



Alles im Lot

Das Seam-Web-Framework

Torsten Fink

Seam ist ein Framework zur Entwicklung Web-basierter Geschäftsanwendungen mit ausgereiften JEE-Technologien. Der Entwickler beschreibt für eine Anwendungskomponente nur die Abhängigkeiten zu anderen Komponenten (Inversion of Control). Seam klinkt sich dann in das Web-Framework JavaServer Faces (JSF) ein und koordiniert zur Laufzeit die Interaktion mit den anderen Technologien. Es stellt damit die Naht (deutsche Übersetzung von Seam) zwischen den unterschiedlichen Technologien dar. Die Vorteile und die Funktionsweise von Seam werden hier an Hand typischer Probleme von JEE-Anwendungen demonstriert.

Die Java Enterprise Edition (JEE) bietet dem Anwendungsentwickler einen Werkzeugkasten an Technologien zur Realisierung von zuverlässigen, skalierbaren, sicheren und wartbaren Geschäftsanwendungen. Zu den wichtigsten Fähigkeiten gehören Transaktionalität, Hochverfügbarkeit (mit Failover-Mechanismen), Parallelbetrieb (Clustering) und Integration mit externen Authentisierungs- und Verwaltungssystemen. Allerdings wurde die JEE nicht nur für Webanwendungen konzipiert, sondern unterstützt ebenso Desktopanwendungen als Klienten und auch reine Backendsysteme.

Aus der Perspektive des Webentwicklers ergibt sich dadurch das Problem, dass er aus seiner Webanwendung heraus die anderen JEE-Technologien manuell integrieren muss. Zu diesen Technologien gehören insbesondere Enterprise-JavaBeans (EJB) für die Anwendungslogik und die Java Persistence API (JPA) für die Speicherung der Geschäftsobjekte. Auch wenn die Benutzung von EJBs in der aktuellen Version JEE 5 deutlich vereinfacht wurde, muss der Entwickler Code schreiben, der die einzelnen Schichten miteinander verknüpft.

Hier setzt Seam ein. Seam ist ein Framework zur Entwicklung von Webanwendungen, das sich in JavaServer Faces (JSF) einklinkt und die Ankopplung der verschiedenen Schichten übernimmt. Der Entwickler kann sich so auf die fachlichen Aspekte konzentrieren und muss keinen technischen Verknüpfungscod (Glue-Code) selbst schreiben. Um die Funktionsweise von Seam zu verdeutlichen, werden im Folgenden ein paar typische Probleme von JEE-Anwendungen eingeführt und es wird gezeigt, wie Seam diese löst. Ein gutes Verständnis von JSF wird dabei vorausgesetzt.

Zugriff auf Applikationslogik

Enterprise-JavaBeans (EJBs) bilden in der JEE das Framework für die Anwendungslogik. Hier ein Ausschnitt aus einer EJB zur Verwaltung von Benutzern:

```
@Stateless
public class BenutzerverwaltungBean implements Benutzerverwaltung {
    ...
    @TransactionAttribute(REQUIRED)
    @RolesAllowed("admin")
    public void loescheBenutzer(String benutzerKennung) {
        ...
    }
}
```

Die Geschäftsmethode wird automatisch in einer Transaktion ausgeführt, alle Zugriffe auf Datenbanken und sonstige Res-

ourcen werden in diese Transaktion aufgenommen. Nur Klienten mit der Rolle **admin** dürfen diese Methode ausführen. Als Beispiel für die Anbindung an eine Weboberfläche dienen hier ein Eingabefeld und ein Ausführungsknopf:

```
<h:form>
  Kennung:
  <h:inputText value="#{verwaltungJSF.kennung}"> <br/>
  <h:commandButton value="Löschen"
    action="#{verwaltungJSF.loescheBenutzer}"/>
</h:form>
```

Der Name **verwaltungJSF** bezeichnet ein von JSF verwaltetes Java-Objekt. Wird der Knopf gedrückt, dann erhält das Attribut **kennung** den Wert aus dem Eingabefeld und die Methode **loescheBenutzer()** wird ausgeführt. Diese delegiert den Aufruf an die EJB:

```
class VerwaltungJSF {
  private String kennung;
  public void setKennung(String kennung) {
    this.kennung = kennung;
  }
  public String loescheBenutzer() {
    Context ctx = new InitialContext();
    Benutzerverwaltung bean = (Benutzerverwaltung)
    ctx.lookup(BENUTZERVERWALTUNG_JNDI_NAME);
    bean.loescheBenutzer(kennung);
    return "geloescht";
  }
}
```

Abbildung 1 zeigt diese Architektur, die als Standardarchitektur in JEE an die bekannte Dreischichtenarchitektur angelehnt ist. Wie an dem obigen Beispiel erkennbar, besteht beim Einsatz von JSF die GUI-Logik im Wesentlichen aus einer Delegation an die zugehörige EJB. Daher stellt sich die Frage, warum das Framework diese Delegation nicht selber durchführt. Genau das ist der Ansatz von Seam.

