Open Source für die Software-Entwicklung

Autoren: Petra Becker-Pechau, Stefan Roock, Joachim Sauer

Professionelle Software-Entwicklungs-Projekte setzen verstärkt Open-Source-Software ein. Durch ihre spezifischen Eigenschaften bietet Open-Source-Software verschiedene Vorteile für die Software-Entwicklung. Dieser Beitrag stellt diese Eigenschaften vor und leitet daraus Vorteile aber auch Risiken des Open-Source-Einsatzes ab. Auf Basis umfangreicher Projekterfahrungen zeigen die Autoren Strategien und Architektur-Richtlinien auf und illustrieren diese anhand von Projektbeispielen.

Inhaltsübersicht

1 Einleitung

Open-Source-Software hat einen bedeutenden Einfluss auf die professionelle Software-Entwicklung. Viele Projekte setzen heute wie selbstverständlich Open-Source-Produkte dort ein, wo vor kurzem noch kommerzielle Software dominierte.

Nach unseren Erfahrungen benutzen gut ausgebildete Software-Entwickler gerne Open-Source-Software. Sie können sich auf Grund ihrer Ausbildung schnell in diese einarbeiten, auch ohne umfassende Benutzungsdokumentation und Support. Sie erkennen wegen ihrer Erfahrung mit anderen Produkten frühzeitig, ob sich die Open-Source-Software für den vorgesehenen Einsatzzweck eignet und können sie in ihrer Funktionalität und Reife bewerten. Die Möglichkeit, auf den Quelltext zuzugreifen, um Details darin nachzusehen, Probleme zu beheben oder manchmal die Software auch umfassender auf die zu erledigende Aufgabe zuzuschneiden, wird von ihnen sehr geschätzt.

Die Autoren dieses Artikels, alles erfahrene Software-Entwickler der it-wps GmbH (siehe www.it-wps.de), kennen Möglichkeiten und Schwierigkeiten des Einsatzes von Open-Source-Produkten aus Beratungsprojekten bei Kunden und aus eigenen Entwicklungsprojekten. In ihrer Arbeit als wissenschaftliche Mitarbeiter am Arbeitsbereich Softwaretechnik des Fachbereichs Informatik der Universität Hamburg haben sie sich mit dem Phänomen Open Source beschäftigt.

Wir unterscheiden grob verschiedene Kategorien von Open-Source-Software:

- **Anwendungssysteme** sind vollständige Produkte für Endbenutzer. Beispiel: Die Office-Suite OpenOffice.
- **Anwendungs-Infrastruktur** stellt eine Basis für Anwendungssysteme zur Verfügung. Beispiel: Der Webserver Apache.
- **Betriebssysteme** sind die zugrunde liegende Steuerungssoftware für Computer. Beispiel: Das unixbasierte Betriebssystem Linux.
- **Software-Werkzeuge** werden von Software-Designern und –Programmierern zur Entwicklung von Anwendungen benutzt. Beispiel: Die Entwicklungsumgebung und universelle Werkzeugplattform Eclipse.
- **Bibliotheken** werden in Anwendungen verbaut und mit ausgeliefert. Beispiel: Der Xerces Parser für XML.

Dieser Artikel beschreibt den Einsatz von Open Source bei der Software-Entwicklung. Daher beschränken wir uns auf Open-Source-Software aus den beiden letzten Kategorien.

2 Nutzen des Open-Source-Einsatzes

Die Vorteile von Open-Source-Software erstrecken sich von der Programmierung bis zur Projektorganisation und Projektplanung. Der Einsatz von Open-Source-Software birgt aber auch Risiken. Im Folgenden diskutieren wir Empfehlungen für den erfolgreichen Einsatz von Open-Source-Software anhand ihrer besonderen Eigenschaften:

- Für Open-Source-Software fallen keine Lizenzkosten an.
- Ihr Quelltext ist frei verfügbar und kann eingesehen werden.
- Die Organisation von Open-Source-Projekten unterscheidet sich von der kommerzieller Projekte.

