Past, Present, Future

Evolution von Seam 3

Torsten Fink

Seam ist ein Framework zur Entwicklung Web-basierter Geschäftsanwendungen. Es „säumt" die entsprechenden JEE-Technologien zusammen. Die Version 2 von Seam passt zur Java EE 5. Die gerade erscheinende Version 3 arbeitet mit der aktuellen Java-Version 6 zusammen. Für wen ist welche Version geeignet?

Seam wurde entworfen, um die Java EE 5-Technologien zu einem komfortablen und leistungsfähigen Framework zur Entwicklung von Webanwendungen zu verbinden. Der Funktionsumfang von Seam hat sich schnell weiter entwickelt und es wurde in vielen Projekten erfolgreich eingesetzt. Die Erfahrungen, die dabei gesammelt wurden, sind wieder in die Version 6 der Java Enterprise Edition (EE) eingeflossen. Insbesondere die beiden Technologien *Contexts and Dependency Injection (CDI)* und *JavaServer Faces (JSF)* haben hiervon profitiert.

Als Konsequenz steht jetzt mit der Java EE 6 ein vollständiges Framework für Webanwendungen bereit. CDI, die Basis des Webframeworks, ist allerdings inkompatibel zu der Seam-Version 2. Die neue Version 3 wird nun den Weg konsequent weiter gehen und wieder die aktuellen Java EE-Technologien einsetzen und fortentwickeln.

Was bedeutet dies aber konkret für aktuelle strategische Technologieentscheidungen? Hat Seam mit Erscheinen der Java EE 6 noch eine Relevanz? Kann oder soll man die alte Version 2 noch für Projekte einplanen oder ist besser gleich auf die neue Version 3 zu setzen?

Seam – der Ursprung

Die ursprüngliche Motivation für Seam bestand darin, ein Entwurfsmuster für effiziente Webanwendungen als Framework umzusetzen. Dieses Entwurfsmuster wurde schon vor Beginn des Seam-Projekts von Christian Bauer und Gavin King in [BauKi05] als *Long Session* beschrieben. Die Idee bestand darin, persistente Entitäten über mehrere Anfragen in dem Sitzungskontext der Anwendung zu halten. Dadurch mussten sie nicht mehr für jede Anfrage neu geladen werden.

Die einfache Lösung, dies über einen Servlet-Filter zu realisieren, der am Ende einer Anfrage alle geladenen Entitäten von der Datenbankverbindung abkapselt und sie bei dem nächsten Request wieder mit dieser verbindet, hatte in der Praxis einige Nachteile. Mit Seam wurde daher das Konzept von *Konversationen* eingeführt, die als neuer Kontext (engl. scope) eine Lebensdauer hatten, welche länger als eine Anfrage, aber kürzer als eine Sitzung war. Dies ermöglichte es, Entitäten, und damit den von ihnen verbrauchten Speicher, unkompliziert wieder freizugeben und mehrere Konversationen gleichzeitig zu verwalten, z. B. um unterschiedliche Fenster mit der gleichen Sitzung zu assoziieren.

Um nun ein komplettes Framework für Webanwendungen zur Verfügung zu stellen, entschieden sich die beiden Autoren, die vorhandenen Java EE 5-Technologien zu benutzen, insbesondere JavaServer Faces (JSF) für die Webmasken, Enterprise-JavaBeans (EJB) für die Anwendungslogik und den Zustand der Masken und die Java Persistence API (JPA) für das Ablegen der Entitäten in der Datenbank. Seam benutzte einen einfachen *Injection*-Mechanismus, um die verschiedenen Technologien miteinander zu koppeln.

Interessant ist, dass von den beiden Ideen, Konversation und Injection, die erste die technische anspruchsvollere war, die zweite aber zu der schnellen Verbreitung und Weiterentwicklung von Seam geführt hat. Es fällt immer wieder auf, dass auch Entwickler, die Seam schon länger im Einsatz haben, sich der Vorteile von Konversationen beim Zugriff auf die Datenbank gar nicht bewusst sind.

