inspiration dienen, um entsprechende Mechanismen in das Framework einfließen zu lassen und darauf aufbauend das Potential und die Herausforderungen webbasierter IDEs zu erforschen.

Ein weiteres Thema das aufbauend auf den Ergebnissen dieser Arbeit betrachtet werden könnte, ist die automatisierte Bereitstellung projekt- und servicespezifischer Arbeitsumgebungen für Entwickler. Durch das Framework wird die Verwaltung und Konfiguration der integrierten Werkzeuge bereits effektiv unterstützt. Über existierende und die vom Framework bereitgestellten Clients können diese aus desktopbasierten IDEs genutzt werden. Darauf aufbauend könnte beispielsweise ein Ansatz zur automatisierten Erstellung und Aktualisierung virtueller Maschinen und der darin installierter Entwicklungsumgebungen umgesetzt werden. Initiale Konzepte diesbezüglich wurden im Rahmen dieser und anderer Arbeiten am Lehrstuhl bereits erarbeitet. In [The11, Eik12] wurden eine Konfigurationssprache und eine zugehörige Infrastruktur zur Verwaltung virtueller Maschinen auf Grundlage der Architektur und der Konzepte des Frameworks entwickelt. Ein weiterer Aspekt wurde in [Fre10] behandelt. Darin wurden Ostp-Services des Frameworks so erweitert, dass die Clients eines Services im zugehörigen Service-Plug-in ausgeliefert und automatisch in der Eclipse IDE installiert werden können. Insgesamt könnte also eine vollständig automatisierte Erstellung und Verwaltung projekt- und servicespezifischer Arbeitsumgebungen in Form von virtuellen Maschinen für Entwickler erreicht werden.

Im Rahmen dieser Arbeit wurde auch das Deployment von Werkzeugen betrachtet, die mit Hilfe der Services einer Framework-Instanz entwickelt worden sind. Beispielsweise wird MontiCore [KRV07b, KRV08, KRV10, Kra10] einerseits mit Hilfe der sselab.de-Instanz entwickelt und andererseits über diese Instanz als Service angeboten. Das Deployment erfolgt dabei automatisiert über den Build-Prozess von MontiCore unter Verwendung eines Framework-Clients. Die Mechanismen zum automatischen Deployment könnten jedoch noch weiter ausgebaut werden. Beispielsweise könnte ebenfalls aufbauend auf [The11, Eik12] eine neue Backend-Art oder einzelne Services für das Framework konzipiert werden, durch die Deployment-Aktivitäten (z. B.

Release, Installation, Aktualisierung) von Web- oder Serveranwendungen automatisiert durchgeführt werden können. Durch eine Framework-Instanz könnte dann beispielsweise sowohl die Entwicklung einer Anwendung mit Hilfe von MontiWIS [RR12] als auch deren Deployment erfolgen.

Die Unterstützung neuer Service-Kategorien bzw. neuer Backend-Arten ist ein weiterer Bereich, in dem das Framework ausgebaut werden könnte. Die Erweiterung erfordert aktuell die Anpassung verschiedener Bereiche innerhalb des Frontends: der Domänenklassen, der Geschäftslogik sowie der graphische Benutzeroberfläche. Durch die Erarbeitung geeigneter Konzepte zum strukturierten Aufbau der Benutzeroberflächen könnte dieser Erweiterungspunkt weiter verbessert werden. Einerseits könnte hier der Einsatz von Plug-in-Systemen innerhalb des Frontends aufbauend auf den Ergebnissen von [MW08] und [KD08] evaluiert werden. Ein weiterer vielversprechender Ansatz ist eine modellbasierte und generative Entwicklung von Teilen des Frontends unter Verwendung von MontiCore, wie beispielsweise in [Mar11] in Bezug auf die Generierung persistenter Datenmodelle behandelt wird.

Auf Grundlage der Social-Services des Frameworks könnte außerdem der Einfluss und Nutzen von Social Software in (global verteilten) Entwicklungsprojekten aufbauend auf offenen Fragestellungen in diesem Bereich [STvDC10, Chi08] erforscht werden. Dazu könnten zunächst die Social-Service-Kategorie so erweitert werden, dass zusätzlich zu Profilinformationen auch soziale Graphen verarbeitet werden können. Darüber hinaus bietet sich auch die Einbindung von Plattformen an, die speziell für die Aggregation von Profilinformationen von Entwicklern ausgelegt sind und von diesen zur Selbstdarstellung und Fortbildung genutzt werden [SFFC+13]. Die initialen Ergebnisse aus [Man12] zur Verarbeitung von Profilinformationen im Kontext von Entwicklungsprojekten, um die Zusammensetzung von Entwicklerteams zu analysieren und zu optimieren, könnten dabei ebenfalls zugrunde gelegt werden.

Literaturverzeichnis

[AAE+11] Timo Aho, Adnan Ashraf, Marc Englund, Joni Katajamäki, Johannes Koskinen, Janne Lautamäki, Antti Nieminen, Ivan Porres, Ilkka Turunen. Designing IDE as a Service. *Communications of Cloud Software*, 1:1–10, 2011.

[ABEKT11] Fredrik Asplund, Matthias Biehl, Jad El-Khoury, Martin Törngren. Tool Integration Beyond Wasserman. In *Proceedings of the CAiSE 2011 International Workshops*, Lecture Notes in Business Information Processing, Seiten 270–281. Springer, 2011.

[ABS02] Larry Augustin, Dan Bressler, Guy Smith. Accelerating Software Development Through Collaboration. In *Proceedings of the 24th International Conference on Software Engineering (ICSE '02)*, Seiten 559–563, New York, NY, USA, 2002. ACM.

[ACE12] Ajax.org. ACE - Ajax.org Cloud Editor. `http://ace.ajax.org/`, Stand: 04.05.2012.

[ACGL08] Fabio Abbattista, Fabio Calefato, Domenico Gendarmi, Filippo Lanubile. Incorporating Social Software into Distributed Agile Development Environments. In *Proceedings of the First International Workshop on Social Software Engineering and Applications*, Seiten 46–51, 2008.

[ADDK03] Frank Altheide, Sven Dörfel, Heiko Dörr, Jan Kanzleiter. An Architecture for a Sustainable Tool Integration. In *TIS 2003 Workshop on Tool Integration in System Development*, Seiten 29–32, 2003.

[ADS02] Frank Altheide, Heiko Dörr, Andy Schürr. Requirements to a Framework for Sustainable Integration of System Development Tools. In *Proceedings of the 3rd European Systems Engineering Conference (EuSEC'02)*, Seiten 53–57, 2002.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[AFH+05] Pär J. Ågerfalk, Brian Fitzgerald, Helena Holmström, Brian Lings, Björn Lundell, Eoin Ó. Conchúir. A Framework for Considering Opportunities and Threats in Distributed Software Development. In *Proceedings of the International Workshop on Distributed Software Development*, Seiten 47–61, Paris, France, 2005. Austrian Computer Society.

[AFHOC08] Pär J. Ågerfalk, Brian Fitzgerald, Helena Holmström Olsson, Eoin Ó. Conchúir. Benefits of Global Software Development: The Known and Unknown. In *Proceedings of the International Conference on Software Process (ICSP'08)*, Seiten 1–9, Berlin, Heidelberg, 2008. Springer-Verlag.

[AJLN08] Navid Ahmadi, Mehdi Jazayeri, Francesco Lelli, Sasa Nesic. A Survey of Social Software Engineering. In *ASE Workshops 2008: 23rd International Conference on Automated Software Engineering*, Seiten 1–12, 2008.

[BA04] Kent Beck, Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2004.

[Bäc06] Michael Bächle. Social Software. *Informatik Spektrum*, 29(2):121–124, 2006.

[Bal09] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Spektrum Lehrbücher der Informatik. Spektrum Akademischer Verlag, 3. Auflage, 2009.

[BASB+10] Predrag Buncic, Carlos Aguado Sanchez, Jakob Blomer, Leandro Franco, Artem Harutyunian, Pere Mato, Yushu Yao. CernVM - a virtual software appliance for LHC applications. *Journal of Physics: Conference Series*, 219(4):1–10, April 2010.

[BB03] Grady Booch, Alan W. Brown. Collaborative Development Environments. *Advances in Computers*, 59:1–27, 2003.

[BBH06] Olivier Berger, Christian Bac, Benoit Hamet. Integration of Libre Software Applications to Create a Collaborative Work Platform for Researchers at GET. *International Journal of Information Technology and Web Engineering*, 1(3):1–16, 2006.

[BBH+09] Christian Bartelt, Manfred Broy, Christoph Herrmann, Eric Knauss, Marco Kuhrmann, Andreas Rausch, Bernhard Rumpe, Kurt Schneider. Orchestration of Global Software Engineering Projects - Position Paper. In *Proceedings of the 3rd International Workshop on Tool Support Development and Management in Distributed Software Projects (REMIDI '09)*, Seiten 332–337, Limerick, Ireland, 2009.

[BBvB+12] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. Agile Manifesto. `http://agilemanifesto.org/`, Stand: 06.07.2012.

[BCK03] Len Bass, Paul Clements, Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, October 2003.

[BHL08] Peter Buxmann, Thomas Hess, Sonja Lehmann. Software as a Service. *Wirtschaftsinformatik*, 50:500–503, 2008.

[Bir05] Dorian Birsan. On Plug-ins and Extensible Architectures. *ACM Queue*, 3(2):40–46, March 2005.

[BKPS07] Manfred Broy, Ingolf H. Kruger, Alexander Pretschner, Christian Salzmann. Engineering Automotive Software. *Proceedings of the IEEE*, 95(2):356 –373, 2007.

[Blo08] Joshua Bloch. *Effective Java, Second Edition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2 Auflage, 2008.

[BM04] Paul V. Biron, Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, W3C, October 2004. `http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/`.

[BM11] Hans Buhl, Marco Meier. Die Verantwortung der Wirtschaftsinformatik bei IT-Großprojekten. *Wirtschaftsinformatik*, 53:59–62, 2011.

[BMJH96] Tilmann Bruckhaus, Nazim H. Madhavji, Ingrid Janssen, John Henshaw. The Impact of Tools on Software Productivity. *IEEE Software*, 13(5):29–38, September 1996.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[Boo07] Grady Booch. Introducing collaborative development environments. Technischer Bericht, IBM Corporation, August 2007. Software development - White paper.

[BP92] Alan W. Brown, Maria H. Penedo. An Annotated Bibliography on Integration in Software Engineering Environments . *SIGSOFT Software Engineering Notes*, 17(3):47–55, July 1992.

[BPM+08] Tim Bray, Jean Paoli, Eve Maler, François Yergeau, C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, W3C, November 2008. `http://www.w3.org/TR/2008/REC-xml-20081126/`.

[Bug12] Bugzilla. `http://www.bugzilla.org/`, Stand: 03.07.2012.

[BW98] Alan W. Brown, Kurt C. Wallnau. The Current State of CBSE. *IEEE Software*, 15(5):37–46, September 1998.

[CCD+08] Ping Cheng, Sunita Chulani, Ya B. Dang, Robert M. Delmonico, Yael Dubinsky, Kate Ehrlich, Mary E. Helander, Tim Klinger, Alexander Kofman, Paul M. Matchen, V. T. Rajan, Gilad Saadoun, Andrew Sempere, Peri L. Tarr, Clay Williams, Pei F. Xiang, Avi Yaeli, Shun X. Yang, Annie T. T. Ying. Jazz as a research platform: experience from the Software Development Governance Group at IBM Research. In *First International Workshop on Infrastructure for Research in Collaborative Software Engineering (IReCoSE '08) at FSE 2008*, 2008.

[CCL06] Ivica Crnkovic, Michel Chaudron, Stig Larsson. Component-Based Development Process and Component Lifecycle. In *Proceedings of the International Conference on Software Engineering Advances (ICSEA '06)*, Seiten 44–53, Washington, DC, USA, 2006. IEEE Computer Society.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note, W3C, March 2001. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[CdSH+03] Li-Te Cheng, Cleidson R. B. de Souza, Susanne Hupfer, John Patterson, Steven Ross. Building Collaboration into IDEs. *ACM Queue*, 1:40–50, December 2003.

[CFH+98] Antonio Carzaniga, Alfonso Fuggetta, Richard S. Hall, Dennis Heimbigner, André van der Hoek, Alexander L. Wolf. A Characterization Framework for Software Deployment Technologies. Technical Report CU-CS-857-98, University of Colorado, Department of Computer Science, April 1998.

[CGL09] Fabio Calefato, Domenico Gendarmi, Filippo Lanubile. Embedding Social Networking Information into Jazz to Foster Group Awareness within Distributed Teams. In *Proceedings of the 2nd International Workshop on Social Software Engineering and Applications (SoSEA '09)*, Seiten 23–28, New York, NY, USA, 2009. ACM.

[Cha09] Scott Chacon. *Pro Git*. Apress, Berkely, CA, USA, 2009.

[Chi08] Ed H. Chi. The Social Web: Research and Opportunities. *IEEE Computer*, 41(9):88–91, September 2008.

[CHR+03] Li-Te Cheng, Susanne Hupfer, Steven Ross, John Patterson, Bryan Clark, Cleidson de Souza. Jazz: A Collaborative Application Development Environment. In *Companion of the 18th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '03)*, Seiten 102–103, New York, NY, USA, 2003. ACM.

[CHRP03] Li-Te Cheng, Susanne Hupfer, Steven Ross, John Patterson. Jazzing up Eclipse with Collaborative Tools. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, Seiten 45–49, New York, NY, USA, 2003. ACM.

[CK09] Martin Campbell-Kelly. Historical Reflections: The Rise, Fall, and Resurrection of Software as a Service. *Communications of the ACM*, 52(5):28–30, May 2009.

[CKGS08] Martin Campbell-Kelly, Daniel D. Garcia-Swartz. Economic Perspectives on the History of the Computer Time-Sharing Industry, 1965-1985. *IEEE Annals of the History of Computing*, 30(1):16–36, January 2008.

[Cla99] James Clark. XSL Transformations (XSLT) Version 1.0. W3C recommendation, W3C, November 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[Col02] Ben Collins-Sussman. The Subversion Project: Buiding a Better CVS. *Linux Journal*, 2002(94), February 2002.