2.1 Keine Lizenzkosten

Open-Source-Software steht kostenfrei zur Verfügung. In der Regel sind die Lizenzkosten von Software-Werkzeugen und -Bibliotheken im Verhältnis zu den Personalkosten gering. Doch es gibt auch teure Software, bei der der Kostenvorteil von Open-Source-Software eine Rolle spielt. Auch kann es für viele Management-Entscheidungen wichtig sein, keine Lizenzkosten zu haben. Für die Kostenplanung ergibt sich der Vorteil, dass der Investitionsplan keine Open-Source-Software aufzuführen braucht. So kann leicht ausprobiert werden, ob eine bestimmte Technologie einsetzbar ist und welche Anforderungen daran bestehen. Während des Projektverlaufs kann noch entschieden werden, welche Open-Source-Bibliotheken oder -Werkzeuge eingesetzt werden.

Allerdings sollte man sich von der kostenlosen und leichten Beschaffbarkeit nicht dazu verleiten lassen, zuviel Open-Source-Software einzusetzen. Dadurch kann unnötige, vielleicht sogar duplizierte Funktionalität ins System kommen. Durch nicht zusammenpassende Komponenten ohne gemeinsame Architektur wird die Systemstruktur verkompliziert. Beim Einsatz kommerzieller Software wirken die Lizenzkosten regulativ: Niemand wird ein zweites Druck-Tool einkaufen, wenn bereits für viel Geld ein Druck-Tool beschafft wurde. Eine andere Gefahr ist, kommerzielle Produkte von vornherein aus der Betrachtung auszuschließen. Damit beschneiden sich die Projekte selbst in ihren Möglichkeiten. Schließlich gibt es nach wie vor sehr nützliche kommerzielle Software, denen keine äquivalente Software aus dem Open-Source-Bereich gegenübersteht.

Da sich Open-Source-Projekte nicht über Lizenzkosten tragen müssen, werden einige sehr nützliche kleine Werkzeuge und Bibliotheken als Open Source angeboten. Diese Systeme würden sich auf Grund ihrer Größe für den kommerziellen Markt kaum lohnen. Das Testrahmenwerk JUnit ist ein weit verbreitetes Beispiel dieser Kategorie, das wir in allen Java-Projekten nutzen. Die kleinen Systeme sind in der Regel einfach gehalten, schnell zu erlernen und effizient einzusetzen.

Der große Erfolg von Open-Source-Software hat Einfluss auf die Verfügbarkeit innovativer Produkte. Einerseits ist es schwieriger geworden, neue kommerzielle Produkte erfolgreich am Markt zu positionieren. Es besteht die Gefahr, dass Ideen sehr schnell in Open-Source-Produkte übernommen werden und damit die Deckung der initialen Entwicklungskosten unwahrscheinlich wird. Andererseits hat die Open-Source-Bewegung selbst viele innovative Produkte hervorgebracht.

2.2 Offener Quelltext

Bei Open-Source-Software steht der Quelltext zur Verfügung. Bei Unklarheiten und Problemen können die Entwickler direkt den Quelltext der Open-Source-Software lesen und haben auch das Recht, ihn zu ändern. Dies hilft sowohl zur Klärung von Fragen, die von der Dokumentation nicht eindeutig oder vollständig abgedeckt werden, als auch zur schnellen Beseitigung von Fehlern. Darüber hinaus kann das System an eigene Bedürfnisse angepasst werden, indem die Funktionalität verändert oder ergänzt wird. Leider führt der offene Quelltext manchmal dazu, dass Open-Source-Software eher spärlich dokumentiert ist und somit der Blick in den Quelltext unumgänglich wird.