Die „Sturm und Drang"-Phase

Im Juni 2006 wurde die erste finale Version Seam 1.0.0.GA veröffentlicht und es war der Startschuss für eine Flut von Erweiterungen. Bis zum März 2007 wurden sechs weitere Versionen mit neuen Fähigkeiten realisiert, die meistens inkompatibel zur Vorversion waren. Mit der Version 1.2.1.GA stand dann schließlich ein komfortables Framework für Webanwendungen zur Verfügung, welches über zusätzliche Module eine breite Palette nützlicher Funktionalitäten unterstützte, wie z. B. das Versenden von E-Mails, die Erzeugung von PDF-Dokumenten, die Integration von Geschäftsprozessen und vieles mehr.

2007 wurde dann unter dem Vorsitz von Gavin King der Java Specification Request 299 [JSR299] unter dem damaligen Namen „Web Beans" gestartet. Er sollte die Erkenntnisse von Seam aufgreifen und standardisieren. Schon im Oktober 2007 gab es den ersten öffentlichen Entwurf.

Die Zeit der Konsolidierung

Im November 2007 wurde bei Seam der erste große Releasewechsel auf 2.0.0.GA vollzogen. Pete Muir übernahm die Führung des Projekts. Die aktuelle Version des 2er Zweigs ist die 2.2.2.Final und es gab dazwischen acht weitere Versionen. Im Gegensatz zur vorherigen Phase waren neue Versionen entweder kompatibel zur Vorgängerversion oder es gab eine ausführliche Migrationsanleitung. Die aktuelle Version findet sich auf der offiziellen Web-Site [Seam].

Parallel dazu trieb Gavin King den JSR 299 voran. In einer Rekordzeit von zwei Jahren hat die Expertengruppe im Dezember 2009 die finale Version verabschiedet. Zusätzlich zu dem Einfluss auf JSR 299 sind viele Erkenntnisse aus Seam in die Version 2 der JavaServer Faces (JSF) eingeflossen. Dies ist zum großen Teil Dan Allen zu verdanken, der als Mitglied der Expertengruppe bei der Erstellung der Spezifikation tatkräftig mitgeholfen hatte.

Die Ernte mit Java EE 6

Mit der aktuellen Version 6 der Java EE wurde die Ernte eingefahren. Neben den Erkenntnissen aus Seam sind noch viele weitere Erfahrungen eingeflossen. Hierzu gehören natürlich die Praxisberichte aus der Benutzung der Version 5, aber auch die Berücksichtigung weiterer Frameworks, die sich in der Community etabliert haben, wie z. B. die Unterstützung von REST-ähnlichen Schnittstellen.

Als Ergebnis steht jetzt mit Java EE 6 ein durchgängiges Framework als Industriestandard zur Verfügung, das komfortabel die Erstellung komplexer Geschäftsanwendungen mit Web-basierten Oberflächen ermöglicht.

Was bleibt dann noch für Seam 3?

Abbildung 1 zeigt noch einmal schematisch, wie Seam 2 mit der Java EE 6 zusammenhängt und wie Seam 3 einzuordnen ist. Einige der JSF-Erweiterungen und insbesondere die Integration von Java EE-Komponenten in JSF sind, wie oben erwähnt, in die Version 2 von JSF eingeflossen.

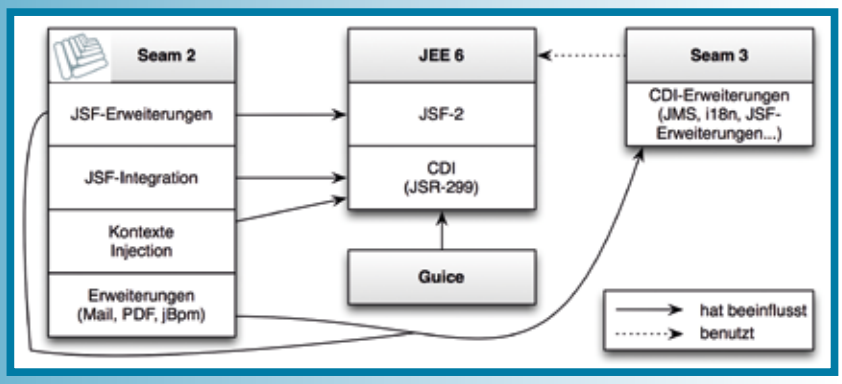

Abb. 1: Abhängigkeiten zwischen Seam 2, Java EE 6 und Seam 3

Der Kern von Seam 2 besteht aus Kontexten und der darauf aufbauenden Injection. Dieser Kern ist die Grundlage für das im JSR 299 spezifizierte *Contexts and Dependency Injection*-Framework (CDI). Aus den Ideen von Seam 2 ist dabei aber nur das Konzept der unterschiedlichen Kontexte und die Idee der Integration in Java EE-Anwendungen per Injection übrig geblieben. Das eigentliche Injection-Framework wurde dagegen runderneuert. Die Grundlage dafür bildete das Framework *Guice*, welches hauptsächlich von dem Google-Mitarbeiter Bob Lee entwickelt wird, der auch Mitglied der Expertengruppe für den JSR 299 war.