Seam ermöglicht es, in der Formulardefinition direkt auf die EJB zuzugreifen:

```
<h:form>
  Kennung:
  <h:inputText value="#{verwaltung.kennung}"> <br/>
  <h:commandButton value="Löschen"
    action="#{verwaltung.loescheBenutzer}"/>
</h:form>
```

Damit Seam zur Laufzeit die EJB auflösen kann, muss sie dem Seam-Framework über einen Namen bekannt gemacht werden. Dies geschieht mit der Seam-Annotation **@Name**. Des Weiteren muss Seam in der Lage sein, an der EJB das Eingabedatum **kennung** zu setzen. Die EJB erhält dadurch einen Zustand und ihr Typ muss von **Stateless** auf **Stateful** geändert werden. Schließlich muss die Methode eine Zeichenkette zurück geben, die von JSF für die weitere Navigation benutzt wird:

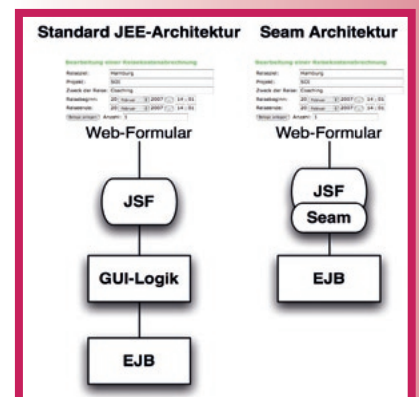


Abb. 1: Seam verzichtet auf die Vermittlungsschicht in der GUI

```

@Stateful
@Name("verwaltung")
public class BenutzerverwaltungBean implements Benutzerverwaltung {
    ...
    private String kennung;
    public void setKennung(String kennung) {
        this.kennung = kennung;
    }
    @TransactionAttribute(REQUIRED)
    @RolesAllowed("admin")
    public String loescheBenutzer() {
        ...
        return "geLoescht";
    }
}
    
```

Dieses Beispiel zeigt sowohl Vor- als auch Nachteile von Seam. Einerseits erleichtert es Seam dem Webentwickler, die Vorteile von EJBs zu benutzen. Andererseits verletzt es die Dreischichtenarchitektur von JEE, indem es die Logikschicht eng an die GUI-Schicht bindet. Wenn in einem Projekt neben einer Web-Oberfläche auch andere Kliententypen unterstützt werden sollen, dann muss dies in der Logikschicht beachtet werden. Es bietet sich eine Vierschichtenarchitektur an, bei der über der Kernlogik eine Adaptionsschicht für die unterschiedlichen Kliententypen zum Einsatz kommt.

Eine weitere elementare Eigenschaft von Seam, die hier deutlich wird, ist der Einsatz von zustandsbehafteten EJBs. Um dieses Thema zu beleuchten, ist zunächst ein kleiner Exkurs über die Verwaltung von Zuständen in Webanwendungen notwendig.

Verwaltung von Zuständen in Webanwendungen

HTTP ist ein zustandsloses Protokoll, d. h. eine Anfrage wird bearbeitet, ohne den anfragenden Klienten zu berücksichtigen. Typische Webanwendungen benötigen aber einen klientenabhängigen Zustand, z. B. für die Speicherung von Warenkörben. Daher unterstützt jedes Web-Framework ein Sitzungskonzept, bei dem für einen Klienten serverseitig Daten in einer Sitzung hinterlegt werden können.

Wenn eine Anwendung komplexer wird und die Anzahl der Anwender zunimmt, besteht die Gefahr, dass die Sitzungsdaten zu umfangreich werden. Der Webserver wird dann zunächst langsamer, bis er aufgrund von Speichermangel zusammenbricht. Der Entwickler hat also u. a. darauf zu achten, dass nicht mehr benötigte Daten aus der Sitzung gelöscht werden. Wenn die Webanwendung in einem Cluster betrieben wird, ergeben sich zusätzliche Probleme, da die Sitzungsdaten repliziert werden müssen. Web-Server unterstützen zwar Replikation, aber ein Webentwickler benötigt einiges an technischer Kompetenz und an Kodierungsdisziplin, um Fehler zu vermeiden.