Die Lebensdauer von Open-Source-Software ist nicht an die Lebensdauer einer Firma gebunden. Selbst wenn die Entwicklergemeinde einer Open-Source-Software zur Ruhe kommen sollte, steht immer noch der Quelltext für eigene Weiterentwicklungen zur Verfügung. So kann es nicht geschehen, dass z.B. eine eingesetzt Bibliothek einen spät entdeckten Fehler enthält, der aber auf Grund der Insolvenz der Herstellerfirma nicht mehr behoben werden kann.

2.3 Organisation von Open-Source-Projekten

Open-Source-Projekte werden meistens von einer räumlich verteilten, heterogenen Entwicklergruppe erstellt und weiterentwickelt. Jeder Interessierte ist eingeladen, der Entwicklergemeinde beizutreten und kann seine Vorschläge,

Änderungen und Korrekturen einbringen. Häufig gibt es eine verantwortliche Person oder Gruppe, die entscheidet, welche Änderungen in die Software einfließen und der Gemeinde zur Verfügung gestellt werden. Die Projekte sind über Webseiten wie http://sourceforge.net/ zugreifbar.

Diskussionsforen bieten schnelle Hilfe bei Fragen oder Problemen. Hier antworten die Entwickler der Software und erfahrene Benutzer, so dass man mit kompetenter Hilfe rechnen kann. Unsere Erfahrungen zeigen, dass häufig gezieltere und schnellere Hilfe geboten wird als bei kommerziellen Support-Hotlines. Einige unserer Projekte setzen z.B. das Eclipse-PlugIn-Modell bzw. die Eclipse-Rich-Client-Plattform zur Modularisierung der Software ein. Dabei erhielten wir stets schnelle Unterstützung. Allerdings geben die Entwickler von Open-Source-Software häufig keine Garantien über Support und Weiterentwicklung. Manchen Support kann man gesondert einkaufen – z.B. bieten JBoss und MySQL kostenpflichtigen Support. Für viele Open-Source-Produkte ist ein Partner, der die benötigten Garantien gibt, aber schwer zu finden.

Nach unserem subjektiven Eindruck hat viel genutzte Open-Source-Software – wie z.B. die Java-Servlet-Engine Apache-Tomcat oder das Testrahmenwerk JUnit – weniger Fehler als kommerzielle Pendants. Den Grund sehen wir in der weiten Verbreitung und darin, dass viele Nutzer der kostenlosen Software wieder etwas an die Entwickler der Open-Source-Software zurückgeben möchten. Dies tun sie in Form von Fehlerberichten.

Neue Versionen stehen schneller zur Verfügung als bei kommerzieller Software. Sie werden ohne zeitaufwendiges Roll-Out direkt über das Internet zur Verfügung gestellt. So können dringend benötigte Funktionalitäten oder Fehlerkorrekturen in der Software-Entwicklung schnell eingesetzt werden. Über Fehlerlisten kann auch direkt Einfluss auf die Entwicklung einer Open-Source-Software genommen werden.

Die Organisation von Open-Source-Projekten macht neue Vorgehensweisen für die Auswahl der Produkte notwendig. Für den Einkauf kommerzieller Software existieren etablierte Mechanismen zur Bewertung der Investitionssicherheit. Neben der Funktionalität und dem Preis wird insbesondere der Hersteller evaluiert. Für die Bewertung von Open-Source-Software müssen diese Instrumente erst aufgebaut werden. Die Download-Statistiken bei SourceForge und anderen geben erste Hinweise; sie reichen aber noch nicht aus. Vielfach ist bei der Auswahl von wenig genutzter Open-Source-Software unklar, ob sie ausreichend lange weiterentwickelt werden wird. Im Zweifelsfall sollte das Projekt in der Lage sein, die Software auszutauschen oder an die eigenen Bedürfnisse anzupassen.

