Verstehen und Nutzen der Eclipse WTP Architektur. Ein Erfahrungsbericht

Jürgen Rückert, Barbara Paech
Universität Heidelberg, Fakultät für Mathematik und Informatik, Lehrstuhl Software Engineering
{rueckert, paech}@informatik.uni-heidelberg.de

Abstract. Die Zahl der Plug-Ins für die Eclipse Plattform wächst zunehmend. Die Basis einer schnellen Entwicklung von Plug-Ins sind geeignete Beschreibungen, wie beispielsweise Architekturmodelle. Architekturmodelle erlauben es wiederverwendbare Komponenten schnell zu identifizieren und Schnittstellen für gewünschte Erweiterungen auf einen Blick zu finden. In diesem Beitrag stellen wir unsere Erfahrungen mit den Architekturmodellen der Eclipse Web Tools Plattform (WTP) vor. Wir zeigen auf, inwieweit diese Beschreibungen Software-Architekten, -Designern und -Entwicklern bei der Beantwortung wichtiger architektonischer Fragen helfen.

1 Einleitung

Die Eclipse Web Tools Platform (WTP) [2] erweitert die Eclipse Plattform [1] mit Funktionalität zur Entwicklung von J2EE-basierten Anwendungen [10]. Die WTP bietet eine Reihe von Navigatoren, Wizards und Editoren für HTML, JavaScript, CSS, JSP, SQL, XML, DTD, XSD und WSDL [14] an.

Wir haben vor die WTP mit einem neuen Anwendungsfall zur Entwicklung von Web Services zu erweitern. Dabei möchten wir einerseits einen neuen Wizard in die WTP integrieren, andererseits möchten wir bestehende Teile der WTP im Wizard wiederverwenden. Die drei Stakeholder unseres Entwicklungsprojekts, Software-Architekt, -Designer und -Entwickler, haben in den verschiedenen Aufgabenbereichen der Entwicklung Fragen zur Architektur der WTP, die sie mit Endbenutzer-Dokumentation [3], Online-Hilfe [4], Entwickler-Dokumentation [5] und Quellcode [6] zu beantworten versuchen.

In Abschnitt 2 stellen wir die damit beantwortbaren, aber auch die unbeantwortbaren Fragen vor. In Abschnitt 3 fassen wir den Beitrag zusammen.

2 Erfahrungen

In diesem Abschnitt kategorisieren wir die verwendeten Beschreibungen (2.1), stellen das Meta-Modell der Eclipse Plattform vor (2.1), welches wir aus den Beschreibungen abgeleitet haben, und stellen die Erfahrungen unserer drei Stakeholder vor (2.2 bis 2.4).

2.1 Beschreibungen

Die Architektur der WTP ist in [3], [4], [5] und [6] dokumentiert mit Hilfe von folgenden Beschreibungen:

- Online-Hilfe: Textuelle Beschreibung von Aufgaben und Anwendungsfällen für Endbenutzer.
- Komponenten-Diagramm: High-Level-Darstellung der Hierarchie von logischen Komponenten für Architekten.
- Komponenten-Beschreibungen: Textuelle Beschreibungen und Übersichten über Sub-Systeme, Features und Plug-Ins von Eclipse-Teilprojekten für Architekten und Designer.

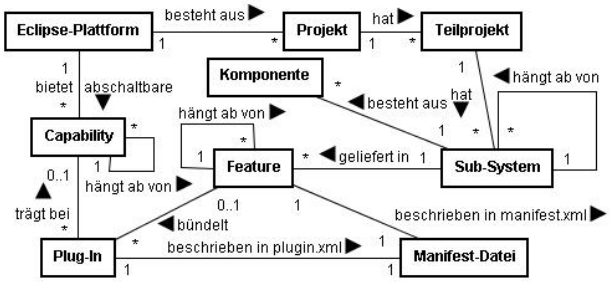

Abbildung 1: Meta-Modell der Eclipse Plattform

- Paketdiagramm: Darstellung von Abhängigkeiten zwischen Sub-Systemen von Eclipse-Projekten für Designer und Entwickler.
- Klassendiagramm: Darstellung von Abhängigkeiten zwischen Features oder von Abhängigkeiten zwischen Capabilites für Designer und Entwickler.
- Manifest-Dateien (XML-Dokumente): Beiträge einer Komponente (Feature, Plug-In, Capability) zur Eclipse Plattform für Designer und Entwickler.