Von JEE-Experten wird für die Verwaltung von Sitzungsdaten der Einsatz von zustandsbehafteten EJBs empfohlen. Diese sind mit geringem Mehraufwand clusterfähig und der Applikationsserver hat die Möglichkeit, unter Hochlast selten benötigte Sitzungsdaten in den Hintergrundspeicher auszulagern. Hierfür ist aber wiederum EJB-Kompetenz notwendig. Auch löst der Einsatz von EJBs nicht das Problem, unbenötigte Daten aus der Sitzung zu entfernen. Er entschärft es nur etwas durch die Auslagerung.

Seam löst diese Probleme durch zwei Konzepte. Erstens wird der Zugriff auf EJBs stark vereinfacht. Zweitens führt Seam das Konzept eines Arbeitsablaufs ein, die sogenannte *Konversation*. Eine Konversation ist ein JSF-Kontext, der zwischen dem Sitzungskontext und dem Anfragekontext angesiedelt ist. Daten in einer Konversation sind also immer einem Klienten zugeordnet, überdauern aber mehrere Benutzerinteraktionen.

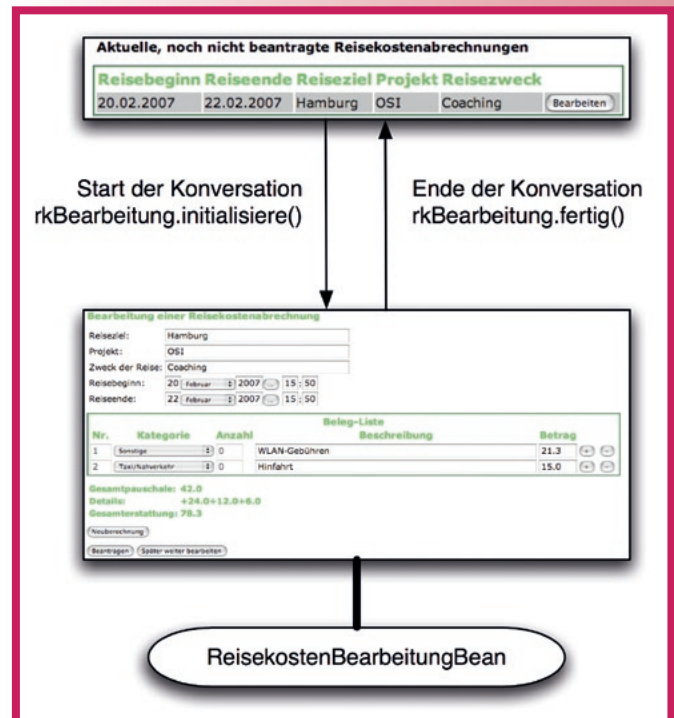


Abb. 2: Mit Konversationen lassen sich Sitzungsdaten natürlich verwalten

Abbildung 2 zeigt ein Beispiel. Der Anwender wählt aus einer Tabelle einen Eintrag zur weiteren Bearbeitung aus. Die Anwendung startet eine neue Konversation, in der die Bearbeitungsmaske ausgeführt wird. Die EJB `ReisekostenBearbeitungBean` verwaltet den Zustand der Maske. Ist die Bearbeitung abgeschlossen, wird die Konversation beendet. Seam entfernt automatisch alle enthaltenen EJBs.

Der Entwickler startet und beendet Konversationen mit Annotationen, die den Methoden hinzugefügt werden:

```

class ReisekostenBearbeitungBean implements ReisekostenBearbeitung {
    @Start
    public String initialisiere() {
        ...
    }
    @End
    public String fertig() {
        ...
    }
}
    
```

Konversationen ermöglichen damit eine zuverlässige Verwaltung von Sitzungsdaten. Da eine Konversation einen eigenen Namensraum darstellt, kann ein Benutzer gleichzeitig mehrere Operationen in unterschiedlichen Browserfenstern ausführen, ohne dass diese sich gegenseitig beeinflussen. Seam bietet vielfältige weitere Möglichkeiten zur Verwaltung von Konversationen an, mit denen eine Konversation auch unterbrochen und später wieder aufgenommen werden kann. Die bisherige Erfahrung zeigt allerdings, dass man sich zunächst auf einfache Benutzungsmuster beschränken sollte.

Zugriff auf Geschäftsobjekte

Eine primäre Funktionalität von Geschäftsanwendungen besteht aus der Bearbeitung der persistenten Geschäftsobjekte. Abbildung 3 zeigt die JEE-Standardarchitektur. Nur