[COPT02] Éric Cousin, Gérald Ouvradou, Pascal Pucci, Samuel Tardieu. PicoLibre: a free collaborative platform to improve students' skills in software engineering. In *IEEE International Conference on Systems, Man and Cybernetics*, 2002.

[Cou12] couriermlm - The Courier mailing list manager. `http://www.courier-mta.org/couriermlm.html`, Stand: 18.09.2012.

[Cur02] Charles Curley. Emacs: the Free Software IDE. *Linux Journal*, 2002(98), June 2002.

[CW09] Jordi Cabot, Greg Wilson. Tools for Teams: A Survey of Web-Based Software Project Portals. *Dr. Dobb's Journal*, October 2009.

[Dea07] Alan Dearle. Software Deployment, Past, Present and Future. In *International Conference on Software Engineering (ICSE '07), Workshop on Future of Software Engineering (FOSE '07)*, Seiten 269–284, Washington, DC, USA, 2007. IEEE Computer Society.

[DHH+11] Constanze Deiters, Christoph Herrmann, Roland Hildebrandt, Eric Knauss, Marco Kuhrmann, Andreas Rausch, Bernhard Rumpe, Kurt Schneider. GloSE-Lab: Teaching Global Software Engineering. In *Proceedings of the 6th International Conference on Global Software Engineering (ICGSE '11)*, Seiten 156 –160, Helsinki, Finland, August 2011.

[DJCD11] Thomas Dvornik, David S. Janzen, John Clements, Olga Dekhtyar. Supporting Introductory Test-Driven Labs with WebIDE. In *Proceedings of the 24th Conference on Software Engineering Education and Training (CSEET '11)*, Seiten 51–60, Washington, DC, USA, 2011. IEEE Computer Society.

[DMH+08] John Vijay Sena Devide, Andrew Meneely, Chih-Wei Ho, Laurie Williams, Michael Devetsikiotis. Jazz Sangam: A Real-time Tool for Distributed Pair Programming of a Team Development Platform. In *First International Workshop on Infrastructure for Research in Collaborative Software Engineering (IReCoSE '08) at FSE 2008*, 2008.

[DOSG12] The Apache Software Foundation. Apache CXF – Distributed OSGi. `http://cxf.apache.org/distributed-osgi.html`, Stand: 26.09.2012.

[DP12] DrProject Website. https://stanley.cdf.toronto.edu/drproject/drproject, Stand: 19.03.2012.

[DRRS09] Michael Dukaczewski, Dirk Reiss, Bernhard Rumpe, Mark Stein. MontiWeb - Modular Development of Web Information Systems. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM '09)*, 2009.

[dRW04] Jim des Rivières, John Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2):371–383, April 2004.

[Ear90] Anthony Earl. Principles of a Reference Model for Computer Aided Software Engineering Environments. In *Software Engineering Environments*, Lecture Notes in Computer Science, Seiten 115–129. Springer Berlin / Heidelberg, 1990.

[EH11] Donald E. Eastlake, Tony Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, Internet Engineering Task Force (IETF), May 2011.

[Eik12] Robert Eikermann. Entwicklung eines integrierenden Services für die Erstellung von virtuellen Systemen. Bachelorarbeit, Software Engineering, RWTH Aachen University, 2012.

[EJB06] EJB 3.0 Expert Group. JSR 220: Enterprise JavaBeans, Version 3.0. Specification – Final Release, Sun Microsystem, Inc., 2006. http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.

[EK08] Khaled El Emam, A. Günes Koru. A Replicated Survey of IT Software Project Failures. *IEEE Software*, 25(5):84–90, September 2008.

[EV10] J. Laurenz Eveleens, Chris Verhoef. The Rise and Fall of the Chaos Report Figures. *IEEE Software*, 27:30–36, 2010.

[FCP11] Jacqui Finlay, Andy M. Connor, Russel Pears. Mining Software Metrics from Jazz. In *9th International Conference on Software Engineering Research, Management and Applications (SERA '11)*, Seiten 39–45, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[FF03] Christopher Ferris, Joel Farrell. What are Web Services? *Communications of the ACM*, 46(6):31, June 2003.

[FGM+99] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul J. Leach, Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force (IETF), 1999.

[FHBH+99] John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, Lawrence C. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, Internet Engineering Task Force (IETF), 1999.

[FKP+10] M. Norbert Fisch, Thomas Kurpick, Claas Pinkernell, Stefan Plesser, Bernhard Rumpe. The Energy Navigator - A Web based Platform for functional Quality Mangement in Buildings. In *Proceedings of the 10th International Conference for Enhanced Building Operations (ICEBO' 10)*, Kuwait City, Kuwait, October 2010.

[FLP+11] M. Norbert Fisch, Markus Look, Claas Pinkernell, Stefan Plesser, Bernhard Rumpe. State-Based Modeling of Buildings and Facilities. In *Proceedings of the 11th International Conference for Enhanced Building Operations (ICEBO' 11)*, New York City, USA, October 2011.

[Fon03] John Fontana. Collaborative Software Ages Slowly. *Network World Fusion*, January 2003.

[Fow02] Martin Fowler. Public versus Published Interfaces. *IEEE Software*, 19(2):18–19, March 2002.

[Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. The Addison-Wesley Signature Series. Addison-Wesley, Boston, MA, USA, 2003.

[Fow06] Martin Fowler. Continuous Integration. `http://martinfowler.com/articles/continuousIntegration.html`, 2006. Stand: 26.04.2012.

[FPPR12] M. Norbert Fisch, Claas Pinkernell, Stefan Plesser, Bernhard Rumpe. The Energy Navigator – A Web-Platform for Performance Design and Management. In *Proceedings of the 7th International Conference on Energy Efficiency in Commercial Buildings (IEECB '12)*, Frankfurt a. M., Germany, April 2012.

[Fre10] Thomas Freese. Automatic installation of OSTP clients in Eclipse. Bachelorarbeit, Software Engineering, RWTH Aachen University, 2010.

[Fro07] Randall Frost. Jazz and the Eclipse Way of Collaboration. *IEEE Software*, 24:114–117, November 2007.

[FS97] Mohamed Fayad, Douglas C. Schmidt. Object-Oriented Application Frameworks. *Communications of the ACM*, 40:32–38, October 1997.

[FW93] Brian Fromme, Judy Walker. An Open Architecture for Tool and Process Integration. In *Proceedings of the Software Engineering Environments Conference*, Seiten 50–62, July 1993.

[GAO95] David Garlan, Robert Allen, John Ockerbloom. Architectural Mismatch: Why Reuse Is So Hard. *IEEE Software*, 12(6):17–26, November 1995.

[Géd10] Walid Joseph Gédéon. *OSGi and Apache Felix 3.0 Beginner's Guide*. Packt Publishing, Limited, 2010.

[GF12] GForge Collaborative Environment. `http://gforge.org/gf/`, Stand: 28.04.2012.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[GHM+05] Olivier Gruber, B. J. Hargrave, Jeff McAffer, Pascal Rapicault, Thomas Watson. The Eclipse 3.0 platform: Adopting OSGi technology. *IBM Systems Journal*, 44(2):289–299, January 2005.

[GKRS06] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler. Integration von Modellen in einen codebasierten Softwareentwicklungsprozess. In *Proceedings of Modellierung 2006 (LNI P-82)*, 2006.

[Gla12] Glassfish, Open Source Application Server. `http://glassfish.java.net/de/`, Stand: 04.05.2012.

[Gla05] Robert L. Glass. IT Failure Rates – 70% or 10-15%? *IEEE Software*, 22(3):112–111, May 2005.

[Gli00] Martin Glinz. Improving the Quality of Requirements with Scenarios. In *Proceedings of the Second World Congress on Software Quality*, Seiten 55–60, 2000.

[Got08] Greg Goth. Software-as-a-Service: The Spark That Will Change Software Engineering? *IEEE Distributed Systems Online*, 9(7), July 2008.

[Gru94] Jonathan Grudin. Computer-Supported Cooperative Work: History and Focus. *IEEE Computer*, 27(5):19–26, May 1994.

[GTHM01] Oliver Günther, Gerrit Tamm, Lars Hansen, Thomas Meseg. Application Service Providers: Angebot, Nachfrage und langfristige Perspektiven. *Wirtschaftsinformatik*, 43(6):555–567, 2001.

[Han11] Adrian Hannah. A Primer to the OAuth Protocol. *Linux Journal*, 2011(206), June 2011.

[Häs11] Matthias Häsel. OpenSocial: An Enabler for Social Applications on the Web. *Communications of the ACM*, 54(1):139–144, January 2011.

[HCRP04] Susanne Hupfer, Li-Te Cheng, Steven Ross, John Patterson. Introducing Collaboration into an Application Development Environment. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW '04)*, Seiten 21–24, New York, NY, USA, 2004. ACM.

[Her05] Christoph Herrmann. Automatische Ableitung von XML-Schema aus XML-Dokumenten. Studienarbeit, Institut für Software Systems Engineering, Carl-Friedrich-Gauß-Fakultät, Technische Universität Braunschweig, 2005.

[Her06] Christoph Herrmann. Online Software Transformation Platform. Diplomarbeit, Institut für Software Systems Engineering, Carl-Friedrich-Gauß-Fakultät, Technische Universität Braunschweig, 2006.

[HKR12a] Christoph Herrmann, Thomas Kurpick, Bernhard Rumpe. Agile User-Feedback and its Management through the SSELab Feedback System. Technischer Bericht AIB-2012-11, RWTH Aachen, 2012.

[HKR12b] Christoph Herrmann, Thomas Kurpick, Bernhard Rumpe. SSELab: A Plug-In-Based Framework for Web-Based Project Portals. In *Proceedings of the 2nd ICSE Workshop on Developing Tools as Plug-Ins (TOPI '12)*, Seiten 61–66, Zurich, Switzerland, June 2012.

[HL10] Eran Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849, Internet Engineering Task Force (IETF), April 2010.

[HLLL04] Alifiya Hussain, Doris Lee, Michelle Levesque, Reza Lotun. GForge: A Modular Support Framework for Team Programming Projects. Technischer Bericht, Department of Computer Science, University of Toronto, 2004.

[HOT00] William Harrison, Harold Ossher, Peri Tarr. Software Engineering Tools and Environments: A Roadmap. In *22nd International Conference on on Software Engineering (ICSE '00), Future of Software Engineering Track*, Seiten 261–277, New York, NY, USA, 2000. ACM.

[HRR12] Arne Haber, Jan Oliver Ringert, Bernard Rumpe. MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technischer Bericht AIB-2012-03, RWTH Aachen, Februar 2012.

[HZ09] Kim Herzig, Andreas Zeller. Mining the Jazz Repository: Challenges and Opportunities. In *Proceedings of the 6th International Working Conference on Mining Software Repositories (MSR '09)*, Seiten 159–162, Washington, DC, USA, 2009. IEEE Computer Society.

[ISO96] ISO/IEC. Information technology — Syntactic metalanguage — Extended BNF (ISO/IEC 14977:1996). International Standard, 1996.

[Jac05] Dean Jacobs. Enterprise Software as Service. *ACM Queue*, 3(6):36–42, July 2005.

[Jaz12] IBM Corporation. Jazz Community Site. `https://jazz.net/`, Stand: 11.06.2012.

[Jaz07] Mehdi Jazayeri. Some Trends in Web Application Development. In *International Conference on Software Engineering (ICSE '07), Workshop on Future of Software Engineering (FOSE '07)*, Seiten 199–213, Washington, DC, USA, 2007. IEEE Computer Society.

[JEE06] Java Platform, Enterprise Edition 5 (Java EE 5). Specification – Final Release, Sun Microsystem, Inc., 2006. `http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html`.

[JF88] Ralph E. Johnson, Brian Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1:22–35, June/July 1988.

[Jir12] Atlassian. Issue & Project Tracking Software | Atlassian JIRA. `http://www.atlassian.com/software/jira/overview`, Stand: 03.07.2012.

[Joh97] Ralph E. Johnson. Frameworks = (Components + Patterns). *Communications of the ACM*, 40:39–42, October 1997.

[Jos06] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, Internet Engineering Task Force (IETF), October 2006.

[JPTO12] IBM Corporation. Jazz Platform Technical Overview. `https://jazz.net/library/LearnItem.jsp?href=content/docs/platform-overview/index.html`, Stand: 11.06.2012.

[JWLM13] Markus Jahn, Reinhard Wolfinger, Markus Löberbauer, Hanspeter Mössenböck. Composing user-specific web applications from distributed plug-ins. *Computer Science - Research and Development*, 28:85–105, 2013.

[JWM10] Markus Jahn, Reinhard Wolfinger, Hanspeter Mössenböck. Extending Web Applications with Client and Server Plug-ins. In *Software Engineering*, Seiten 33–44, 2010.

[KCSS10] Moo Nam Ko, Gorrell P. Cheek, Mohamed Shehab, Ravi Sandhu. Social-Networks Connect Services. *IEEE Computer*, 43(8):37 –43, aug. 2010.

[KD08] Simon Richard Kaegi, Dwight Deugo. Modular Java Web Applications. In *Proceedings of the 2008 Symposium on Applied Computing (SAC '08)*, Seiten 688–693, New York, NY, USA, 2008. ACM.

[KK09] Daniel Kadenbach, Carsten Kleiner. Benefits and Challenges of Using Collaborative Development Environments with Social Software in Higher Computer Science Education. In *Proceedings of the 3rd International Conference on Online Communities and Social Computing: Held as Part of HCI International 2009*, OCSC '09, Seiten 479–487, Berlin, Heidelberg, 2009. Springer-Verlag.

[KK11] Daniel Kadenbach, Carsten Kleiner. Recent Trends in Software Support for Online Communities for Teaching and Research Projects in Higher Education. In *Proceedings of the 4th International Conference on Online Communities and Social Computing: Held as Part of HCI International 2011*, OCSC'11, Seiten 50–59, Berlin, Heidelberg, 2011. Springer-Verlag.

[KM06] Mik Kersten, Gail C. Murphy. Using Task Context to Improve Programmer Productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, Seiten 1–11, New York, NY, USA, 2006. ACM.