Diese Runderneuerung war sicherlich eine sehr gute Entscheidung, da der Injection-Teil von Seam 2 einige Schwächen aufweist. Es gibt keine Typsicherheit beim Auflösen von Referenzen, Fehler sind erst zur Laufzeit erkennbar und es gibt zu viele, teilweise konkurrierende Konzepte für passende Anwendungsarchitekturen. Der Nachteil dieser Entscheidung ist die resultierende Inkompatibilität von Seam 2 zu CDI.

Im Prinzip gab es daher bei dem Entwurf von Seam 3 zwei Alternativen:

- Umhüllen von CDI-Komponenten als klassische Seam 2-Komponenten oder
- komplette Neuentwicklung von Seam 3 auf Basis von CDI.

Die erste Alternative hat den Vorteil, dass sehr schnell eine Version 3 mit einem großen Funktionsumfang fertigzustellen ist und Seam 2-Anwendungen sich einfach und in kleinen Schritten migrieren lassen. Allerdings hat sie auch den Nachteil, dass der gesamte Ballast, der sich insbesondere in der „Sturm und Drang"-Phase angesammelt hat, weiter mitgeschleppt werden muss.

Exemplarisch soll zur Illustration des Ballasts der Fehler JBSEAM-3873 dienen, der im Projektverwaltungswerkzeug Jira einsehbar ist [Jira]. Hierbei handelt es sich um einen Fehler in der Transaktionsverwaltung, der zu einem Speicherloch führt. Der Fehler wurde mit einem Testprogramm zur Reproduktion des Fehlverhaltens im Dezember 2008 eingestellt. Trotz einiger Versuche, den Fehler zu beheben, ist er immer noch offen. Die Hauptursache hierfür liegt nach Meinung des Autors in dem historisch gewachsenen Code, durch den eine Architekturänderung sehr aufwendig wird. Die praktische Bedeutung dieses Problems relativiert sich glücklicherweise etwas, da zusätzlich zu dem Fehler auch gleichzeitig ein Umgehen des Problems in Jira eingestellt wurde.

Diese Historie dürfte ein maßgeblicher Grund gewesen sein, dass sich die Entwicklergruppe um Pete Muir dazu entschied, bei Seam 3 die zweite Alternative, komplette Neuentwicklung auf Basis von CDI, zu wählen. CDI besitzt standardisierte Schnittstellen zur Erweiterung, die von Seam 3 benutzt werden. Die Idee besteht darin, alle Funktionalitäten, die sich in Seam 2 als nützlich für die Entwicklung von Webanwendungen erwiesen haben, und natürlich auch alle neuen Ideen als CDI-Erweiterungen in Form von Seam 3-Modulen umzusetzen. Damit lässt sich Seam 3 unabhängig von dem konkreten CDI-Container einsetzen.

Was bedeutet das für die Entwicklung?

Die Konsequenz der Entscheidung ist, dass das grundlegende Web-Framework die Java EE ist. Seam bietet in Version 3 nur Erweiterungen. Zur Demonstration des Entwicklungsparadigmas soll ein kleines einfaches

Abb. 2: Beispiel einer Java EE 6-Anwendung

Beispiel aus der Weld-Distribution dienen, der Referenzimplementierung von CDI. Abbildung 2-A zeigt die Oberfläche der „Anwendung“. In einem Textfeld wird ein Text aus mehreren Sätzen eingegeben. Per Knopfdruck wird er „übersetzt“, in dem Sinne, dass jeder Satz zu einem konstanten String transformiert wird.

Abbildung 3 zeigt die Klassen der Anwendung. Hier ist nicht die eher triviale Übersetzungslogik interessant, sondern das Konzept, wie diese Logik mit der Webanwendung verknüpft ist. Die relevante Klasse dabei ist `TranslatorControllerBean`, die in Abbildung 2-C dargestellt ist. Wie an den Annotationen zu erkennen ist, handelt es sich um eine zustandsbehaftete EJB. Sie lebt im Kontext einer einzelnen Anfrage (`@RequestScoped`) und ist an der Weboberfläche unter dem Namen `translator` verfügbar.

