Server-Side Eclipse: Mit der Eclipse-Plattform mehr als Rich Clients entwickeln

Die neue Sonnenfinsternis

VON MARTIN LIPPERT UND BERND KOLB

Eclipse als Basis zum Bau von IDEs zu verwenden ist längst kalter Kaffee. Eclipse als Plattform für Rich Clients einzusetzen hat sich innerhalb des letzten Jahres ebenfalls etabliert. Was kommt als Nächstes? Die Antwort liegt auf der Hand: serverbasierte Anwendungen! Werfen wir einen genaueren Blick darauf, warum die Eclipse-Technologie im nächsten Schritt die Server-Seite erobern wird und welche Möglichkeiten es gibt, Eclipse schon heute für serverbasierte Anwendungen zu nutzen.

Die Eclipse Rich Client Platform hat gezeigt, dass sich weite Teile der Eclipse-Technologie gewinnbringend für normale Business-Anwendungen einsetzen lassen. Wie der Name der Plattform schon vermuten lässt, konzentriert sich die RCP allerdings auf die Implementierung von Rich-Client-Anwendungen. Heutige Anwendungssysteme werden allerdings schon lange nicht mehr als reine Rich-Client-Anwendungen entwickelt. Häufig kommt ein Server zum Einsatz, der dem Klienten Aufgaben abnimmt – wie in einer typischen Multi-Tier-Architektur üblich. Teilweise verzichten größere Systeme sogar völlig auf einen Rich Client – entweder, weil ein ausschließlich Web-basiertes Frontend implementiert wird (mit Web 2.0 erlebt das Web-basierte Frontend ja wieder eine Renaissance) oder weil die Anwendungen überhaupt kein interaktives Frontend benötigen, da es sich um reine Backend-Anwendungen handelt.

Wieso dann Eclipse?

Sicherlich besteht ein großer Teil der Eclipse Rich Client Platform aus der Workbench und den damit verbundenen Konzepten und Frameworks, um Rich Client UIs zu implementieren. Aber schauen wir einmal unter die UI-Oberfläche eines Systems. Was verbirgt sich dort, wenn man mit der Eclipse-Technologie Anwendungen implementiert?

Das System wird aus Komponenten, den Plug-ins oder OSGi Bundles zusammengesetzt. Die Abhängigkeiten zwischen diesen Komponenten des Systems werden definiert und es wird exakt deklariert, welcher Teil einer Komponente für andere Teile des Systems sichtbar sein soll. Mithilfe der Eclipse IDE werden diese Abhängigkeiten und Sichtbarkeiten schon während der Entwicklung überprüft, die OSGi Runtime führt diese Prüfung auch zur Laufzeit durch. Die OSGi Runtime ist darüber hinaus auch dafür verantwortlich, dass Komponenten dynamisch (also zur Laufzeit des Systems) hinzugefügt, entfernt oder ausgetauscht werden können. Diese Modularisierungstechnik bildet das Rückgrad jeder Eclipse-basierten Anwendung.

Darüber hinaus nutzt man Extension Points und Extensions, um das System offen und flexibel für Erweiterungen zu gestalten. Schnell benutzt man nicht nur die von der Eclipse-Plattform vordefinierten Extension Points, um das System zu erweitern, sondern definiert eigene, um auch die selbst implementierten Komponenten für Erweiterungen zu öffnen – und oft beziehen sich diese Extension Points nicht mehr nur auf Oberflächen. Ist es nicht elegant, beispielsweise neue zu verkaufende Produkte einfach inklusive deren Data Transfer Objects, spezieller Services und dem passenden spezialisierten UI dem System per Extension durch ein neues Plug-in hinzuzufügen? Und wäre es nicht verführerisch einfach, auf dem gleichen Wege dem serverbasierten Teil des Systems dieses neue Produkt bekannt zu machen?

Natürlich benötigt der Server nicht das UI des neuen Produktes. Dazu trennen wir die Implementierung des Produktes in zwei Komponenten auf: Die erste Komponente enthält nur die Core-Implementierung des Produktes (alle Nicht-UI-Teile), die zweite Komponente liefert das spezialisierte UI für das Produkt. Eine Design-Richtlinie, die sich in der Vergangenheit bereits für Eclipse-RCP-Anwendungen als nützlich erwiesen hat.

Allerdings besitzt das System nicht nur eine Rich-Client-Oberfläche, sondern ist in Teilen auch per Weboberfläche zu bedienen. Selbstverständlich soll auch das neue Produkt in der Weboberfläche eine Rolle spielen. Wir möchten also das Core-Plug-in des Produktes auch der Webanwendung zugänglich machen, ebenso wie ein spezielles Web-UI-Plug-in, welches die Webanwendung um ein spezialisiertes UI für das Produkt bereichert.