3 Software-Architektur für den Open-Source-Einsatz

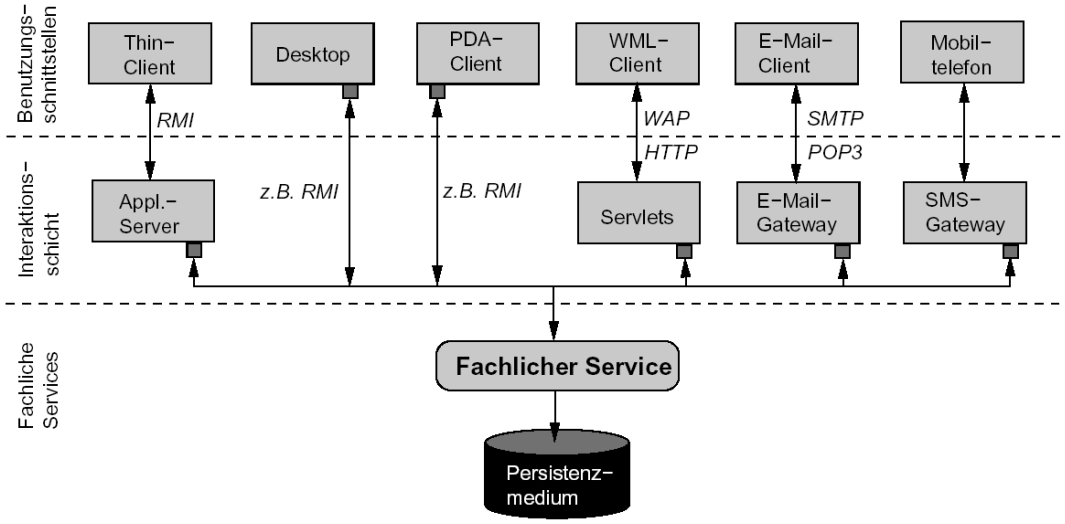

Abbildung 1: Mehrkanalfähige Modellarchitektur mit Fachlichen Services

Eine gute Software-Architektur ist mitentscheidend für die Qualität der entwickelten Produkte. Dies gilt auch und insbesondere bei dem Einsatz von Open-Source-Bibliotheken:

- Die freie Verfügbarkeit von Open-Source-Software führt dazu, dass oft viele verschiedene Komponenten in einem Entwicklungsprojekt eingesetzt werden.

- Es erscheinen häufig neue Versionen von Open-Source-Software mit erweiterter Funktionalität und Fehlerbehebungen. Dabei können sich auch Programmierschnittstellen (APIs) ändern.
- Manchmal wird Open-Source-Software nicht weiterentwickelt, z.B. wenn sich eine andere Software als besser herausstellt und Entwickler in dieses Projekt wechseln. Deshalb sollte es möglich sein, eingesetzte Open-Source-Software durch ein anderes System ähnlicher Funktionalität zu ersetzen.

Selbst wenn die zu Beginn eine Projekts ausgewählten Open-Source-Bibliotheken architektonisch passen, so droht durch neue Versionen und das Austauschen von Bibliotheken der Zerfall der Softwarearchitektur (siehe Abschnitt 2.1). Durch die explizite Wahl einer geeigneten Modellarchitektur (eines Architekturstils, siehe [Züllighoven 1998]) lässt sich der potentielle Architekturzerfall verhindern. Modellarchitekturen legen den prinzipiellen Aufbau einer Softwarearchitektur fest und beschreiben die Art der möglichen Komponenten und deren Zusammenspiel. Sie können für mehrere Projekte wiederverwendet werden. Anhand der Modellarchitektur lassen sich die Open-Source-Bibliotheken passend in ein Softwaresystem eingliedern.

In unseren Projekten verwenden wir sehr häufig eine Modellarchitektur mit Fachlichen Services (vgl. Abbildung 1). Fachliche Services kapseln die fachliche Funktionalität und zugehörige Zugriffe auf ein Persistenzmedium (siehe [Lippert et al. 2001]). Sie werden über eine fachlich motivierte Schnittstelle angesprochen. Die konkrete Implementierung kann einfach ausgetauscht werden (z.B. kann eine datenbasierte Persistierung unter Verwendung von XML durch eine relationale Datenbank ersetzt werden). Weitere Vorteile dieser Architektur sind ihre Mehrkanalfähigkeit, d.h. die Unterstützung verschiedener Ein- und Ausgabeformen auf unterschiedlichen Endgeräten, sowie die einfache Realisierbarkeit von verteilten Anwendungen.