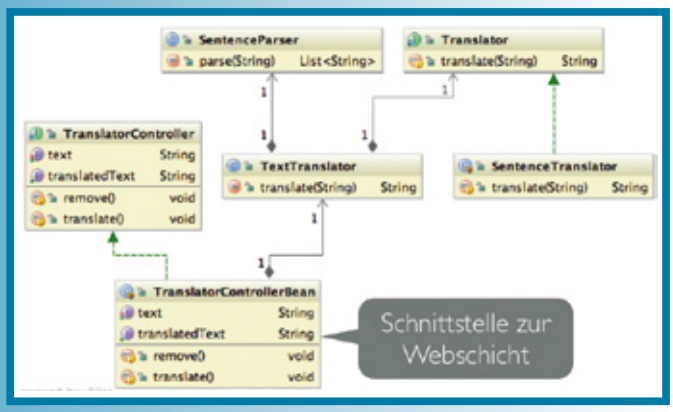

Abb. 3: Klassen der Beispielanwendung

Der Zugriff auf andere Logikkomponenten geschieht ebenfalls über Injection. In diesem Fall wird der Textübersetzer `TextTranslator` injiziert. Im Gegensatz zu Seam 2 ist diese Referenzauflösung typsicher und wird bei der Installation der Anwendung überprüft. Zusätzlich zu der einfachen Injection bietet CDI auch viele weitere Fähigkeiten, wie z. B. Fabrikmethoden, Interzeptoren und ereignisbasierte Interaktion, die auch von anderen modernen Injektionsbibliotheken bekannt sind.

In Abbildung 2-B ist schließlich zu erkennen, wie in der JSF-Seite die Weboberfläche über den Namen der CDI-Komponenten (`translator`) die Verknüpfung mit der Logikkomponente realisiert. Attribute der Komponenten werden mit Textfeldern zur Ein- und Ausgabe verknüpft. Knöpfe, die Aktionen auslösen, werden mit Komponentenmethoden assoziiert.

Das Beispiel zeigt, dass sich jeder Seam 2-Entwickler sofort in Java EE 6 heimisch fühlen wird, da die Architektur die gleichen Konzepte verfolgt.

Warum dann noch Seam 3?

Es stellt sich dann natürlich die Frage, was Seam 2 mehr bietet als Java EE 6, um eine weitere Version Seam 3 noch zu rechtfertigen. Aus Sicht des Autors zählen hierzu insbesondere

- die transparente Benutzung eines von der Anwendung verwalteten Entity-Managers, also einer der beiden Grundideen von Seam,
- die Erweiterung einfacher POJOs um nichtfunktionale Eigenschaften, wie Transaktionalität und Zugriffsschutz,
- JSF-Erweiterungen, wie verteilte Navigationsregeln und verschachtelte Konversationen,
- die Integration weiterer Frameworks, die für Webanwendungen interessant sind, wie z. B. die Integration eines Regelinterpreters, die Erzeugung von PDF- und Excel-Dateien.

Seam 3 möchte nun genau für diese Lücken Erweiterungen in Form von Modulen anbieten. Im April 2011 ist die erste Version 3.0.0 erschienen. Als Projektleiter wurde inzwischen Pete Muir von Shane Bryzak abgelöst. Abbildung 4 zeigt die grobe funktionale Architektur von Seam 3. Das notwendige Fundament, auf dem es aufsetzt, besteht aus einem CDI-konformen Container und *Seam Solder*, einer portablen Bibliothek, die Erweiterungen zum CDI-Standard enthält. Auf dieser Kombination setzen dann die restlichen Module auf.