Land der Möglichkeiten

Wir sehen, dass die grundlegenden Techniken Eclipse-basierter Anwendungen längst nicht auf den Rich Client begrenzt sind. Aber welche Möglichkeiten existieren bereits heute, um auch für serversei-

tige Anwendungen oder Anwendungsteile diese Techniken nutzen zu können?

Headless Eclipse

Die OSGi Runtime sowie die Extension Registry von Eclipse sind völlig ohne UI (im so genannten „Headless"-Modus) sofort einsatzfähig. Das Eclipse SDK selbst bringt hier schon eine Reihe von Beispielen mit:

- Startet man ausschließlich die OSGi Runtime, wird kein UI angezeigt. Die Runtime selbst lässt sich dabei über eine Konsole steuern.
- Der automatisierte Build-Prozess des Eclipse-Projektes ist als Headless-Eclipse-Anwendung realisiert.
- Des Weiteren kann z.B. der Java Code Formatter ohne UI ausgeführt werden oder auch der EMF-Code-Generator hat eine eigene Eclipse-Application, welche ohne UI auskommt.

Mit solchen Headless-Anwendungen lassen sich also bereits einfache Batch-artige Anwendungen auf Basis der Eclipse-Technologie implementieren, die aus Plug-ins zusammengesetzt werden und sowohl die OSGi Runtime als auch den Extension-Point-Mechanismus nutzen können. Hiermit nun aber eine echte serverbasierte Anwendung zu erstellen, wäre sehr mühsam, da ein Web- oder App-Server uns bereits viel Arbeit abnimmt, die wir nicht selbst implementieren wollen.

Web-Container und OSGi

Im nächsten Schritt wollen wir die serverseitigen Anwendungsteile innerhalb eines Webservers (quasi als Application Server) ablaufen lassen. Dabei gibt es grundsätzlich zwei unterschiedliche Ansätze, die beiden Technologien (Webserver und OSGi) zu kombinieren:

- die Servlet-Bridge: Im Rahmen des Equinox-Incubator-Projekts „Server-Side Eclipse" hat Simon Kaegi eine Bridge entwickelt, mit der es möglich ist, die OSGi Runtime von einem Servlet aus anzusprechen und so Webanwendungen aus OSGi Bundles zusammenzusetzen. Der Vorteil dieser Lösung ist, dass der Webserver genauso betrieben werden kann wie eh und je. Dieser Vorteil wiegt besonders schwer, wenn der Webserver zentral administriert wird und auch andere Anwendungen hostet. Diese Servlet Bridge wird bereits vom Eclipse Communication Framework-(ECF-)Projekt genutzt, um einen zentralen Kommunikationsserver zur Verfügung zu stellen. Diese Lösung setzt allerdings die aktuellen Milestone Builds der Eclipse-Version 3.2 voraus. Das einführende Beispiel von Wolfgang Gehner zeigt, wie man die ersten Schritte mit der Servlet-Bridge gehen kann (siehe www.infonoia.com/en/content.jsp?d=inf.05.07).
- der OSGi-basierte Webserver: Startet man den Webserver selbst auf Basis der OSGi Runtime als ein Bundle, können wie selbstverständlich auch alle Webanwendungen die zugrunde liegende OSGi Runtime nutzen. Das Eclipse SDK nutzt diese Art, einen Webserver zu verwenden, um das Hilfe-System von Eclipse zu starten. Dies setzt allerdings voraus, dass man die Start-Prozedur des Webservers beeinflussen kann (damit anstelle des Webservers zunächst die OSGi Runtime gestartet wird, die ihrerseits dann den Webserver zum Laufen bringt), was in vielen Firmen nicht unbedingt der Fall ist.

Web-Container und Extension Points

Neben der Modularisierung mittels OSGi Bundles (bzw. Plug-ins) ist der Extension-Point-Mechanismus der zweite wichtige Bestandteil der Eclipse Runtime, den wir nutzen können, um eine gut strukturierte und vor allem erweiterbare Architektur zu realisieren. Mit der kommenden Eclipse-Version 3.2 wird es möglich sein, den Extension-Point-Mechanismus von Eclipse auch ohne eine OSGi Runtime zu nutzen. Das gestattet es uns, in serverseitigen Anwendungen den Extension-Point-Mecha-

nismus zu verwenden und gegebenenfalls beim Deployment des Systems auf die OSGi Runtime zu verzichten.