[KPLR12] Thomas Kurpick, Claas Pinkernell, Markus Look, Bernhard Rumpe. Modeling Cyber-Physical Systems: Model-Driven Specification of Energy Efficient Buildings. In *Proceedings of the Modelling of the Physical World Workshop (MOTPW '12)*, Seiten 2:1–2:6, New York, NY, USA, 2012. ACM.

[KPW04] Sunghun Kim, Kai Pan, E. James Whitehead. WebDAV based Open Source Collaborative Development Environment. In *Proceedings of the 4th ICSE Workshop on Open Source Software Engineering*, Seiten 53–57, 2004.

[Kra10] Holger Krahn. *MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering*. Aachener Informatik-Berichte, Software Engineering, Band 1. Shaker Verlag, 2010.

[KRV07a] Holger Krahn, Bernhard Rumpe, Steven Völkel. Efficient Editor Generation for Compositional DSLs in Eclipse. In *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM '07)*, 2007.

[KRV07b] Holger Krahn, Bernhard Rumpe, Steven Völkel. Integrated Definition of Abstract and Concrete Syntax for Textual Languages. In *Proceedings of Models 2007*, Seiten 286–300, 2007.

[KRV08] Holger Krahn, Bernhard Rumpe, Steven Völkel. MontiCore: Modular Development of Textual Domain Specific Languages. In *Proceedings of Tools Europe*, Lecture Notes in Business Information Processing. Springer-Verlag Berlin-Heidelberg, 2008.

[KRV10] Holger Krahn, Bernhard Rumpe, Steven Völkel. MontiCore: a Framework for Compositional Development of Domain Specific Languages. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(5):353–372, September 2010.

[KS06] Mike Keith, Merrick Schincariol. *Pro EJB 3: Java Persistence API*. Apress, Berkely, CA, USA, 2006.

[Lan08] Fabian Lange. *Eclipse Rich Ajax Platform: Bringing Rich Client to the Web*. Apress, Berkely, CA, USA, 2008.

[Lan09] Filippo Lanubile. Collaboration in Distributed Software Development. In *Software Engineering*, Lecture Notes in Computer Science, Seiten 174–193. Springer-Verlag, Berlin, Heidelberg, 2009.

[LEPV10] Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, Aurora Vizcaino. Collaboration Tools for Global Software Engineering. *IEEE Software*, 27:52–55, 2010.

[LH07] Steve Loughran, Erik Hatcher. *Ant in Action*. Manning Publications Co., 2007.

[LHK+12] Olga Liskin, Christoph Herrmann, Eric Knauss, Thomas Kurpick, Bernhard Rumpe, Kurt Schneider. Supporting Acceptance Testing in Distributed Software Projects with Integrated Feedback Systems: Experiences and Requirements. In *Proceedings of 7th International Conference on Global Software Engineering (ICGSE '12)*, Seiten 84 – 93, Porto Alegre, Brazil, 2012. IEEE Computer Society.

[Lin12] LinkedIn Corporation. LinkedIn. `http://www.linkedin.com/`, Stand: 05.07.2012.

[LL02] Ben Laurie, Peter Laurie. *Apache: The Definitive Guide*. O'Reilly Media, 3. Auflage, 2002.

[LNK+12] Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, Tommi Mikkonen, Marc Englund. CoRED – Browser-based Collaborative Real-Time Editor for Java Web Applications. In *Proceedings of the 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*, Seiten 1307–1316, New York, NY, USA, 2012. ACM.

[Lou06a] Panagiotis Louridas. Using Wikis in Software Development. *IEEE Software*, 23:88–91, 2006.

[Lou06b] Panagiotis Louridas. Version Control. *IEEE Software*, 23:104–107, 2006.

[LZV08] Phillip A. Laplante, Jia Zhang, Jeffrey Voas. What's in a Name? Distinguishing between SaaS and SOA. *IT Professional*, 10(3):46–50, May 2008.

[Man12] Zied Mansouri. Using Social Networking Information to Enhance Collaboration in the SSELab. Masterarbeit, Software Engineering, RWTH Aachen University, 2012.

[Mar01] Klaus Marquardt. Patterns for Plug-Ins. In *Proceedings of the 4th European Conference on Pattern Languages of Programming and Computing (EuroPLoP '99)*, Seiten 203–232, Bad Irsee, Germany, 2001. UVK - Universitätsverlag Konstanz.

[Mar11] Jan Marten. Development of a Framework for Generating Persistent Data Models. Masterarbeit, Software Engineering, RWTH Aachen University, 2011.

[McI68] M. Douglas McIlroy. Mass-Produced Software Components. In *Proceedings of the NATO Conference on Software Engineering*, Seiten 88–98, Garmisch, Germany, 1968. NATO Science Committee.

[Mec12] Daniel Meckenstock. Entwicklung eines Dokumenten-Transformations-Services für die Online Software Transformation Platform. Bachelorarbeit, Software Engineering, RWTH Aachen University, 2012.

[Mho12] MHonArc - A mail-to-HTML converter. `http://www.mhonarc.org/`, Stand: 18.09.2012.

[MLA10] Jeff McAffer, Jean-Michel LeMieux, Chris Aniszczyk. *Eclipse Rich Client Platform*. Eclipse Series. Addison-Wesley Professional, 2nd Auflage, 2010.

[MMS03] Johannes Mayer, Ingo Melzer, Franz Schweiggert. Lightweight Plug-In-Based Application Development. In *Objects, Components, Architectures, Services, and Applications for a Networked World*, Lecture Notes in Computer Science, Seiten 87–102. Springer Berlin / Heidelberg, 2003.

[MOB+11] Manfred Moser, Tim O'Brien, Damian Bradicich, John Casey, Tamás Cservenák, Brian Demers, Brian Fox, Marvin Froeder, Anders Hammar, Rich Seddon, Juven Xu, Jason van Zyl. *Repository Management with Nexus*. Sonatype, Inc., 3.2 Auflage, 2011.

[Moz12] Mozilla. Developer Hub :: Add-ons for Firefox. `https://addons.mozilla.org/en-US/developers/`, Stand: 16.05.2012.

[Mur07] San Murugesan. Understanding Web 2.0. *IT Professional*, 9(4):34–41, July 2007.

[MV03] Klaus Marquardt, Markus Völter. Plug-Ins – applikationsspezifische Komponenten. *JavaSpektrum*, 2:20 – 25, 2003.

[MW12] MediaWiki Website. `http://www.mediawiki.org/`, Stand: 20.04.2012.

[MW08] Dirk Mascher, Björn Wand. Serverside OSGi: JEE und OSGi integrieren. *Java Magazin*, 7:81–88, 2008.

[Nex12] Sonatype Inc. Sonatype.org: Nexus. `http://www.sonatype.org/nexus/`, Stand: 20.04.2012.

[NSD08] Thanh H. D. Nguyen, Adrian Schröter, Daniela Damian. Mining Jazz: An Experience Report. In *First International Workshop on Infrastructure for Research in Collaborative Software Engineering (IReCoSE '08) at FSE 2008*, 2008.

[NWD08] Thanh Nguyen, Timo Wolf, Daniela Damian. Global Software Development and Delay: Does Distance Still Matter? In *Proceedings of the 3rd International Conference on Global Software Engineering (ICGSE '08)*, Seiten 45–54, Washington, DC, USA, 2008. IEEE Computer Society.

[OMG06] Object Management Group. Deployment and Configuration of Component-based Distributed Applications Specification. Specification, April 2006. `http://www.omg.org/spec/DEPL/4.0/PDF/`.

[OMG10a] Object Management Group. OMG Unified Modeling Language (OMG UML), Infrastructure Version 2.3 (10-05-03). Specification, May 2010. `http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/`.

[OMG10b] Object Management Group. OMG Unified Modeling Language (OMG UML), Superstructure Version 2.3 (10-05-03). Specification, May 2010. `http://www.omg.org/spec/UML/2.3/Superstructure/PDF/`.

[OS11a] OpenSocial and Gadgets Specification Group. OpenSocial Specification 2.0.1. Technical Specification, 2011. http://opensocial-resources.googlecode.com/svn/spec/2.0.1/OpenSocial-Specification.xml.

[OS11b] OpenSocial and Gadgets Specification Group. OpenSocial Social Data Specification 2.0.1. Technical Specification, 2011. http://opensocial-resources.googlecode.com/svn/spec/2.0.1/Core-Data.xml.

[OSG09a] OSGi Alliance. *OSGi Service Platform, Core Specification, Release 4, Version 4.2*. aQute Publishing, 2009.

[OSG09b] OSGi Alliance. *OSGi Service Platform, Service Compendium, Release 4, Version 4.2*. aQute Publishing, 2009.

[OSLC12] Open Services for Lifecycle Collaboration. http://open-services.net/, Stand: 14.06.2012.

[Par72] David L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.

[PCF08] C. Michael Pilato, Ben Collins-Sussman, Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, 2. Auflage, 2008.

[Per13] Tim Perdue. The GForge Story. http://perdue.net/personal/gforgestory.php, Stand: 11.01.2013.

[PM09] Shelly Park, Frank Maurer. The Role of Blogging in Generating a Software Product Vision. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE '09)*, Seiten 74–77, Washington, DC, USA, 2009. IEEE Computer Society.

[PP12] Fachhochschule Hannover. Projektserver Proanvil. http://proanvil.inform.fh-hannover.de/, Stand: 19.03.2012.

[Pre95] Wolfgang Pree. *Design Patterns for Object-Oriented Software Development*. Addison Wesley Longman, 1. Auflage, 1995.

[PVEP10] Javier Portillo-Rodriguez, Aurora Vizcaino, Christof Ebert, Mario Piattini. Tools to Support Global Software Development Processes: A Survey. In *Proceedings of the 5th International Conference on Global Software Engineering (ICGSE '10)*, Seiten 13–22, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[Rat08] Jörg Rathlev. Plug-ins: an Architectural Style for Component Software. In *Proceedings of the thirteenth International Workshop on Component-Oriented Programming (WCOP 2008)*, Seiten 5–9, Karlsruhe, Germany, Oktober 2008. Universität Karlsruhe, Fakultät für Informatik.

[Rat11] Jörg Rathlev. Anwendungsentwicklung mit Plug-in-Architekturen: Erfahrungen aus der Praxis. In *Software Engineering*, Lecture Notes in Informatics, Seiten 183–194, Bonn, 2011. Gesellschaft für Informatik.

[Rei96] Steven P. Reiss. Software Tools and Environments. *ACM Computing Surveys*, 28(1):281–284, March 1996.

[Ric10] Scott Rich. IBM's Jazz Integration Architecture: Building a Tools Integration Architecture and Community Inspired by the Web. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*, Seiten 1379–1382, New York, NY, USA, 2010. ACM.

[RJ96] Don Roberts, Ralph Johnson. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. In *Proceedings of the Third Conference on Pattern Languages and Programming*. Addison-Wesley, 1996.

[Rob05] Jason Robbins. Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools. In *Perspectives on Free and Open Source Software*, Seiten 245–264. The MIT Press, Cambridge, MA, June 2005.

[RR12] Dirk Reiß, Bernhard Rumpe. Using Lightweight Activity Diagrams for Modeling and Generation of Web Information Systems. In *Proceedings of the 4th International United Information Systems Conference (UNISCON '12) (to appear)*, 2012.

[Rum11] Bernhard Rumpe. *Modellierung mit UML*. Springer Berlin, 2te Auflage, September 2011.

[Rum12] Bernhard Rumpe. *Agile Modellierung mit UML*. Springer Berlin, 2te Auflage, März 2012.

[RW07] Karen L. Reid, Gregory V. Wilson. DrProject: A Software Project Management Portal to Meet Educational Needs. In *Proceedings of the 38th SIGCSE technical symposium on Computer Science Education*, SIGCSE '07, Seiten 317–321, New York, NY, USA, 2007. ACM.

[Rya07] William Ryan. Web-Based Java Integrated Development Environment. BEng Thesis, University of Edinburgh, Division of Informatics, 2007.

[Sar05] Anita Sarma. A Survey of Collaborative Tools in Software Development. Technischer Bericht UCI-ISR-05-3, Institute for Software Research, Donald Bren School of Information and Computer Sciences, University of California, Irvine, March 2005.

[SB02] Ken Schwaber, Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 1. Auflage, 2002.

[SBPM08] David Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, zweite Auflage, December 2008.

[SC12] SensorCloud. http://sensorcloud.de, Stand: 08.10.2012.

[Sch12] Martin Schindler. *Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P*. Aachener Informatik-Berichte, Software Engineering, Band 11. Shaker Verlag, 2012.

[SCR+03] Spencer Shepler, Brent Callaghan, David Robinson, Robert Thurlow, Carl Beame, Mike Eisler, David Noveck. Network File System (NFS) version 4 Protocol. RFC 3530, Internet Engineering Task Force (IETF), April 2003.

[SF12] Geeknet Inc. SourceForge. http://sourceforge.net/, Stand: 19.03.2012.

[SFFC+13] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, Kurt Schneider. Mutual Assessment in the Social Programmer Ecosystem: An Empirical Investigation of Developer Profile Aggregators. In *Proceedings of the ACM 2013 conference on Computer Supported Cooperative Work and Social Computing (CSCW '13)*, Seiten 103–116, New York, NY, USA, 2013. ACM.

[Sma11] John Ferguson Smart. *Jenkins: The Definitive Guide*. O'Reilly Media, 2011.

[Som03] Peter Sommerlad. Reverse Proxy Patterns. In *Proceedings of the 8th European Conference on Pattern Languages of Programms (EuroPLoP '2003)*, Seiten 431–458, Irsee, Germany, 2003. UVK - Universitätsverlag Konstanz.

[Sta81] Richard M. Stallman. EMACS The Extensible, Customizable Self-Documenting Display Editor. In *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, Seiten 147–156, New York, NY, USA, 1981. ACM.

[Ste87] Vic Stenning. On the Role of an Environment. In *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)*, Seiten 30–35, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[STvDC10] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, Li-Te Cheng. The Impact of Social Media on Software Engineering Practices and Tools. In *Proceedings of the FSE/SDP workshop on Future of Software Engineering Research (FoSER '10)*, Seiten 359–364, New York, NY, USA, 2010. ACM.