Aus der vorgestellten Dokumentation kann das Meta-Modell der Eclipse Plattform abgeleitet werden, welches in Abbildung 1 dargestellt ist. Projekte sind beispielsweise „Tools“, „Web Tools“ und „Test & Performance Tools“ [8]. Teilprojekte des WTP-Projekts sind beispielsweise „WST“ (Web Standard Tools), „JST“ (J2EE Standard Tools) und „JSF“ (Java Server Faces Tools), wobei das WST-Teilprojekt aus 19 Sub-Systemen besteht, die teilweise voneinander abhängen. Die Features von Sub-Systemen stellen die kleinste Auslieferungseinheit dar, deren Plug-Ins die kleinste Entwicklungseinheit. Das WST-Teilprojekt beispielsweise besteht aus 141 Plug-Ins. Plug-Ins sind häufig so stark gekoppelt, dass eine Wiederverwendung unmöglich wird [18].

2.2 Fragen des Architekten

F.1.1 Aus welchen Komponenten besteht die WTP? Eine zuverlässige Antwort gibt die Verzeichnisstruktur des Quellcodes, die eine Namenskonvention [7] erfüllt.
F.1.2 Welche Funktionalität bietet die WTP? Zur Beantwortung wird ein *„Nutzungs-Modell“* benötigt, welches Rollen, Aufgaben, Nutzungsfälle und Systemfunktionen in Beziehung setzt, welches derzeit aber nicht verwendet wird. Die Aufgaben-orientierte Softwareentwicklung [12] setzt bereits ein solche Modell ein.
F.1.3 Mit welchen Komponenten der WTP wird welche Funktionalität umgesetzt? Zur Beantwortung wird ein *„Dienst-Modell“* benötigt, welches Systemfunktionen (Dienste) mit architektonischen Komponenten in Beziehung setzt. Ein solches Modell existiert derzeit nicht.
F.1.4 Welche Komponenten bieten welche Schnittstellen an? Die Antwort geben die Manifeste von Features und Plug-Ins (siehe Quellcode [6]).

F.1.5 Welche Architekturmuster wurden angewendet und sollen angewendet werden? Das bedeutendste Architekturmuster resultiert im Mikro-Kernel [9] der Eclipse Plattform, der eine Implementierung der „OSGi R4 core framework specification“ [13] darstellt und zumindest Aufschluss über die Verwaltung der Plug-Ins zur Laufzeit gibt, nicht notwendigerweise über deren Architektur.
F.1.6 Sind die Komponenten zur Laufzeit verteilt und, wenn ja, wie? Siehe die Dokumentation des Mikro-Kernels [9], betreffend der Verwaltung der Plug-Ins. Die Plug-In-spezifische Dokumentation gibt Aufschluss über eine Verteilung der Komponenten, aus denen das Plug-In aufgebaut ist.
F.1.7 Um welche Art von Komponente (Oberfläche, Server, etc.) handelt es sich? Diese Frage kann durch die von den Sub-Systemen verwendete Namenskonvention [7] und die Dokumentation der Plug-Ins beantwortet werden.

2.3 Fragen des Designers

F.2.1 Durch welche Klassen werden die Komponenten repräsentiert? Siehe Quellcode der Plug-Ins [6].
F.2.2 Wird ein Lebenszyklus-Mechanismus der Komponenten vorgegeben? Siehe die Dokumentation der „OSGi R4 core framework specification“ [13].
F.2.3 Wie ist der Datenfluss zwischen Komponenten und zwischen Klassen? Zur Beantwortung wird ein *Schnittstellen-Kontext-Modell* benötigt, welches die aufrufenden Extension Points und die empfangenden Extensions abhängig vom Anwendungsfall darstellt. Ein solches Modell wird derzeit nicht eingesetzt.

2.4 Fragen des Entwicklers

F.3.1 Welche Schnittstellenklassen der WTP müssen die neu zu erstellenden Klassen realisieren, damit neue Funktionalität durch bestehende Funktionalität ersetzt werden kann? Siehe die API-Dokumentation der Eclipse Plattform (in [4]) und die Extension Points (im Manifest) von erweiterbaren Plug-Ins.
F.3.2 Welche Daten sollen die Schnittstellenoperationen erhalten, welche sollen sie zurückgeben und wann sind Ausnahmen zu werfen? Zur Beantwortung wird ein *Schnittstellen-Kontext-Modell* benötigt, welches Extension Points mit diesen Informationsaspekten erweitert. Derzeit werden Extension Points nicht erweitert modelliert.
F.3.3 In welchen Paketen befinden sich die Klassen und Schnittstellenklassen? Ist Quellcode verfügbar oder nur Bytecode? Siehe das Archiv (gepackte Datei) des gelieferten Plug-Ins.