Web-Container und Eclipse-UI

Über die reine Verwendung der OSGi Runtime und des Extension-Point-Mechanismus hinaus geht das neue Eclipse Rich AJAX Platform Project Proposal. In diesem Projekt wird daran gearbeitet, eine Eclipse-ähnliche Workbench mittels AJAX-Technologien für Webanwendungen zu erstellen. Damit wird dem Entwickler die Möglichkeit gegeben, nicht nur die gleiche Modularisierungs- und Komponenten-Technologie der Eclipse Rich Client Platform zu nutzen, sondern auch ähnliche APIs und Programmiermodelle für die Entwicklung von Webanwendungen einzusetzen. Das Projekt-Proposal findet sich unter: www.eclipse.org/proposals/rap/. Eine Demo der Technologie kann man im Web finden: rap.innoopract.com/webworkbench.

Java EE, Application Server und Spring

Reicht ein Web-Container nicht für die serverseitige Anwendung als Ablaufumgebung aus und muss stattdessen ein vollwertiger Java EE Application Server her, stellt sich die nächste Frage: Kann die OSGi Runtime in einem kompletten Application Server genutzt werden?

In Mailinglisten behaupten zwar immer wieder Teilnehmer, dass sie bereits JBoss oder andere App-Server mit OSGi zusammen zum Laufen gebracht hätten. Fertig nutzbaren Code scheint es aber bisher nirgends zu geben. Dennoch scheinen sich immer mehr für eine OSGi-Integration in Application-Servern zu interessieren. Auf Anfrage des Eclipse Healthcare-Projektes scheint sich auch das Apache-Geronimo-Projekt die Aufgabe gestellt zu haben, einen OSGi Container zu implementieren.

Der Einsatz von Application-Servern für größere serverseitige Java-Anwendungen ist längst nicht mehr unumstritten. Vor allem das Spring Framework (www.springframework.org) geht hier einen anderen Weg und fördert die leichtgewichtige Konstruktion von serverseitigen Enterprise-Anwendungen mit Java, ohne dass ein Application Server zum Einsatz kommen muss.

Da das Spring Framework selbst keine Hilfe bietet, um große Anwendungen sauber zu modularisieren, scheint eine Integration mit OSGi vielversprechend. Erste implementierte Prototypen zeigen, dass sich beide Technologien nicht nur recht einfach miteinander kombinieren lassen, sondern sich auch hervorragend ergänzen. So lassen sich mit OSGi Bundles Spring-Konfigurationen in Module verteilen, während der Spring-Dependency-Injection-Mechanismus über Modul-Grenzen hinweg einsetzbar bleibt. Darüber hinaus lassen sich beispielsweise mit dem Extension-Point-Mechanismus spezielle, selbst definierte Erweiterungspunkte definieren, deren Extensions vom Anbieter des Extension Point automatisch dem Spring Framework unter einer dezidierten Konfiguration bekannt gemacht werden (einen detaillierteren Blick auf die Kombination von Eclipse Runtime mit dem Spring Framework werfen wir in einem zukünftigen Artikel).

Ausblick

Die Eclipse-Technologie ist längst nicht mehr nur für die Entwicklung von Client-Anwendungen interessant. Wir haben gezeigt, dass auch serverseitige Anwendungen von der Eclipse-Plattform profitieren können. Die unterschiedlichen Möglichkeiten demonstrieren, dass vielfältige Einsatzszenarien denkbar sind und bereits von unterschiedlichen Projekten genutzt werden.

Eine ausführlichere Beschreibung der unterschiedlichen Möglichkeiten, die Eclipse-Technologie für Serveranwendungen zu nutzen, findet sich im kommenden *Eclipse Magazin* (Vol. 7). Dass die Eclipse-Technologie auch die Server-Seite erobern wird, steht für uns mittlerweile völlig außer Frage.

Martin Lippert ist Senior-IT-Berater bei der it-agile GmbH. Er arbeitet dort als Coach und Berater für agile Softwareentwicklung, Refactoring und Eclipse-Technologie und ist Committer im Eclipse-Equinox-Incubator-Projekt. Kontakt: martin.lippert@it-agile.de.

Bernd Kolb ist freiberuflicher Berater und Coach. Er ist (Mit-)Autor von diversen Artikeln sowie eines Buches und Sprecher auf Konferenzen. Seine Schwerpunkte liegen auf modellgetriebener Softwareentwicklung sowie Eclipse-Technologien. Kontakt: b.kolb@kolbware.de.