[SVBP07] Jeff Sutherland, Anton Viktorov, Jack Blount, Nikolai Puntikov. Distributed Scrum: Agile Project Management with Outsourced Development Teams. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS '07)*, Washington, DC, USA, 2007. IEEE Computer Society.

[Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2te Auflage, 2002.

[Szy03] Clemens Szyperski. Component Technology: What, Where, and How? In *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, Seiten 684–693, Washington, DC, USA, 2003. IEEE Computer Society.

[TBB03] Mark Turner, David Budgen, Pearl Brereton. Turning Software into a Service. *IEEE Computer*, 36(10):38–44, October 2003.

[TBMM04] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. W3C Recommendation, W3C, October 2004. `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`.

[The11] Christian Theis. Entwicklung einer DSL für Modelle von virtuellen Systemen. Bachelorarbeit, Software Engineering, RWTH Aachen University, 2011.

[TN92] Ian Thomas, Brian A. Nejmeh. Definitions of Tool Integration for Environments. *IEEE Software*, 9(2):29–35, March 1992.

[Tra12] Edgewall Software. The Trac Project. `http://trac.edgewall.org/`, Stand: 06.04.2012.

[vDMC+10] Arie van Deursen, Ali Mesbah, Bas Cornelissen, Andy Zaidman, Martin Pinzger, Anja Guzzi. Adinda: A Knowledgeable, Browser-Based IDE. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE '10) - Volume 2*, Seiten 203–206, New York, NY, USA, 2010. ACM.

[Vö11] Steven Völkel. *Kompositionale Entwicklung domänenspezifischer Sprachen*. Aachener Informatik-Berichte, Software Engineering, Band 9. Shaker Verlag, 2011.

[Was90] Anthony I. Wasserman. Tool Integration in Software Engineering Environments. In *Software Engineering Environments*, Lecture Notes in Computer Science, Seiten 137–149. Springer Berlin / Heidelberg, 1990.

[WD05] Mike N. Wicks, Rick G. Dewar. Ad Hoc Tool Integration; a Case Study. In *Proceedings of the International Conference on Software Development (SWDC-REK)*, Seiten 214–219, 2005.

[WD07] Mike N. Wicks, Rick G. Dewar. Controversy Corner: A new research agenda for tool integration. *Journal of Systems and Software*, 80(9):1569–1585, September 2007.

[WDPM06] Reinhard Wolfinger, Deepak Dhungana, Herbert Prähofer, Hanspeter Mössenböck. A Component Plug-In Architecture for the .NET Platform. In *Proceedings of the 7th Joint Conference on Modular Programming Languages (JMLC '06)*, Seiten 287–305, Berlin, Heidelberg, 2006. Springer-Verlag.

[Web11] Michael Weber. Bot-Accounts zur Autorisierung von Service-Interaktionen im SSELab. Bachelorarbeit, Software Engineering, RWTH Aachen University, 2011.

[WG99] E. James Whitehead, Jr., Yaron Y. Goland. WebDAV: A network protocol for remote collaborative authoring on the Web. In *Proceedings of the sixth European Conference on Computer Supported Cooperative Work (ECSCW'99)*, Seiten 291–310, Norwell, MA, USA, 1999. Kluwer Academic Publishers.

[Whi07] Jim Whitehead. Collaboration in Software Engineering: A Roadmap. In *International Conference on Software Engineering (ICSE '07), Workshop on Future of Software Engineering (FOSE '07)*, Seiten 214–225, Washington, DC, USA, 2007. IEEE Computer Society.

[WHKL08] Gerd Wütherich, Nils Hartmann, Bernd Kolb, Matthias Lübken. *Die OSGi Service Platform: Eine Einführung mit Eclipse Equinox*. dpunkt.verlag GmbH, 2008.

[Wic06] Mike N. Wicks. Tool Integration within Software Engineering Environments: An Annotated Bibliography. Technischer Bericht, Heriot-Watt University, 2006.

[Wie12] Paul Wiedeking. Entwicklung von Web-Demonstratoren für OSTP-Services. Bachelorarbeit, Software Engineering, RWTH Aachen University, 2012.

[Wol08] Reinhard Wolfinger. Plug-in Architecture and Design Guidelines for Customizable Enterprise Applications. In *Companion to the 23rd Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA '08)*, Seiten 893–894, New York, NY, USA, 2008. ACM.

[WP12] Wordpress - Blog Tool, Publishing Platform, and CMS. `http://wordpress.org/`, Stand: 05.05.2012.

[WP07] Reinhard Wolfinger, Herbert Prähofer. Integration Models in a .NET Plug-in Framework. In *Software Engineering (SE 2007)*, Seiten 217–230, 2007.

[YBCD08] Jin Yu, Boualem Benatallah, Fabio Casati, Florian Daniel. Understanding Mashup Development. *IEEE Internet Computing*, 12(5):44–52, September 2008.

[YEC+08] Annie T. T. Ying, Kate Ehrlich, Li-Te Cheng, Harold L. Ossher, Thomas V. Frauenhofer, Frank van Ham. Jazz development data: a community perspective. In *First International Workshop on Infrastructure for Research in Collaborative Software Engineering (IReCoSE '08) at FSE 2008*, 2008.

[Zel07] Andreas Zeller. The Future of Programming Environments: Integration, Synergy, and Assistance. In *International Conference on Software Engineering (ICSE '07), Workshop on Future of Software Engineering (FOSE '07)*, Seiten 316–325, Washington, DC, USA, 2007. IEEE Computer Society.

[ZW11] Carl Zetie, John Wiegand. The Business Value of the Jazz Architecture: How the Right Architectural Choices Unlock the Value of Lifecycle Tools. Technischer Bericht REDP-4781-00, IBM Corporation, September 2011.

Abbildungsverzeichnis

1.1 Abhängigkeit von Funktionsumfang und Langlebigkeit von Integrationslösungen (adaptiert aus [WD07]) . . . 8
1.2 Trends und Herausforderungen im Bereich von Entwicklungsumgebungen . . . 9

2.1 Toaster-Referenzmodell (adaptiert aus [Ear90]) . . . 15
2.2 Hierarchische Struktur eines komponentenbasierten Systems . . . 19
2.3 Schematische Darstellung einer klassischen und einer reinen Plug-in-Architektur (adaptiert aus [Bir05, Rat11]) . . . 23

3.1 Rollen und deren Interaktion mit dem SSELab . . . 31

4.1 Service-Kategorien des SSELab-Frameworks . . . 44
4.2 Kontotypen des SSELab-Frameworks . . . 46
4.3 Projekte und deren Beziehungen zu Konten und Services . . . 47
4.4 Komponentendiagramm der Architektur des SSELab-Frameworks . . . 48
4.5 Base-Backend-Instanz mit zwei installierten Service-Plug-ins und den zugehörigen Services . . . 52
4.6 Grundlegende Architektur aller Backend-Arten . . . 55
4.7 Installation und Aktualisierung eines Plug-ins in einer Backend-Instanz . . . 56
4.8 Deinstallation von Plug-ins in einer Backend-Instanz . . . 58
4.9 Komponentendiagramm der Frontend-Architektur . . . 61
4.10 Verteilungsdiagramm mit einer Frontend- und drei Base-Backend-Instanzen . . . 63
4.11 Eigenschaften der Service-Kategorien . . . 64
4.12 Klassen zur Verwaltung von Service-Plug-ins . . . 64
4.13 Domänenklassen zur Repräsentation und Identifizierung von Backend-Instanzen 65
4.14 Sequenzdiagramm zur Installation eines neuen Services auf drei Backend-Instanzen 67
4.15 Sequenzdiagramm zur Synchronisation von Services der Backend-Instanzen . . . 69
4.16 Details der Kontotypen im SSELab-Framework . . . 70
4.17 Domänenmodell zur Verwaltung von Profilinformationen der Benutzer . . . 71
4.18 Interaktionsvarianten bei denen Bot-Accounts zum Einsatz kommen . . . 73
4.19 Projekte und deren Beziehungen zu weiteren Klassen des Domänenmodells . . . 76
4.20 Domänenmodell zur Repräsentation von Einladungen zu Projekten . . . 77
4.21 Klassen zur Beschreibung der Beziehungen zwischen Services und Projekten . . . 78
4.22 Domänenmodell zur Verwaltung von Clients . . . 79
4.23 Komponenten der Framework-Client-APIs . . . 83
4.24 Realisierte Framework-Clients . . . 84

5.1 Erweiterungspunkte des SSELab-Frameworks . . . 87
5.2 Methodik zur Unterstützung neuer Service-Kategorien . . . 89
5.3 Neu zu erzeugende Komponenten eines Web-IDE-Backends . . . 90
5.4 Erweiterung der Schichtenarchitektur des Frontends . . . 92
5.5 Rollen der Komponenten des Base-Backends in der Plug-in-Architektur . . . 96
5.6 Vererbungshierarchie des Subversion-Plug-ins . . . 97
5.7 Klassen der Komponente Common-Plug-in-API . . . 98
5.8 Konfiguration für den Zugriff auf Subversion-Repositories durch den CI-Server mit Hilfe von Bot-Accounts . . . 101
5.9 Methodik zur Integration von Anwendungen als Base-Services . . . 102
5.10 Klassen der Komponenten Base-Plug-in-API und Common-Plug-in-API . . . 103
5.12 Klassen zur Definition von Konfigurationsoptionen eines Base-Services . . . 106
5.13 Methodik zur Integration von Werkzeugen als Ostp-Services . . . 108
5.14 Kategorien von Ostp-Services mit ihrem charakteristischen Ein- und Ausgabeverhalten . . . 109
5.15 Klassen der Komponenten Ostp-Plug-in API und Common-Plug-in-API . . . 110
5.16 Klassen zur Definition der Parameter eines Ostp-Services . . . 111
5.17 Methoden der Ostp-Plug-in-API zum Aufruf eines dateiverarbeitenden Werkzeugs 112
5.18 Rückgabetypen der Ostp-Plug-in-API . . . 113
5.19 Autorisierung des Zugriffs auf geschützte Ressourcen durch einen Client mit Hilfe von OAuth . . . 115
5.20 Methodik zur Integration von Funktionen sozialer Netzwerke als Social-Services 116
5.21 Klassen der Komponenten Social-Plug-in-API und Common-Plug-in-API . . . 117
5.22 Parameter- und Rückgabetyp der Social-Plug-in-API . . . 118
5.23 Architektur der Framework-Client-APIs . . . 120
5.24 Auszug der APIs aus User-Client-Common und Admin-Client-Common . . . 121
5.25 High-Level-API für die Ausführung von Ostp-Services . . . 122

6.1 Beziehungen zur Base-Plug-in-API und realisierte Methoden des Subversion-Plug-ins . . . 130
6.2 Konfigurationsoptionen des Subversion-Plug-ins . . . 131
6.3 Ausschnitte aus der Web-Oberfläche zur Konfiguration von Subversion . . . 132
6.4 Ablauf der Erzeugung eines Subversion-Repositories . . . 132
6.5 Ablauf der Aktualisierung eines Subversion-Repositories . . . 133
6.6 Oberklasse und realisierte Methoden des Trac-Plug-ins . . . 135
6.7 Ablauf der Erzeugung einer Trac-Instanz . . . 135
6.8 Ablauf der Aktualisierung einer Trac-Instanz . . . 136
6.9 Sperrung des Zugriffs auf eine Trac-Instanz . . . 136
6.10 Oberklasse, realisierte Methoden und Hilfsklassen des Jenkins-Plug-ins . . . 138
6.11 Ablauf der Aktualisierung von Jenkins . . . 139
6.12 Architektur des Feedback-Systems . . . 140
6.13 Oberklasse, realisierte Methoden und Hilfsklassen des Feedback-Plug-ins . . . 141
6.14 Ablauf der Erzeugung der Webseite und der Clients des Feedback-Services . . . 142

6.15 Laufzeitkomponenten von MontiCore (adaptiert aus [Kra10]) 143
6.16 Oberklassen und realisierte Methoden des MontiCore-Plug-ins sowie einige MontiCore-Klassen 144
6.17 Parameter des MontiCore-Plug-ins 144
6.19 Implementierte Methoden der WikiBot-Plug-ins 146
6.20 Parameter der WikiBot-Plug-ins 147
6.22 Komponenten des Clients der WikiBot-Plug-ins 148
6.24 Oberklassen und realisierte Methoden des Google-Plug-ins 149
6.26 Schritte zur Integration von Anwendungen als Base-Services 152
6.27 Schritte zur Integration von Werkzeugen als Ostp-Services 154
6.28 Schritte zur Integration von Funktionen sozialer Netzwerke als Social-Services 156

7.1 Verteilungsdiagramm mit einer Frontend- und drei Backend-Instanzen 160
7.2 Gemeinsames Hosting der Daten und der Administrationsoberfläche einer SSELab-Instanz 161
7.3 Getrenntes Hosting der Daten und der Administrationsoberfläche einer SSELab-Instanz 162
7.4 Laufzeitumgebung eines Base-Backends mit installierten Service-Plug-ins und den zugehörigen Anwendungen 165
7.5 Laufzeitumgebung einer Frontend-Instanz 166
7.6 Methodik des Deployments von Backend-Instanzen 169
7.7 Methodik des Deployments von Frontend-Instanzen 171
7.8 Methodik des Deployments von Services 173
7.9 Methodik des Deployments von Clients 174

8.1 Anzahl der Benutzerkonten und der aktivierten Projekte (Stand: 07.12.2012) . . 177
8.2 Statistiken zu Projekten, Services und Benutzern (Stand: 13.02.2013) 178
8.3 Nutzung von Services der Instanz in Projekten (Stand: 13.02.2013) 178
8.4 Startseite der sselab.de-Instanz 181
8.5 Ausschnitt der Webseite zur Erstellung eines neuen Benutzerkontos 182
8.6 Überblicksseite nach erfolgreichem Login 183
8.7 Ausschnitt der Webseiten zur Beantragung eines neuen Projekts 184
8.8 Projektseite des neuen Projekts 184
8.9 Seite zur Verwaltung der Manager eines Projekts 185
8.10 Seite zur Verwaltung der Base-Services eines Projekts 186
8.11 Startseite nach dem Login eines Administrators 187
8.12 Verteilung der Komponenten der sselab.de-Instanz 190

Quellcodeverzeichnis

5.11 Aufbau des Pfads von Service-URLs 104

6.18 Implementierung der `transform()`-Methode im MontiCore-Plugin 145
6.21 Implementierung der `extract()`-Methode im WikiBot-Extractor-Plug-in . . 147
6.23 Ausgabe eines Hilfetextes nach Aufruf des WikiBot-Clients ohne Parameter . . 148
6.25 Implementierung der `createOAuth3LeggedScheme()`-Methode im Google-Plug-in 149

Tabellenverzeichnis

C.1 Kennzahlen der Komponenten des SSELab-Frameworks 242
C.2 Kennzahlen der Service-Plug-ins 243

Anhang A

Glossar

Im Folgenden wird die Verwendung einiger wichtiger Begriffe und Abkürzungen dieser Arbeit definiert.