Diese Architektur trennt fachliche von technischer Funktionalität. Fachliche Funktionalität, das Kernwissen der Anwendung, das i.a. langlebig ist, sollte geschützt werden vor technischen Details, z.B. der verwendeten Datenbank oder der konkreten Benutzungsoberfläche. Gerade für diese technischen Komponenten wird oft Open-Source-Software eingesetzt (OR-Mapper, Datenbanken, UI-Toolkits).

Um Komponenten austauschen zu können, ist deren Kapselung wichtig. Komponenten sollten über eine wohldefinierte Schnittstelle integriert werden. Diese darf nicht auf ein spezifisches Produkt zugeschnitten sein, sondern sollte verschiedene Implementierungen ermöglichen. Bei ausschließlichem Zugriff über die Schnittstelle fällt ein Komponentenwechsel viel leichter, als wenn technische Details von außen sichtbar wären. Die vorgestellte Modellarchitektur ermöglicht diese Kapselung durch Fachliche Services. Die rein fachlich motivierte Schnittstelle braucht z.B. bei einem Wechsel der Persistierung nicht geändert zu werden, die Architektur bleibt stabil. Für Bibliotheken in Benutzungsoberflächen, z.B. UI-Toolkits in Desktop-Anwendungen verwenden wir in obiger Modellarchitektur ähnliche Prinzipien.

4 Projektbeispiele

Die folgenden zwei Praxisbeispiele zeigen sinnvolle Kombinationen von Open-Source-Software. Bei Projekt A handelt es sich um eine Produktentwicklung, die wir in ihrem fachlichen Zusammenhang erläutern. In Projekt B wurde eine Individualsoftware entwickelt, die wir anonymisiert darstellen. Wir haben bewußt zwei technologisch verschiedene Projekte gewählt: Während Projekt A eine Rich-Client-Architektur illustriert, handelt es sich bei Projekt B um eine Webanwendung, auf die verschiedene Anwendergruppen verteilt zugreifen. Beide Projekte basieren auf der vorgestellten Modellarchitektur und wurden in der Programmiersprache Java realisiert.

4.1 Projekt A

MobiVisit ist ein Anwendungssystem zur Verbesserung der Dokumentationsqualität in der Onkologie und Hämatologie (siehe http://www.mobivisit.de/). Ärzte können mit einem Tablet-PC während der Visite auf Patientendaten zugreifen und Diagnosen, Nebenwirkungen u.ä. gleich am Krankenbett erfassen. Auf stationären Rechnern werden diese Daten dann aufbereitet und ausgewertet.

Die Entwicklung beruht fast ausschließlich auf Open-Source-Produkten. Als Grundgerüst wird das Rahmenwerk JWAM (<Verweis auf Kasten>) verwendet. Fachliche Services verwalten u.a. Patienten und Diagnosen. Während der anfänglichen Entwicklung wurden sie durch Objektserialisierung persistent gemacht. Mit beginnender Einsatzphase erfolgte die Umstellung auf die relationale Open-Source-Datenbank SAP-DB. Zur Umwandlung der Objekte in eine flache Tabellenstruktur wird der OR-Mapper Torque eingesetzt. Weitere verwendete Bibliotheken sind iText (PDF-Erzeugung), JFreeChart (Diagrammdarstellung), log4j (Logging) und Xerces (XML-Parser).