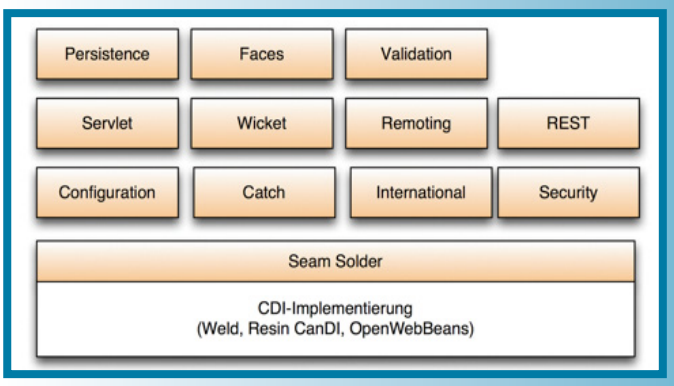

Abb. 4: Die Module von Seam 3

Der funktionale Umfang von Seam 3 ist schon recht groß. *Persistence* implementiert die ersten beiden Punkte der oben aufgeführten Liste von Seam-2 Fähigkeiten, die in Java EE 6 fehlen, die Unterstützung eines erweiterten Entity-Managers und die Erweiterung von POJOs um transaktionale Eigenschaften. Der dritte Punkt der Liste, die Erweiterung von JSF, wird von dem Modul *Faces* bedient. Die restlichen Module zielen auf den vierten und letzten Listenpunkt, der Integration weiterer Frameworks.

Mit *Wicket* steht Seam-Entwicklern eine Oberflächenalternative zu JSF zur Verfügung. *Validation* bietet eine Integration des Hibernate-Validator-Frameworks, welches den JSR-303 unterstützt. *Servlet* ermöglicht einen erweiterten und komfortablen Zugang zu der Basisschnittstelle für Webanwendungen in Java. *Remoting* ermöglicht den entfernten Ajax-Zugriff auf CDI-Komponenten. *REST* integriert beliebige JAX-RS kompatible REST-Frameworks. *Configuration* vereinfacht die Konfiguration per XML-Dateien. *Catch* unterstützt die Verarbeitung von Ausnahmen. *International* vereinfacht den Umgang mit mehreren Sprachen. *Security* schließlich bietet eine Authentisierung über externe Namensdienste oder interne CDI-Komponenten und Möglichkeiten, um komplexere Zugriffschutzszenarien umzusetzen.

Zusammengefasst beinhaltet Seam 3 genau die Ideen, die aus Seam 2 nicht in die Java EE 6 übernommen wurden. Auch wenn die Vielfalt unterstützter Frameworks noch bei Weitem nicht der von Seam 2 entspricht, ist dennoch eine tragfähige Basis für die Entwicklung von Webanwendungen entstanden. Es ist auch davon auszugehen, dass Konzepte, die sich in Seam 3 bewähren, wieder zurück in die Java EE fließen werden.

Eine Bewertung aus strategische Perspektive

Als Architekt oder Entwicklungsleiter steht man regelmäßig vor der Frage, welchen Technologiestack man wählen soll. Ne-

ben den technischen Fähigkeiten ist vor allem die Zukunftssicherheit wichtig. Für vorhandene Seam 2-Anwendungen empfiehlt es sich wahrscheinlich in den meisten Fällen, bei Seam 2 zu bleiben. Eine Migration auf Seam 3 und CDI wird mit Sicherheit sehr aufwendig und es wird immer noch etwas dauern, bis Seam 3 eine vergleichbare Funktionalität aufweisen kann.

Bei neuen Java EE-Projekten ist dagegen der Einsatz von CDI zu empfehlen. Zusammen mit JSF 2 steht einem ein leistungsfähiger und zukunftssicherer Technologiestack zur Verfügung, der dann auch ohne großen Änderungsaufwand um die Fähigkeiten von Seam 3 oder anderer CDI-Module erweitert werden kann.

Literatur

[BauKi05] Ch. Bauer, G. King, Hibernate in Action, Manning, 2005

[Fink07] T. Fink, Das Seam-Web-Framework, in: JavaSPEKTRUM, 3/2007

[Jira] Red Hat, Das offizielle Jira-Projekt für JBoss Seam, 2010, `https://jira.jboss.org/browse/JBSEAM`

[JSR299] Java Specification Request 299: Contexts and Dependency Injection for the Java™ EE platform, `http://jcp.org/en/jsr/detail?id=299` (früher Web Beans genannt)

[Seam] Seam Framework – JBoss Seam, 2010, `http://www.seamframework.org/`

Dr. Torsten Fink ist Leiter des JBoss Competence Center der akquinet AG. Neben der Leitung von JEE-Projekten führt er Architektur- und Technologieberatungen durch. Er ist zertifizierter JEMS Master Architect.
E-Mail: torsten.fink@akquinet.de