Administrator
: Ein Administrator ist bei einer →SSELab-Instanz authentifiziert und hat administrative Rechte im gesamten System. Er kann alle →Benutzer, →Projekte sowie die angebotenen →Services über die Web-Oberfläche verwalten.

Backend
: Die Backends sind für die Konfiguration bzw. für die Ausführung der →Services verantwortlich. Im Rahmen dieser Arbeit wurden drei unterschiedliche Backend-Arten realisiert: Das Base-Backend ermöglicht die Konfiguration der →Base-Services für →Projekte. Das Ostp-Backend bietet einen einheitlichen Zugriff auf die →Ostp-Services und das Social-Backend ermöglicht die Nutzung von Funktionen sozialer Netzwerke über →Social-Services.

Base-Service
: Base-Services sind server- bzw. webbasierte Anwendungen, die sowohl die Kerninfrastruktur von Software Engineering Projekten als auch von anderen Projektarten bilden.

Benutzer
: Ein Benutzer ist bei einer →SSELab-Instanz authentifiziert und kann auf seine →Projekte und auf die →Services der Instanz zugreifen. Im Kontext eines Projekts besitzt ein Benutzer eine der beiden Rollen →Manager oder →Member.

Bot-Account
: Im Kontext von →Projekten können Bot-Accounts angelegt und dann für den automatisierten Zugriff auf die →Services des Projekts genutzt werden.

CDE
: →Collaborative Development Environment

Client
: Unter dem Begriff Client werden in dieser Arbeit sowohl die →Framework-Clients des →SSELabs als auch existierende →Service-Clients zusammengefasst.

Client-Entwickler

Ein Client-Entwickler realisiert auf Grundlage der Client-APIs des →SSELab-Frameworks einen oder mehrere Clients, mit deren Hilfe Funktionen des →Frontends oder einzelne →Services aufgerufen werden können.

Collaborative Development Environment

Eine Entwicklungsumgebung zur Kollaboration (engl. collaborative development environment, CDE) enthält primär webbasierte →Werkzeuge, die die Produktivität von Entwicklerteams verbessern. Eine CDE ist eine Entwicklungsumgebung, in der alle Projektbeteiligten gemeinsam arbeiten können und durch die viele alltägliche, nicht-kreative Aufgaben eliminiert bzw. automatisiert sowie die Kommunikation innerhalb eines Teams angeregt werden.

Deployment

Deployment ist ein Prozess, der sowohl nach der Entwicklung und der Auslieferung als auch nach der Akquise eines Softwareprodukts durch den Besitzer selbst durchgeführt wird. Die Aktivitäten, die nach der Installation ausgeführt werden, beispielsweise die Aktualisierung, Überwachung und Deinstallation, werden auch als Deployment-Aktivitäten bezeichnet.

Entwicklungsumgebung

In Entwicklungsumgebungen werden →Werkzeuge integriert, um die Produktivität einzelner Entwickler oder ganzer Entwicklerteams zu verbessern. Im ersten Fall spricht man von einer IDE (→Integrated Development Environment) und im zweiten von einer CDE (→Collaborative Development Environment) oder auch von einem →Projektportal.

Framework

Ein Framework ist ein wiederverwendbares abstraktes Design, das eine bewährte bzw. allgemeine Lösung für eine Menge von Problemen in einer bestimmten Domäne beschreibt. Ein Framework ist demnach selbst keine ausführbare Anwendung, sondern gibt die Architektur und die Eigenschaften von Anwendungen in der Domäne vor.

Framework-Client

Zur Abgrenzung von den →Service-Clients werden diese Clients, die direkt mit einer →SSELab-Instanz kommunizieren und ein Bestandteil des →SSELab-Frameworks sind, als Framework-Clients bezeichnet.

Framework-Entwickler

Ein Framework-Entwickler ist mit der inneren Architektur des →SSELab-Frameworks vertraut und kann neue Funktionen innerhalb des Frameworks realisieren.

Frontend

Das Frontend ist die zentrale webbasierte Administrationskomponente des →SSELabs, in der alle relevanten Daten der →Services, →Benutzer und →Projekte sowie aller →Backends und der →Clients verwaltet werden. Es ermöglicht einen direkten Zugriff per Browser

und bietet mehrere Web-Service-Schnittstellen an, über die →Framework-Clients mit dem Frontend kommunizieren können.

Gast

Ein Gast ist nicht bei einer →SSELab-Instanz authentifiziert und kann daher nur deren öffentlich zugängliche Seiten nutzen.

Host-Anwendung

Eine Host-Anwendung ist Bestandteil einer →Plug-in-Architektur und gibt sowohl den Kontext als auch die Anforderungen an integrierbare →Plug-ins durch mehrere Schnittstellen vor.

Hosted Service

Als Hosted Service wird eine Anwendung bezeichnet, die auf einer Serverinfrastruktur und nicht auf dem Rechner eines Benutzers installiert ist. Im Gegensatz zu dem Begriff →Software as a Service wird dabei keine Aussage zum Verhältnis des Anbieters und des Nutzers der Software gemacht, so dass der Betreiber auch gleichzeitig Nutzer sein kann.

IDE

→Integrated Development Environment

Integrated Development Environment

Wenn Werkzeuge, die die Produktivität einzelner Entwickler verbessern, in eine Umgebung integriert werden, spricht man von einer integrierten Entwicklungsumgebung (engl. integrated development environment, IDE).

Klassische Plug-in-Architektur

In klassischen Plug-in-Architekturen erweitern die →Plug-ins eine →Host-Anwendung um bestimmte Funktionalität über wohldefinierte Schnittstellen.

Komponente

Eine Komponente ist eine Kompositionseinheit, die ihre Abhängigkeiten und Schnittstellen explizit angibt und die durch ein unabhängiges →Deployment durch Dritte mit anderen Komponenten kombiniert werden kann.

Konto

Unter dem Begriff Konten werden sowohl →Bot-Accounts als auch →personengebundene Konten zusammengefasst.

Manager

Ein Manager ist ein Teilnehmer eines →Projekts und kann administrative Aufgaben, beispielsweise die Aktivierung oder Deaktivierung von →Services, durchführen.

Member

Ein Member ist ein Teilnehmer eines →Projekts und kann im Gegensatz zu einem →Manager nur die →Services des Projekts nutzen und keine administrativen Aufgaben ausführen.

Ostp-Service
: Dateiverarbeitende Transformationswerkzeuge werden als Ostp-Services bezeichnet. Beispiele dafür sind Konvertierungswerkzeuge oder Codegeneratoren, die eine Menge von Eingabedateien verarbeiten und eine Menge von Ausgabedateien produzieren.

Personengebundene Konten
: Unter dem Begriff personengebundene Konten werden sowohl →Benutzer als auch →Administratoren zusammengefasst.

Plug-in
: Plug-ins können eine →Host-Anwendungen um bestimmte Funktionalität ergänzen, die von der Anwendung selbst nicht bereitgestellt, jedoch über entsprechende Schnittstellen vorgesehen wird. Plug-ins können insbesondere zur Laufzeit der Host-Anwendung geladen und ausgeführt werden.

Plug-in-Architektur
: Unter dem Begriff Plug-in-Architekturen werden sowohl →klassische Plug-in-Architekturen als auch →reine Plug-in-Architekturen zusammengefasst.

Projekt
: Im Kontext des →SSELabs wird durch ein Projekt eine Menge von →Konten und →Services zusammengefasst. Innerhalb eines Projekts besitzt ein →Benutzer entweder die Rolle →Manager oder →Member.

Projektportal
: →Collaborative Development Environment

Reine Plug-in-Architektur
: In einer reinen Plug-in-Architektur wird die gesamte Funktionalität einer Anwendung durch →Plug-ins realisiert. Die →Host-Anwendung ist nur eine minimale Laufzeitumgebung ohne Endnutzerfunktionalität.

SaaS
: →Software as a Service

Service
: Als Services werden →Werkzeuge oder einzelne Funktionen von Werkzeugen bezeichnet, die in das →SSELab integriert und darüber genutzt werden können.

Service-Client
: Service-Clients interagieren direkt mit einem in das →SSELab-Framework integrierten →Werkzeug.

Service-Entwickler
: Ein Service-Entwickler kennt die Erweiterungsmechanismen und APIs des →SSELab-Frameworks und nutzt diese, um →Werkzeuge als →Services zu integrieren.

Service-Plug-in
Durch ein Service-Plug-in wird jeweils ein spezifisches →Werkzeug als →Service in das →SSELab-Framework integriert.

Social-Service
Funktionen sozialer Netzwerke oder anderer webbasierter Projektportale werden als Social-Services bezeichnet. Diese →Services können über eine →SSELab-Instanz genutzt werden, um Profilinformationen von →Benutzern zu importieren und zu verarbeiten.

Social Software
Unter dem Begriff Social Software werden Anwendungen zusammengefasst, die die Interaktion und Kommunikation deren Benutzer ermöglichen.

Software as a Service
Software as a Service (SaaS) beschreibt ein Nutzungs- bzw. ein Auslieferungskonzept von Software, bei dem ein Anbieter dedizierte Anwendungen auf einer Serverinfrastruktur als Dienstleistung bereitstellt und dessen Kunden über das Internet darauf zugreifen. Durch dieses Konzept wird der Besitz von der Nutzung der Software getrennt. Im Gegensatz zu dem Begriff →Hosted Service steht hier auch der Dienstleistungscharakter im Vordergrund.

SSELab
Der Begriff SSELab kombiniert die Aspekte der Begriffe →SSELab-Framework und →SSELab-Instanz.

SSELab-Framework
Das SSELab ist ein erweiterbares Framework, das aus mehreren Komponenten besteht: dem zentralen →Frontend, mehreren →Backends für die Service-Kategorien und Client-APIs für verschiedene Benutzergruppen.

SSELab-Instanz
Die Bildung einer SSELab-Instanz erfolgt durch die Konfiguration und die Instanziierung der Komponenten des →SSELab-Frameworks und deren Installation in der zugehörigen Laufzeitumgebung auf einer Serverinfrastruktur.

System-Administrator
Ein System-Administrator hat Zugang zu den Servermaschinen einer →SSELab-Instanz auf Betriebssystemebene und ist verantwortlich für die Verwaltung und Konfiguration der gesamten Serverinfrastruktur.

Werkzeug
Werkzeuge werden im Software Engineering eingesetzt, um die Komplexitätssteigerung der Systeme zu beherrschen, effektive Kollaboration zu ermöglichen und die Entwickler in ihren vielfältigen Aktivitäten zu unterstützen. Für die meisten Tätigkeiten in der Softwareentwicklung wurden diverse Werkzeuge entwickelt. Dazu gehören sowohl einzelne Werkzeuge wie Compiler, Generatoren und Editoren als auch integrierte oder webbasierte

→Entwicklungsumgebungen und serverbasierte Werkzeuge wie Continuous Integration Server.

Werkzeugintegration

Die Werkzeugintegration beschreibt sowohl die Gemeinsamkeiten als auch die Beziehungen von →Werkzeugen. Dies bezieht sich unter anderem auf die genutzten Datenformate, auf Konventionen für Benutzeroberflächen und auf die Verwendung gemeinsamer Funktionen.

Anhang B

Notationen

In den Abbildungen dieser Arbeit werden vorwiegend die Diagrammtypen der UML verwendet. Der konkrete Typ einer Abbildung wird jeweils durch eine Markierung spezifiziert. Im Folgenden wird ein Überblick über alle verwendeten Markierungen gegeben.

Aktivitätsdiagramm

Aktivitätsdiagramme [OMG10a, OMG10b] werden in dieser Arbeit zur Illustration von Methodiken, also von teilweise automatisierten Handlungsanweisungen verwendet. Darüber hinaus werden sie zur Beschreibung von Abläufen, beispielsweise der Installation und Konfiguration eines Services, eingesetzt.

Klassendiagramm

Klassendiagramme [Rum11] werden in dieser Arbeit zur Veranschaulichung struktureller Zusammenhänge eingesetzt. Durch sie werden die Eigenschaften, Methoden und Beziehungen der Domänenklassen und APIs des SSELab-Frameworks auf verschiedenen Abstraktionsebenen sowie die Architektur und Implementierung einzelner Komponenten beschrieben.

Komponentendiagramm

Komponentendiagramme [OMG10a, OMG10b] werden zur Beschreibung von Architekturen eingesetzt. Sowohl die Architektur des gesamten SSELab-Frameworks als auch der einzelnen Komponenten werden mit Hilfe von Komponentendiagrammen illustriert.

Verteilungsdiagramm

Verteilungsdiagramme [OMG10a, OMG10b] werden zur Veranschaulichung der Verteilung von Komponenten genutzt. Dabei wird beispielsweise die Verteilung einer SSELab-Instanz auf mehrere Server-Maschinen sowie der verwendeten Laufzeitumgebung beschrieben.

Objektdiagramm

Objektdiagramme [OMG10a, OMG10b] beschreiben konkrete Zustände eines Systems und werden in dieser Arbeit zur Darstellung von Objektstrukturen im Kontext der zugehörigen Klassendiagramme verwendet.