Die Entwicklung erfolgt u.a. mit den Open-Source-Werkzeugen Eclipse (Entwicklungsumgebung), Ant (Build Tool), Integration Guard (Absicherung des Integrationsservers), CVS (Versionskontrolle) und Chiki (wiki-basierte Kommunikationsplattform). Für das Testen werden JUnit (Modultests) und Jemmy (Akzeptanztests) eingesetzt.

Die verschiedenen Bibliotheken decken unterschiedliche Bereiche ab. Sie lassen sich auf Basis der gewählten Modellarchitektur gut kombinieren. Die Open-Source-Werkzeuge konnten alle in die Entwicklungsumgebung Eclipse integriert werden. So können die Entwickler effizient damit umgehen. Insgesamt lässt sich feststellen, dass sich die Open-Source-Software im Projekt gut bewährt hat.

4.2 Projekt B

In Projekt B wurde eine Webanwendung erstellt. Anwendergruppen an verschiedenen Orten koordinieren sich über diese Anwendung. Das Projekt zeichnet sich durch einen hohen Einsatz von Open-Source-Werkzeugen aus.

Als Rahmenwerk für die Web-Entwicklung mit Java dient die Open-Souce-Bibliothek Struts. Sie basiert auf verschiedenen Standard-Technologien wie Java-Servlets, JavaBeans und XML. Mit Struts werden die an der Benutzungsoberfläche auslösbaren Aktionen mit der fachlichen Logik verbunden und Folgeseiten definiert. Struts erlaubt es, Anwendungs-Architekturen nach dem Model-View-Controller-Muster zu erstellen. Diese können gut zu Fachlichen Service-Architekturen ausgebaut werden. Da bei Struts einige Informationen, wie z.B. die jeweilige Nachfolgeseite, in XML-Dateien festgehalten werden, ist ihre Korrektheit in Java leider nicht statisch prüfbar. Dennoch hat sich Struts auf Grund seiner Qualität als Rahmenwerk für Web-Anwendungen etabliert. Neben Struts werden die Fachwerte aus JWAM zur Definition von fachlichen Werttypen eingesetzt.

Abgestimmt auf die Web-Entwicklung mit Struts werden neben JUnit (Modultests) auch die Testwerkzeuge httpUnit (Test von Web-Requests) und StrutsTestCase (Test von Struts-Actions) genutzt. Weitere eingesetzte Werkzeuge: Ant (Build Tool), Integration Guard (Absicherung des Integrationsservers) CVS (Versionskontrolle) und XPlanner (Koordination von Benutzer-Anforderungen in agilen Softwareprojekten).

Das Projekt hat positive Erfahrungen mit der eingesetzten Open-Source-Software gemacht. Es hat sich z.B. bewährt, Struts in Kombination mit den web-spezifischen Testwerkzeugen httpUnit und StrutsTestCase einzusetzen. Mit httpUnit können Anteile der Akzeptanztests automatisiert werden. Leider müssen bei kleinen Änderungen in den Webseiten, wie das Umbenennen eines Formulars, die Tests angepasst werden. Der Aufwand für händisch ausgeführte Tests wäre jedoch viel höher. StrutsTestCase ermöglicht Modultests der sogenannten Struts-Actions. Diese enthalten die Anwendungslogik, die beim Absenden einer Webanfrage ausgeführt wird.

5 Fazit

Open-Source-Werkzeuge und -Bibliotheken sind wertvolle Ergänzungen in Softwareprojekten. Viele Projekte setzen sie heutzutage selbstverständlich ein. Open-Source-Software steht kostenfrei zur Verfügung, sie wird in einem offenen Entwicklungsprozess erstellt, der Quelltext liegt bei und darf angepasst werden. Diese Eigenschaften von Open-Source-Software machen sie attraktiv für den Einsatz in Softwareprojekten. Die Kombination von verschiedenen, sich verändernden Bibliotheken birgt aber das Risiko eines Architekturzerfalls. Durch eine Modellarchitektur, wie die obige Service-Architektur, kann dieser verhindert werden. So werden Open-Source-Produkte optimal eingesetzt.