3 Zusammenfassung

Wir haben versucht die Architektur der Eclipse WTP anhand der verfügbaren Informationen auf der Eclipse Webseite zu verstehen (2.1). Die dort vorliegende, verteilte Dokumentation hat sich für Software-Architekt (2.2), -Designer (2.3) und -Entwickler (2.4) als zu unvollständig und teilweise zu veraltet herausgestellt. Spezifikationen und Entwürfe, die konsistent mit der Implementierung sind, sind nicht vorhanden. Es erscheint deshalb nicht möglich schnell und zuverlässig Teile der WTP wiederzuverwenden, um eine eigene Lösung daraus aufzubauen.

Wenngleich aufwendiger, hat es sich bewährt den Quellcode zu Rate zu ziehen, da dieser durch Namenskonventionen die System-Bestandteile gut wiedergibt und durch die Manifeste von Features, Plug-Ins und Capabilites zuverlässige und auffindbare Schnittstellenbeschreibungen enthält.

Ein Reverse Engineering der Architektur vor einer Neu-Entwicklung von Plug-Ins ist unumgänglich. Die von uns vorgeschlagenen, dabei entstehenden, ergänzenden Modelle wie das *Nutzungs-Modell* (2.2), das *Dienst-Modell* (2.2) und das *Schnittstellen-Kontext-Modell* (2.3, 2.4), könnten eine schnellere Entwicklung auf Basis wiederverwendbarer Teile ermöglichen.

Literatur

[1] Eclipse Plattform. `http://www.eclipse.org/`. Stand 2006-08.

[2] Eclipse Web Tools Platform. [1]`/webtools`. Stand 2006-08.

[3] Eclipse Plattform, Endbenutzer-Dokumentation. [1]`/webtools/`, Links „Community“, „events“, „articles“, „books“, „tutorials“, „presentations“. Stand 2006-08.

[4] Web Application Development Guide. [1]`/webtools/`, Links „WTP 1.5 (June 30, 2006)“, „Download“. Nach Installation öffnen des Menüs „Help > Help Contents“.

[5] Eclipse Web Tools Plattform, Entwickler-Dokumentation. [1]`/webtools/`, Link „Development Resources“. Stand 2006-08.

[6] CVS Repository der Eclipse Web Tools Plattform. Server `dev.eclipse.org`. CVS-Root `/cvsroot/webtools`.

[7] Eclipse Plattform, Namenskonventionen (in Online-Hilfe [4]). Stand 2006-08.

[8] Eclipse Projects. [1]`/projects/`. Stand 2006-08.

[9] Eclipse Platform, Teilprojekt Equinox. [1]`/equinox/`. Stand 2006-08.

[10] Java 2 Enterprise Edition. Sun Microsystems. `http://java.sun.com/javaee/`. Stand 2006-08.

[11] Eclipse Web Tools Platform, Project Overview. Naci Dai, Arthur Ryman. EclipseCon 2005, Burlingame. [2], Link „presentations“. 2005-03-02.

[12] Aufgabenorientierte Softwareentwicklung. Barbara Paech. Springer Verlag, Berlin. 2000.

[13] Spezifikation der OSGi Service Plattform. Release 4. `http://osgi.org`, Link „OSGi Technology“, Link „Download Specs“. 2006-09-09.

[14] Web Services Definition Language 1.1. W3C Note. `http://www.w3.org/TR/wsdl`. 2001-03.

[15] Web Standard Tools (WST) Teilprojekt. [2]`/wst/main.html`. Stand 2006-08.

[16] Komponentenstruktur des Web Standard Tool (WST) Teilprojekts. [2]`/wst/components.html`. Stand 2006-08.

[17] WTP Architectural Overview. [2], Link „Development Resources“, Link „Architectural Overview“. 2004-12.

[18] WTP Subsystems and Features. David Williams. Version 0.2. [2], Link „WTP Subsystems and Features“. 2005-10-09.