Sequenzdiagramm

Sequenzdiagramme [OMG10a, OMG10b] werden zur Illustration exemplarischer Abläufe eingesetzt. Ein Beispiel dafür ist der Austausch von Nachrichten bzw. Web-Service-Aufrufen zwischen einer Frontend- und mehreren Backend-Instanzen.

Anhang C

Kennzahlen

Damit der Umfang der im Rahmen dieser Arbeit realisierten Komponenten des Frameworks sowie der realisierten Service-Plug-ins eingeschätzt werden kann, werden im Folgenden einige Kennzahlen angegeben. Die Kennzahlen wurden auf Grundlage einer aktuellen Version aller Komponenten mit Hilfe von Werkzeugen zur statischen Codeanalyse gemessen.

Kennzahlen der Framework-Realisierung

In diesem Abschnitt werden einige Kennzahlen angegeben, durch die der Umfang realisierten Komponenten des SSELab-Frameworks eingeschätzt werden kann. In Tabelle C.1 ist für jede Komponente die Anzahl der Typen, also die Anzahl der Klassen und der Interfaces, angegeben sowie deren Source Lines of Code (SLOC). Dies sind die Zeilen der zugehörigen Dateien, die nicht leer und kein Kommentar sind. Bei der Ermittlung der Kennzahlen wurden nur der Produktivcode und keine Tests berücksichtigt. Die letzten beiden Spalten der Tabelle geben die Anzahl der XHTML-Dateien sowie deren Source Lines of Code an. Diese Kennzahlen beschreiben den Umfang der Web-Oberfläche und sind daher nur für die Realisierung der Komponente SSELab-Web relevant.

Anhand der gezeigten Kennzahlen wird unter anderem die Mischung aus Anwendungs- und Framework-Entwicklung deutlich. Die umfangreichste Komponente ist SSELab-Web, durch die die Web-Oberfläche realisiert wird. Diese Komponente wie auch die Komponenten SSELab-Domain und SSELab-Core ermöglichen es, dass eine Instanz des Frameworks nach dem Deployment unmittelbar produktiv genutzt werden kann. Im Gegensatz dazu enthalten die realisierten Backends des Frameworks eine deutlich geringere Menge an Code, was auf deren ausgeprägteren Framework-Charakter zurückzuführen ist. Hervorzuheben ist außerdem auch der geringe Umfang der Plug-in-APIs, so dass die Einarbeitung von Service-Entwicklern in diese APIs effizient möglich ist.

Kennzahlen der realisierten Service-Plug-ins

In diesem Abschnitt werden einige Kennzahlen für die realisierten Service-Plug-ins angegeben, mit deren Hilfe deren Umfang eingeschätzt werden kann. Tabelle C.2 zeigt die Anzahl der Klassen und Schnittstellen jedes Service-Plug-ins sowie deren Source Lines of Code (SLOC), also die Anzahl der Zeilen die kein Kommentar und nicht leer sind. Bei der Messung dieser Kennzahlen wurden nur der Produktivcode und keine Tests berücksichtigt. Die letzten beiden

Framework-Komponente	Anzahl Typen	SLOC Typen	Anzahl XHTML-Dateien	SLOC XHTML-Dateien
SSELab-Domain	132	11345	0	0
SSELab-Util	22	1259	0	0
Common-Plug-in-API	3	95	0	0
Base-Plug-in-API	4	274	0	0
Base-Plug-in-System	7	1037	0	0
Ostp-Plug-in-API	7	138	0	0
Ostp-Plug-in-System	14	1595	0	0
Social-Plug-in-API	3	236	0	0
Social-Plug-in-System	7	843	0	0
SSELab-Web	143	12635	219	24882
SSELab-Core	104	10314	0	0
Base-Service-Client	34	957	0	0
Ostp-Service-Client	38	1179	0	0
Social-Service-Client	43	1142	0	0
User-Service-Client	39	1211	0	0
User-Client-Common	10	1129	0	0
Eclipse-Client	39	2541	0	0
Batch-Client	5	178	0	0
Shell-Client	26	1623	0	0
Admin-Service-Client	17	456	0	0
Admin-Client-Common	5	251	0	0
Admin-Client	1	220	0	0
Summe	703	50658	219	24882

Tabelle C.1: Kennzahlen der Komponenten des SSELab-Frameworks

Spalten der Tabelle geben darüber hinaus die Anzahl der erstellten Skripte und deren Lines of Code (LOC), also deren nichtleeren Zeilen, an. Bei der Ermittlung dieser Kennzahlen wurden nur Skripte berücksichtigt, die zur Installation und Konfiguration einer Anwendung sowie zur automatischen Konfiguration von deren Instanzen benötigt werden. Skripte zur Anpassung des Erscheinungsbilds wurden nicht mit aufgenommen, da sie einerseits einen stark variierenden Umfang besitzen und andererseits gemäß der Methodik zur Integration von Base-Services optional sind (vgl. Abschnitt 5.2.2).

Einerseits wird durch diese Kennzahlen deutlich, dass eine relative geringe Menge an Code für die Integration von Werkzeugen und Anwendungen als Service erzeugt werden muss. Andererseits zeigt die Tabelle, dass der Aufwand bei der Integration von Base-Services vergleichsweise höher als bei der Integration von Ostp- oder Social-Services ist. Dies ergibt sich dadurch, dass bei Base-Services zusätzlich zu der Anwendung auch die Laufzeitumgebung (Web-, Application-

Komponente	Anzahl Typen	SLOC Typen	Anzahl Skripte	LOC Skripte
Subversion-Plug-in	2	389	5	785
Git-Plug-in	2	340	6	724
Storage-Plug-in	1	75	1	5
Trac-Plug-in	1	258	4	259
Internal-MediaWiki-Plug-in	1	98	7	561
External-MediaWiki-Plug-in	1	115	9	762
Web-Space-Plug-in	1	95	2	15
Mailing-List-Plug-in	2	376	10	81
Feedback-System-Plug-in	1	163	1	18
Jenkins-Plug-in	4	694	5	208
Nexus-Plug-in	4	353	1	26
Wordpress-Plug-in	1	136	12	427
MontiCore-Plug-in	2	122	0	0
MontiArc-Plug-in	1	330	0	0
MontiWIS-Plug-in	1	71	0	0
XML-Schema-Generator-Plug-in	1	85	0	0
Wikibot-Extractor-Plug-in	1	70	0	0
Wikibot-Publisher-Plug-in	1	74	0	0
Wikibot-Connector	4	152	0	0
ppt2eps-Plug-in	3	308	0	0
Google-Plug-in	1	40	0	0
MySpace-Plug-in	1	40	0	0
LinkedIn-Plug-in	1	116	0	0
Summe	38	4500	63	3871

Tabelle C.2: Kennzahlen der Service-Plug-ins

oder Datenbankserver) installiert sowie die Konfiguration der Anwendung bei Änderungen eines Projekts aktualisiert werden muss.

Tabelle C.2 verdeutlicht weiterhin, dass bei der Integration einer Anwendung als Base-Services jedes mal entschieden werden muss, wie viel Code innerhalb des Service-Plug-ins realisiert werden kann und welche Anteile in Skripte ausgelagert werden sollten. Beispielsweise besitzen die Plug-ins und Skripte der Services External-MediaWiki und Jenkins in der Summe ungefähr den gleichen Umfang. Im Fall von External-MediaWiki wurde jedoch die Integration in großen Anteilen durch Skripte realisiert, da diese auf den APIs von MediaWiki aufbauen. Eine Realisierung der gleichen Funktionalität im Service-Plug-in wäre deutlich aufwendiger gewesen. Im Jenkins-Plug-in kann dagegen die REST-API des Servers effektiv aufgerufen werden, so dass nur die Installation und initiale Konfiguration von Jenkins durch Skripte realisiert ist.

Anhang D

Lebenslauf

Name	Herrmann
Vorname	Christoph
Geburtstag	05.08.1980
Geburtsort	Braunschweig
Staatsangehörigkeit	deutsch
ab 2012	Mitarbeiter bei der synavision GmbH
2009 - 2012	Wissenschaftlicher Mitarbeiter am Lehrstuhl Software Engineering der RWTH Aachen
2006 - 2009	Wissenschaftlicher Mitarbeiter am Institut für Software Systems Engineering der TU Braunschweig
2006	Abschluss als Diplom-Informatiker
2001 - 2006	Studium der Informatik an der TU Braunschweig
2000 - 2001	Zivildienst
2000	Abitur
1987 - 2000	Grundschule, Orientierungsstufe und Gymnasium

Related Interesting Work from the SE Group, RWTH Aachen

Agile Model Based Software Engineering

Agility and modeling in the same project? This question was raised in [Rum04]: "Using an executable, yet abstract and multi-view modeling language for modeling, designing and programming still allows to use an agile development process." Modeling will be used in development projects much more, if the benefits become evident early, e.g with executable UML [Rum02] and tests [Rum03]. In [GKRS06], for example, we concentrate on the integration of models and ordinary programming code. In [Rum12] and [Rum11], the UML/P, a variant of the UML especially designed for programming, refactoring and evolution, is defined. The language workbench MontiCore [GKR+06] is used to realize the UML/P [Sch12]. Links to further research, e.g., include a general discussion of how to manage and evolve models [LRSS10], a precise definition for model composition as well as model languages [HKR+09] and refactoring in various modeling and programming languages [PR03]. In [FHR08] we describe a set of general requirements for model quality. Finally [KRV06] discusses the additional roles and activities necessary in a DSL-based software development project.

Generative Software Engineering

The UML/P language family [Rum12, Rum11] is a simplified and semantically sound derivate of the UML designed for product and test code generation. [Sch12] describes a flexible generator for the UML/P based on the MontiCore language workbench [KRV10, GKR+06]. In [KRV06], we discuss additional roles necessary in a model-based software development project. In [GKRS06] we discuss mechanisms to keep generated and handwritten code separated. In [Wei12] we show how this looks like and how to systematically derive a transformation language in concrete syntax. To understand the implications of executability for UML, we discuss needs and advantages of executable modeling with UML in agile projects in [Rum04], how to apply UML for testing in [Rum03] and the advantages and perils of using modeling languages for programming in [Rum02].

Unified Modeling Language (UML)

Many of our contributions build on UML/P described in the two books [Rum11] and [Rum12] are implemented in [Sch12]. Semantic variation points of the UML are discussed in [GR11]. We discuss formal semantics for UML [BHP+98] and describe UML semantics using the "System Model" [BCGR09a], [BCGR09b], [BCR07b] and [BCR07a]. Semantic variation points have, e.g., been applied to define class diagram semantics [CGR08]. A precisely defined semantics for variations is applied, when checking variants of class diagrams [MRR11c] and objects diagrams [MRR11d] or the consistency of both kinds of diagrams [MRR11e]. We also apply these concepts to activity diagrams (ADs) [MRR11b] which allows us to check for semantic differences of activity diagrams [MRR11a]. We also discuss how to ensure and identify model quality [FHR08], how models, views and the system under development correlate to each other [BGH+98] and how to use modeling in agile development projects [Rum04], [Rum02] The question how to adapt and extend the UML in discussed in [PFR02] on product line annotations for UML and to more general discussions and insights on how to use meta-modeling for defining and adapting the UML [EFLR99], [SRVK10].

Domain Specific Languages (DSLs)

Computer science is about languages. Domain Specific Languages (DSLs) are better to use, but need appropriate tooling. The MontiCore language workbench [GKR+06], [KRV10], [Kra10] describes an integrated abstract and concrete syntax format [KRV07b] for easy development. New languages and tools

can be defined in modular forms [KRV08, Völ11] and can, thus, easily be reused. [Wei12] presents a tool that allows to create transformation rules tailored to an underlying DSL. Variability in DSL definitions has been examined in [GR11]. A successful application has been carried out in the Air Traffic Management domain [ZPK+11]. Based on the concepts described above, meta modeling, model analyses and model evolution have been examined in [LRSS10] and [SRVK10]. DSL quality [FHR08], instructions for defining views [GHK+07], guidelines to define DSLs [KKP+09] and Eclipse-based tooling for DSLs [KRV07a] complete the collection.

Modeling Software Architecture & the MontiArc Tool

Distributed interactive systems communicate via messages on a bus, discrete event signals, streams of telephone or video data, method invocation, or data structures passed between software services. We use streams, statemachines and components [BR07] as well as expressive forms of composition and refinement [PR99] for semantics. Furthermore, we built a concrete tooling infrastructure called MontiArc [HRR12] for architecture design and extensions for states [RRW13]. MontiArc was extended to describe variability [HRR+11] using deltas [HRRS11] and evolution on deltas [HRRS12]. [GHK+07] and [GHK+08] close the gap between the requirements and the logical architecture and [GKPR08] extends it to model variants. Co-evolution of architecture is discussed in [MMR10] and a modeling technique to describe dynamic architectures is shown in [HRR98].

Compositionality & Modularity of Models

[HKR+09] motivates the basic mechanisms for modularity and compositionality for modeling. The mechanisms for distributed systems are shown in [BR07] and algebraically underpinned in [HKR+07]. Semantic and methodical aspects of model composition [KRV08] led to the language workbench MontiCore [KRV10] that can even develop modeling tools in a compositional form. A set of DSL design guidelines incorporates reuse through this form of composition [KKP+09]. [Völ11] examines the composition of context conditions respectively the underlying infrastructure of the symbol table. Modular editor generation is discussed in [KRV07a].

Semantics of Modeling Languages

The meaning of semantics and its principles like underspecification, language precision and detailedness is discussed in [HR04]. We defined a semantic domain called “System Model” by using mathematical theory. [RKB95, BHP+98] and [GKR96, KRB96]. An extended version especially suited for the UML is given in [BCGR09b] and in [BCGR09a] its rationale is discussed. [BCR07a, BCR07b] contain detailed versions that are applied on class diagrams in [CGR08]. [MRR11a, MRR11b] encode a part of the semantics to handle semantic differences of activity diagrams and [MRR11e] compares class and object diagrams with regard to their semantics. In [BR07], a simplified mathematical model for distributed systems based on black-box behaviors of components is defined. Meta-modeling semantics is discussed in [EFLR99]. [BGH+97] discusses potential modeling languages for the description of an exemplary object interaction, today called sequence diagram. [BGH+98] discusses the relationships between a system, a view and a complete model in the context of the UML. [GR11] and [CGR09] discuss general requirements for a framework to describe semantic and syntactic variations of a modeling language. We apply these on class and object diagrams in [MRR11e] as well as activity diagrams in [GRR10]. [Rum12] embodies the semantics in a variety of code and test case generation, refactoring and evolution techniques. [LRSS10] discusses evolution and related issues in greater detail.