JWAM 2

JWAM (siehe http://www.jwam.de) ist ein Framework für die Konstruktion interaktiver Java-Anwendungen nach dem WAM-Ansatz (WAM = Werkzeug, Automat, Material; siehe [Züllighoven 1998]). Seine Wurzeln reichen bis ins Jahr 1997 zurück. Seit 2000 wird JWAM im Rahmen der it-wps professionell weiterentwickelt. Es wurde bis 2003 an Kunden kommerziell lizenziert. So entstanden insgesamt neun JWAM-Versionen, deren Funktionsumfang stetig anstieg. In den letzten Versionen enthielt JWAM deutlich mehr als 1.000 Klassen.
Während der kommerziellen Lizenzierung haben wir eine Reihe wichtiger Erfahrungen gesammelt, die schließlich in dem Entschluss mündeten, JWAM zu restrukturieren und unter eine Open-Source-Lizenz zu stellen:

- *Blaupause*: JWAM gibt eine Architektur für interaktive Anwendungen vor. Es hat sich herausgestellt, dass komplexe Projekte immer wieder leicht unterschiedliche Anforderungen an die Software-Architektur haben. Diese speziellen Anforderungen lassen sich am einfachsten dadurch umsetzen, dass man JWAM selbst anpasst. Daher wurde JWAM schon immer im Quellcode ausgeliefert. Die Umstellung auf eine Open-Source-Lizenz ist die konsequente Weiterführung dieser Entwicklung. In diesem Sinne verstehen wir JWAM heute als Blaupause für ein Architektur-Framework, auf deren Basis in konkreten Projekten eigene Frameworks entwickelt werden.
- *Weiterentwicklung von JWAM*: Im Rahmen der projektspezifischen Anpassungen entstehen immer wieder neue Ideen und Implementierungen, die auch JWAM direkt bereichern können. Es fällt den Projekten viel leichter, solchen Code für die Verwendung in JWAM freizugeben, wenn JWAM Open-Source ist.

Seit Anfang 2004 steht JWAM in der restrukturierten Fassung JWAM 2 unter einer Open-Source-Lizenz. Um die Verwendung in beliebigen kommerziellen Kontexten zu erlauben, wurde JWAM unter die sehr freie *Common Public License* gestellt. Die Restrukturierungen von JWAM haben dazu geführt, dass die einzelnen JWAM-Komponenten in eigenen Projekten weiterentwickelt werden. So ist außerdem gewährleistet, dass unerwünschte Beziehungen zwischen den Komponenten vermieden werden. Die verbliebenen JWAM-Komponenten enthalten nur noch einen Bruchteil der Klassen von JWAM 1. Mit der Größenreduktion konnte auch die Qualität der verbliebenen Funktionalität erhöht werden.

6 Literatur

[Apache 2004] The Apache Software Foundation, http://www.apache.org/.
[Bleek et al. 1999] Bleek, W.-G. et al.: Frameworkbasierte Anwendungsentwicklung (Teile 1 bis 6), OBJEKTSpektrum 1/99 bis 2/00, SIGS-DATACOM, Troisdorf, 1999/2000.
[Lippert et al. 2001] Lippert, M; Wolf, H.; Züllighoven, H.: Domain Services for Multichannel Application Software. In: Proceedings of Hawaii International Conference on System Sciences 2001, HICSS 34, IEEE Computer Society, 2001.
[OSI 2004] Open Source Initiative (OSI), http://www.opensource.org/
[SourceForge 2004] Open-Source-Software Entwicklungsseite SourceForge.net, http://sourceforge.net/.
[Züllighoven 1998] Züllighoven, H.: Das objektorientierte Konstruktionshandbuch. dpunkt.verlag, Heidelberg, 1998.

Stichwörter: Architekturstil, Open Source, Software-Entwicklung, Service-Architektur, Software-Architektur, Software-Bibliotheken, Software-Werkzeuge, JWAM