Evolution & Transformation of Models

Models are the central artifact in model driven development, but as code they are not initially correct and need to be changed, evolved and maintained over time. Model transformation is therefore essential to effectively deal with models. Many concrete model transformation problems are discussed: evolution [LRSS10, MMR10, Rum04], refinement [PR99, KPR97, PR94], refactoring [Rum12, PR03], translating models from one language into another [MRR11c, Rum12] and systematic model transformation language development [Wei12]. [Rum04] describes how comprehensible sets of such transformations support software development, maintenance and [LRSS10] technologies for evolving models within a language and across languages and linking architecture descriptions to their implementation [MMR10]. Automaton refinement is discussed in [PR94, KPR97], refining pipe-and-filter architectures is explained in [PR99]. Refactorings of models are important for model driven engineering as discussed in [PR03, Rum12]. Translation between languages, e.g., from class diagrams into Alloy [MRR11c] allows for comparing class diagrams on a semantic level.

Variability & Software Product Lines (SPL)

Many products exist in various variants, for example cars or mobile phones, where one manufacturer develops several products with many similarities but also many variations. Variants are managed in a Software Product Line (SPL) that captures the commonalities as well as the differences. Feature diagrams describe variability in a top down fashion, e.g., in the automotive domain [GHK+08] using 150% models. Reducing overhead and associated costs is discussed in [GRJA12]. Delta modeling is a bottom up technique starting with a small, but complete base variant. Features are added (that sometimes also modify the core). A set of applicable deltas configures a system variant. We discuss the application of this technique to Delta-MontiArc [HRR+11, HRR+11] and to Delta-Simulink [HKM+13]. Deltas can not only describe spacial variability but also temporal variability which allows for using them for software product line evolution [HRRS12]. [HHK+13] describes an approach to systematically derive delta languages. We also apply variability to modeling languages in order to describe syntactic and semantic variation points, e.g., in UML for frameworks [PFR02]. And we specified a systematic way to define variants of modeling languages [CGR09] and applied this as a semantic language refinement on Statecharts in [GR11].

Cyber-Physical Systems (CPS)

Cyber-Physical Systems (CPS) [KRS12] are complex, distributed systems which control physical entities. Contributions for individual aspects range from requirements [GRJA12], complete product lines [HRRW12], the improvement of engineering for distributed automotive systems [HRR12] and autonomous driving [BR12a] to processes and tools to improve the development as well as the product itself [BBR07]. In the aviation domain, a modeling language for uncertainty and safety events was developed, which is of interest for the European airspace [ZPK+11]. A component and connector architecture description language suitable for the specific challenges in robotics is discussed in [RRW13]. Monitoring for smart and energy efficient buildings is developed as Energy Navigator toolset [KPR12, FPPR12, KLPR12].

State Based Modeling (Automata)

Today, many computer science theories are based on state machines in various forms including Petri nets or temporal logics. Software engineering is particularly interested in using state machines for modeling systems. Our contributions to state based modeling can currently be split into three parts: (1) understanding how to model object-oriented and distributed software using statemachines resp. Statecharts

[GKR96, BCR07b, BCGR09b, BCGR09a], (2) understanding the refinement [PR94, RK96, Rum96] and composition [GR95] of statemachines, and (3) applying statemachines for modeling systems. In [Rum96] constructive transformation rules for refining automata behavior are given and proven correct. This theory is applied to features in [KPR97]. Statemachines are embedded in the composition and behavioral specifications concepts of Focus [BR07]. We apply these techniques, e.g., in MontiArcAutomaton [THR+13] as well as in building management systems [FLP+11].

Robotics

Robotics can be considered a special field within Cyber-Physical Systems which is defined by an inherent heterogeneity of involved domains, relevant platforms, and challenges. The engineering of robotics applications requires composition and interaction of diverse distributed software modules. This usually leads to complex monolithic software solutions hardly reusable, maintainable, and comprehensible, which hampers broad propagation of robotics applications. The MontiArcAutomaton language [RRW12] extends ADL MontiArc and integrates various implemented behavior modeling languages using MontiCore [RRW13] that perfectly fits Robotic architectural modelling. The LightRocks [THR+13] framework allows robotics experts and laymen to model robotic assembly tasks.

Automotive, Autonomic Driving & Intelligent Driver Assistance

Introducing and connecting sophisticated driver assistance, infotainment and communication systems as well as advanced active and passive safety-systems result in complex embedded systems. As these feature-driven subsystems may be arbitrarily combined by the customer, a huge amount of distinct variants needs to be managed, developed and tested. A consistent requirements management that connects requirements with features in all phases of the development for the automotive domain is described in [GRJA12]. The conceptual gap between requirements and the logical architecture of a car is closed in [GHK+07, GHK+08]. [HKM+13] describes a tool for delta modeling for Simulink [HKM+13]. [HRRW12] discusses means to extract a well-defined Software Product Line from a set of copy and paste variants. Quality assurance, especially of safety-related functions, is a highly important task. In the Carolo project [BR12a, BR12b], we developed a rigorous test infrastructure for intelligent, sensor-based functions through fully-automatic simulation [BBR07]. This technique allows a dramatic speedup in development and evolution of autonomous car functionality, and thus, enables us to develop software in an agile way [BR12a]. [MMR10] gives an overview of the current state-of-the-art in development and evolution on a more general level by considering any kind of critical system that relies on architectural descriptions. As tooling infrastructure, the SSElab storage, versioning and management services [HKR12] are essential for many projects.

Energy Management

In the past years, it became more and more evident that saving energy and reducing CO2 emissions is an important challenge. Thus, energy management in buildings as well as in neighbourhoods becomes equally important to efficiently use the generated energy. Within several research projects, we developed methodologies and solutions for integrating heterogeneous systems at different scales. During the design phase, the Energy Navigators Active Functional Specification (AFS) [FPPR12, KPR12] is used for technical specification of building services already. We adapted the well-known concept of statemachines to be able to describe different states of a facility and to validate it against the monitored values [FLP+11]. We show how our data model, the constraint rules and the evaluation approach to compare sensor data can be applied [KLPR12].

Cloud Computing & Enterprise Information Systems

The paradigm of Cloud Computing is arising out of a convergence of existing technologies for web-based application and service architectures with high complexity, criticality and new application domains. It promises to enable new business models, to lower the barrier for web-based innovations and to increase the efficiency and cost-effectiveness of web development. Application classes like Cyber-Physical Systems [KRS12], Big Data, App and Service Ecosystems bring attention to aspects like responsiveness, privacy and open platforms. Regardless of the application domain, developers of such systems are in need for robust methods and efficient, easy-to-use languages and tools. We tackle these challenges by perusing a model-based, generative approach [PR13]. The core of this approach are different modeling languages that describe different aspects of a cloud-based system in a concise and technology-agnostic way. Software architecture and infrastructure models describe the system and its physical distribution on a large scale. We apply cloud technology for the services we develop, e.g., the SSELab [HKR12] and the Energy Navigator [FPPR12, KPR12] but also for our tool demonstrators and our own development platforms. New services, e.g.,c collecting data from temperature, cars etc. are easily developed.

References

[BBR07] Christian Basarke, Christian Berger, and Bernhard Rumpe. Software & Systems Engineering Process and Tools for the Development of Autonomous Driving Intelligence. *Journal of Aerospace Computing, Information, and Communication (JACIC)*, 4(12):1158–1174, October 2007.

[BCGR09a] Manfred Broy, Maria Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Considerations and Rationale for a UML System Model. In Kevin Lano, editor, *UML 2 Semantics and Applications*, pages 43–61. John Wiley & Sons, 2009.

[BCGR09b] Manfred Broy, Maria Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Definition of the UML System Model. In Kevin Lano, editor, *UML 2 Semantics and Applications*, pages 63–93. John Wiley & Sons, 2009.

[BCR07a] Manfred Broy, Maria Victoria Cengarle, and Bernhard Rumpe. Towards a System Model for UML. Part 2: The Control Model. Technical Report TUM-I0710, TU Munich, February 2007.

[BCR07b] Manfred Broy, Maria Victoria Cengarle, and Bernhard Rumpe. Towards a System Model for UML. Part 3: The State Machine Model. Technical Report TUM-I0711, TU Munich, February 2007.

[BGH+97] Ruth Breu, Radu Grosu, Christoph Hofmann, Franz Huber, Ingolf Krüger, Bernhard Rumpe, Monika Schmidt, and Wolfgang Schwerin. Exemplary and Complete Object Interaction Descriptions. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Proceedings OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, TUM-I9737, TU Munich, 1997.

[BGH+98] Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Systems, Views and Models of UML. In M. Schader and A. Korthaus, editors, *Proceedings of the Unified Modeling Language, Technical Aspects and Applications*. Physica Verlag, Heidelberg, 1998.

[BHP+98] Manfred Broy, Franz Huber, Barbara Paech, Bernhard Rumpe, and Katharina Spies. Software and System Modeling Based on a Unified Formal Semantics. In M. Broy and B. Rumpe, editors, *RTSE '97: Proceedings of the International Workshop on Requirements Targeting Software and Systems Engineering*, LNCS 1526, pages 43–68, Bernried, Germany, October 1998. Springer.

[BR07] Manfred Broy and Bernhard Rumpe. Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. *Informatik-Spektrum*, 30(1):3–18, Februar 2007.

[BR12a] Christian Berger and Bernhard Rumpe. Autonomous Driving - 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System. In *Proceedings of the 10th Workshop on Automotive Software Engineering (ASE 2012)*, pages 789–798, Braunschweig, Germany, September 2012.

[BR12b] Christian Berger and Bernhard Rumpe. Engineering Autonomous Driving Software. In C. Rouff and M. Hinchey, editors, *Experience from the DARPA Urban Challenge*. Springer, 2012.

[CGR08] María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. System Model Semantics of Class Diagrams. Informatik-Bericht 2008-05, CfG Fakultät, TU Braunschweig, 2008.

[CGR09] María Victoria Cengarle, Hans Grönniger, and Bernhard Rumpe. Variability within Modeling Language Definitions. In *Model Driven Engineering Languages and Systems. Proceedings of MODELS 2009*, LNCS 5795, pages 670–684, Denver, Colorado, USA, October 2009.

[EFLR99] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Meta-Modelling Semantics of UML. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*. Kluver Academic Publisher, 1999.

[FHR08] Florian Fieber, Michaela Huhn, and Bernhard Rumpe. Modellqualität als Indikator für Softwarequalität: eine Taxonomie. *Informatik-Spektrum*, 31(5):408–424, Oktober 2008.

[FLP+11] Norbert Fisch, Markus Look, Claas Pinkernell, Stefan Plesser, and Bernhard Rumpe. State-Based Modeling of Buildings and Facilities. In *Proceedings of the 11th International Conference for Enhanced Building Operations (ICEBO' 11)*, New York City, USA, October 2011.

[FPPR12] Norbert Fisch, Claas Pinkernell, Stefan Plesser, and Bernhard Rumpe. The Energy Navigator - A Web-Platform for Performance Design and Management. In *Proceedings of the 7th International Conference on Energy Efficiency in Commercial Buildings (IEECB)*, Frankfurt a. M., Germany, April 2012.

[GHK+07] Hans Grönniger, Jochen Hartmann, Holger Krahn, Stefan Kriebel, and Bernhard Rumpe. View-based Modeling of Function Nets. In *Proceedings of the Object-oriented Modelling of Embedded Real-Time Systems (OMER4) Workshop*, Paderborn, Germany, October 2007.

[GHK+08] Hans Grönniger, Jochen Hartmann, Holger Krahn, Stefan Kriebel, Lutz Rothhardt, and Bernhard Rumpe. Modelling Automotive Function Nets with Views for Features, Variants, and Modes. In *Proceedings of 4th European Congress ERTS - Embedded Real Time Software*, Toulouse, 2008.

[GKPR08] Hans Grönniger, Holger Krahn, Claas Pinkernell, and Bernhard Rumpe. Modeling Variants of Automotive Systems using Views. In *Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen (MBEFF)*, Informatik Bericht 2008-01, pages 76–89, CFG Fakultät, TU Braunschweig, March 2008.

[GKR96] Radu Grosu, Cornel Klein, and Bernhard Rumpe. Enhancing the SysLab System Model with State. Technical Report TUM-I9631, TUM, Munich, Germany, 1996.

[GKR+06] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. MontiCore 1.0 - Ein Framework zur Erstellung und Verarbeitung domänspezifischer Sprachen. Technical Report 2006-04, CfG Fakultät, TU Braunschweig, August 2006.

[GKRS06] Hans Grönniger, Holger Krahn, Bernhard Rumpe, and Martin Schindler. Integration von Modellen in einen codebasierten Softwareentwicklungsprozess. In *Proceedings der Modellierung 2006*, Lecture Notes in Informatics LNI P-82, Innsbruck, März 2006. GI-Edition.

[GR95] Radu Grosu and Bernhard Rumpe. Concurrent Timed Port Automata. Technical Report TUM-I9533, TUM, Munich, Germany, 1995.

[GR11] Hans Grönniger and Bernhard Rumpe. Modeling Language Variability. In *Workshop on Modeling, Development and Verification of Adaptive Systems. 16th Monterey Workshop*, LNCS 6662, pages 17–32, Redmond, Microsoft Research, 2011. Springer.

[GRJA12] Tim Gülke, Bernhard Rumpe, Martin Jansen, and Joachim Axmann. High-Level Requirements Management and Complexity Costs in Automotive Development Projects: A Problem Statement. In *Requirements Engineering: Foundation for Software Quality. 18th International Working Conference, Proceedings, REFSQ 2012*, Essen, Germany, March 2012.

[GRR10] Hans Grönniger, Dirk Reiß, and Bernhard Rumpe. Towards a Semantics of Activity Diagrams with Semantic Variation Points. In *Model Driven Engineering Languages and Systems, Proceedings of MODELS*, LNCS 6394, Oslo, Norway, 2010. Springer.

[HHK+13] Arne Haber, Katrin Hölldobler, Carsten Kolassa, Markus Look, Klaus Müller, Bernhard Rumpe, and Ina Schaefer. Engineering Delta Modeling Languages. In *Proceedings of the 17th International Software Product Line Conference (SPLC), Tokyo*, pages 22–31. ACM, September 2013.

[HKM+13] Arne Haber, Carsten Kolassa, Peter Manhart, Pedram Mir Seyed Nazari, Bernhard Rumpe, and Ina Schaefer. First-Class Variability Modeling in Matlab / Simulink. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, pages 11–18, New York, NY, USA, 2013. ACM.

[HKR+07] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. An Algebraic View on the Semantics of Model Composition. In D. H. Akehurst, R. Vogel, and R. F. Paige, editors, *Proceedings of the Third European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2007), Haifa, Israel*, pages 99–113. Springer, 2007.

[HKR+09] Christoph Herrmann, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Scaling-Up Model-Based-Development for Large Heterogeneous Systems with Compositional Modeling. In H. Arabnia and H. Reza, editors, *Proceedings of the 2009 International Conference on Software Engineeering in Research and Practice*, Las Vegas, Nevada, USA, 2009.

[HKR12] Christoph Herrmann, Thomas Kurpick, and Bernhard Rumpe. SSELab: A Plug-In-Based Framework for Web-Based Project Portals. In *Proceedings of the 2nd International Workshop on Developing Tools as Plug-Ins (TOPI) at ICSE 2012*, pages 61–66, Zurich, Switzerland, June 2012. IEEE.

[HR04] David Harel and Bernhard Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"? *IEEE Computer*, 37(10):64–72, Oct 2004.

[HRR98] Franz Huber, Andreas Rausch, and Bernhard Rumpe. Modeling Dynamic Component Interfaces. In Madhu Singh, Bertrand Meyer, Joseph Gil, and Richard Mitchell, editors, *TOOLS 26, Technology of Object-Oriented Languages and Systems*. IEEE Computer Society, 1998.

[HRR+11] Arne Haber, Holger Rendel, Bernhard Rumpe, Ina Schaefer, and Frank van der Linden. Hierarchical Variability Modeling for Software Architectures. In *Proceedings of International Software Product Lines Conference (SPLC 2011)*. IEEE Computer Society, August 2011.

[HRR12] Arne Haber, Jan Oliver Ringert, and Bernhard Rumpe. MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical Report AIB-2012-03, RWTH Aachen, February 2012.

[HRRS11] Arne Haber, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. Delta Modeling for Software Architectures. *Tagungsband des Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII, fortiss GmbH*, February 2011.

[HRRS12] Arne Haber, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. Evolving Delta-oriented Software Product Line Architectures. In *Large-Scale Complex IT Systems. Development, Operation and Management, 17th Monterey Workshop 2012*, LNCS 7539, pages 183–208, Oxford, UK, March 2012. Springer.

[HRRW12] Christian Hopp, Holger Rendel, Bernhard Rumpe, and Fabian Wolf. Einführung eines Produktlinienansatzes in die automotive Softwareentwicklung am Beispiel von Steuergerätesoftware. In *Software Engineering 2012: Fachtagung des GI-Fachbereichs Softwaretechnik in Berlin*, Lecture Notes in Informatics LNI 198, pages 181–192, 27. Februar - 2. März 2012.

[KKP+09] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design Guidelines for Domain Specific Languages. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09), Sprinkle, J., Gray, J., Rossi, M., Tolvanen, J.-P., (eds.)*, Techreport B-108, Helsinki School of Economics, Orlando, Florida, USA, October 2009.

[KLPR12] Thomas Kurpick, Markus Look, Claas Pinkernell, and Bernhard Rumpe. Modeling Cyber-Physical Systems: Model-Driven Specification of Energy Efficient Buildings. In *Proceedings of the Modelling of the Physical World Workshop MOTPW'12, Innsbruck, October 2012*, pages 2:1–2:6. ACM Digital Library, October 2012.

[KPR97] Cornel Klein, Christian Prehofer, and Bernhard Rumpe. Feature Specification and Refinement with State Transition Diagrams. In *Fourth IEEE Workshop on Feature Interactions in Telecommunications Networks and Distributed Systems*. P. Dini, IOS-Press, 1997.

[KPR12] Thomas Kurpick, Claas Pinkernell, and Bernhard Rumpe. Der Energie Navigator. In *Entwicklung und Evolution von Forschungssoftware. Tagungsband, Rolduc, 10.-11.11.2011*, Aachener Informatik-Berichte, Software Engineering Band 14. Shaker Verlag Aachen, 2012.

[Kra10] Holger Krahn. *MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering*. Aachener Informatik-Berichte, Software Engineering Band 1. Shaker Verlag, Aachen, Germany, 2010.

[KRB96] Cornel Klein, Bernhard Rumpe, and Manfred Broy. A stream-based mathematical model for distributed information processing systems - SysLab system model. In *Proceedings of the first International Workshop on Formal Methods for Open Object-based Distributed Systems*, pages 323–338. Chapmann & Hall, 1996.

[KRS12] Stefan Kowalewski, Bernhard Rumpe, and Andre Stollenwerk. Cyber-Physical Systems - eine Herausforderung für die Automatisierungstechnik? In *Proceedings of Automation 2012, VDI Berichte 2012*, pages 113–116. VDI Verlag, 2012.

[KRV06] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Roles in Software Development using Domain Specific Modelling Languages. In J. Gray, J.-P. Tolvanen, and J. Sprinkle, editors, *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling 2006 (DSM'06), Portland, Oregon USA*, Technical Report TR-37, pages 150–158, Jyväskylä University, Finland, 2006.

[KRV07a] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Efficient Editor Generation for Compositional DSLs in Eclipse. In *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling (DSM' 07), Montreal, Quebec, Canada*, Technical Report TR-38, pages 8–10, Jyväskylä University, Finland, 2007.

[KRV07b] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Integrated Definition of Abstract and Concrete Syntax for Textual Languages. In G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, editors, *Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MODELS 2007), Nashville, TN, USA, October 2007*, LNCS 4735. Springer, 2007.

[KRV08] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Monticore: Modular Development of Textual Domain Specific Languages. In R. F. Paige and B. Meyer, editors, *Proceedings of the 46th International Conference Objects, Models, Components, Patterns (TOOLS-Europe), Zurich, Switzerland, 2008*, Lecture Notes in Business Information Processing LNBIP 11, pages 297–315. Springer, 2008.

[KRV10] Holger Krahn, Bernhard Rumpe, and Stefen Völkel. MontiCore: a Framework for Compositional Development of Domain Specific Languages. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(5):353–372, September 2010.

[LRSS10] Tihamer Levendovszky, Bernhard Rumpe, Bernhard Schätz, and Jonathan Sprinkle. Model Evolution and Management. In *MBEERTS: Model-Based Engineering of Embedded Real-Time Systems, International Dagstuhl Workshop, Dagstuhl Castle, Germany*, LNCS 6100, pages 241–270. Springer, October 2010.

[MMR10] Tom Mens, Jeff Magee, and Bernhard Rumpe. Evolving Software Architecture Descriptions of Critical Systems. *IEEE Computer*, 43(5):42–48, May 2010.

[MRR11a] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. ADDiff: Semantic Differencing for Activity Diagrams. In *Proc. Euro. Soft. Eng. Conf. and SIGSOFT Symp. on the Foundations of Soft. Eng. (ESEC/FSE'11)*, pages 179–189. ACM, 2011.

[MRR11b] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. An Operational Semantics for Activity Diagrams using SMV. Technical Report AIB-2011-07, RWTH Aachen University, Aachen, Germany, July 2011.

[MRR11c] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In *Model Driven Engineering Languages and Systems (MODELS 2011), Wellington, New Zealand*, LNCS 6981, pages 592–607, 2011.

[MRR11d] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Modal Object Diagrams. In *Proc. 25th Euro. Conf. on Object Oriented Programming (ECOOP'11)*, LNCS 6813, pages 281–305. Springer, 2011.

[MRR11e] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically Configurable Consistency Analysis for Class and Object Diagrams. In *Model Driven Engineering Languages and Systems (MODELS 2011), Wellington, New Zealand*, LNCS 6981, pages 153–167. Springer, 2011.

[PFR02] Wolfgang Pree, Marcus Fontoura, and Bernhard Rumpe. Product Line Annotations with UML-F. In G. J. Chastek, editor, *Software Product Lines - Second International Conference, SPLC 2*, LNCS 2379, pages 188–197, San Diego, 2002. Springer.

[PR94] Barbara Paech and Bernhard Rumpe. A new Concept of Refinement used for Behaviour Modelling with Automata. In M. Naftalin, T. Denvir, and M. Bertran, editors, *FME'94: Industrial Benefit of Formal Methods*, LNCS 873. Springer, October 1994.

[PR99] Jan Philipps and Bernhard Rumpe. Refinement of Pipe-and-Filter Architectures. In J. Davies J. M. Wing, J. Woodcock, editor, *FM'99 - Formal Methods, Proceedings of the World Congress on Formal Methods in the Development of Computing System*, LNCS 1708, pages 96–115. Springer, 1999.

[PR03] Jan Philipps and Bernhard Rumpe. Refactoring of Programs and Specifications. In H. Kilov and K. Baclawski, editors, *Practical foundations of business and system specifications*, pages 281–297. Kluwer Academic Publishers, 2003.

[PR13] Antonio Navarro Perez and Bernhard Rumpe. Modeling Cloud Architectures as Interactive Systems. In I. Ober, A. S. Gokhale, J. H. Hill, J. Bruel, M. Felderer, D. Lugato, and A. Dabholka, editors, *Proc. of the 2nd International Workshop on Model-Driven Engineering for High Performance and Cloud Computing. Co-located with MODELS 2013, Miami*, Sun SITE Central Europe Workshop Proceedings CEUR 1118, pages 15–24. CEUR-WS.org, 2013.

[RK96] Bernhard Rumpe and Cornel Klein. Automata Describing Object Behavior. In H. Kilov and W. Harvey, editors, *Specification of Behavioral Semantics in Object-Oriented Information Modeling*, pages 265–286. Kluwer Academic Publishers, 1996.

[RKB95] Bernhard Rumpe, Cornel Klein, and Manfred Broy. Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - Syslab-Systemmodell. Technical Report TUM-I9510, Technische Universität München, 1995.

[RRW12] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. A Requirements Modeling Language for the Component Behavior of Cyber Physical Robotics Systems. In N. Seyff and A. Koziolek, editors, *Modelling and Quality in Requirements Engineering: Essays Dedicated to Martin Glinz on the Occasion of His 60th Birthday*. Monsenstein und Vannerdat, Münster, 2012.

[RRW13] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. MontiArcAutomaton: Modeling Architecture and Behavior of Robotic Systems. In *Workshops and Tutorials Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA), May 6-10, 2013, Karlsruhe, Germany*, pages 10–12, 2013.

[Rum96] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme*. Herbert Utz Verlag Wissenschaft, ISBN 3-89675-149-2, 1996.

[Rum02] Bernhard Rumpe. Executable Modeling with UML - A Vision or a Nightmare? In T. Clark and J. Warmer, editors, *Issues & Trends of Information Technology Management in Contemporary Associations, Seattle*, pages 697–701. Idea Group Publishing, Hershey, London, 2002.

[Rum03] Bernhard Rumpe. Model-Based Testing of Object-Oriented Systems. In F. de Boer, M. Bonsangue, S. Graf, W.-P. de Roever, editor, *Formal Methods for Components and Objects*, LNCS 2852, pages 380–402. Springer, November 2003.

[Rum04] Bernhard Rumpe. Agile Modeling with the UML. In M. Wirsing, A. Knapp, and S. Balsamo, editors, *Radical Innovations of Software and Systems Engineering in the Future. 9th International Workshop, RISSEF 2002. Venice, Italy, October 2002*, LNCS 2941. Springer, October 2004.

[Rum11] Bernhard Rumpe. *Modellierung mit UML*. Springer, second edition, September 2011.

[Rum12] Bernhard Rumpe. *Agile Modellierung mit UML: Codegenerierung, Testfälle, Refactoring*. Springer, second edition, Juni 2012.

[Sch12] Martin Schindler. *Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P*. Aachener Informatik-Berichte, Software Engineering Band 11. Shaker Verlag, Aachen, Germany, 2012.

[SRVK10] Jonathan Sprinkle, Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai. Metamodelling: State of the Art and Research Challenges. In *MBEERTS: Model-Based Engineering of Embedded Real-Time Systems, International Dagstuhl Workshop, Dagstuhl Castle, Germany*, LNCS 6100, pages 57–76, October 2010.

[THR+13] Ulrike Thomas, Gerd Hirzinger, Bernhard Rumpe, Christoph Schulze, and Andreas Wortmann. A New Skill Based Robot Programming Language Using UML/P Statecharts. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 461–466, Karlsruhe, Germany, May 2013. IEEE.

[Völ11] Steven Völkel. *Kompositionale Entwicklung domänenspezifischer Sprachen*. Aachener Informatik-Berichte, Software Engineering Band 9. Shaker Verlag, Aachen, Germany, 2011.

[Wei12] Ingo Weisemöller. *Generierung domänenspezifischer Transformationssprachen*. Aachener Informatik-Berichte, Software Engineering Band 12. Shaker Verlag, Aachen, Germany, 2012.

[ZPK+11] Massimiliano Zanin, David Perez, Dimitrios S Kolovos, Richard F Paige, Kumardev Chatterjee, Andreas Horst, and Bernhard Rumpe. On Demand Data Analysis and Filtering for Inaccurate Flight Trajectories. In D. Schaefer, editor, *Proceedings of the SESAR Innovation Days*. EUROCONTROL, November 2011.