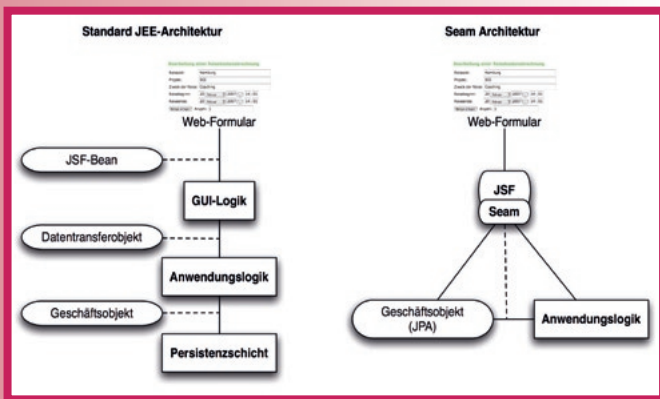


Abb. 3: Verzicht auf redundante Datenklassen in Seam

die Anwendungslogik greift direkt auf Geschäftsobjekte zu. Die GUI-Logik erhält die für die jeweilige Operation relevanten Daten verpackt in Datentransferobjekte. Diese sind dann für das jeweilige GUI-Framework anzupassen. Bei JSF müssen die benötigten Daten in Objekte kopiert werden, die von JSF verwaltet werden.

Diese Architektur wurde für verteilte Client/Server-Systeme und für ältere Persistenzmechanismen konzipiert. Die GUI-Logik einer JEE-Webanwendung läuft im Allgemeinen in dem gleichen Adressraum wie die Anwendungslogik, sodass keine Verteilungsgrenzen zu überschreiten sind. Moderne Persistenzmechanismen, wie z. B. die JPA, ermöglichen es, Assoziationen erst bei der Anfrage nachzuladen, sodass es nicht notwendig ist, für jede einzelne Operation ein maßgeschneidertes Datentransferobjekt zu erstellen. Daher ist diese JEE-Standardarchitektur für Webanwendungen zu kompliziert und so in der Realität auch selten anzutreffen.

Seam ermöglicht es, in den Masken der Webanwendung direkt auf die Geschäftsobjekte zuzugreifen. Der Entwicklungsaufwand reduziert sich dadurch erheblich. In dem in Abbildung 2 skizzierten Beispiel wird eine Reisekostenabrechnung in einem Formular editiert. Damit Seam auf das dahinter stehende persistente Geschäftsobjekt zugreifen kann, muss dieses wie bei den anderen EJBs mit einem Namen versehen werden:

```
@Entity
@Name("rkAntrag")
public class Reisekostenantrag {
    @Length(min = 3, max = 50)
    private String reiseziel;
    ...
}
```

Das JSF-Formular greift über diesen Namen direkt auf die Attribute des Geschäftsobjekts zu:

```
<s:validateAll><s:decorate>
<h:inputText value="#{rkAntrag.reiseziel}"/>
...
</s:decorate></s:validateAll>
```

Der aktuelle Wert des Attributs wird dadurch automatisch dargestellt und Änderungen des Anwenders werden automatisch in das Geschäftsobjekt und in die Datenbank übernommen. Die beiden Elemente `<s:validateAll>` und `<s:decorate>` sorgen für eine Validierung der Eingabedaten anhand der Annotationen am Geschäftsobjekt. Dadurch muss der Entwickler die Validierungsinformationen nur an einer Stelle pflegen. Setzt man Hibernate als JPA-Implementierung ein, wird die Validie-

rung zusätzlich automatisch vor dem Schreiben in die Datenbank durchgeführt.

Offen ist aber noch die Frage, woher die Anwendung weiß, welches Geschäftsobjekt nun konkret zu bearbeiten ist. Im Beispiel in Abbildung 2 wählt der Benutzer seinen Antrag aus einer Tabelle aus. Der Name der Geschäftsobjektklasse reicht für die Selektion des Objekts selber natürlich nicht aus. Dies ist Aufgabe der Anwendungslogik. Die Initialisierungsmethode `rkBearbeitung.initialisiere()` muss zusätzlich zum Starten einer Konversation das Geschäftsobjekt aus der Datenbank laden und Seam zur Verfügung stellen. Um die Selektion aus der Datenbank zu bestimmen, benötigt die Methode den Index der ausgewählten Tabellenzeile. Diesen erhält sie von Seam mit der Annotation `@DataModelSelectionIndex`. Für das Laden des Objekts aus der Datenbank werden die Standard-JPA-Mechanismen eingesetzt. Um das Objekt dann Seam bekannt zu machen, wird dieses mit einer `@Out`-Annotation versehen:

```
class ReisekostenBearbeitungBean implements ReisekostenBearbeitung {
    @DataModelSelectionIndex
    private int ausgewaehlteTabellenzeile;
    @In
    private EntityManager entityManager;
    @Out(scope=CONVERSATION)
    private Reisekostenantrag rkantrag;
    @Start
    public String initialisiere() {
        long id = bestimmeAntragsId();
        rkantrag = em.find(Reisekostenantrag.class, id);
    }
    ...
}
```

Der Zugriff auf das JPA-Verwaltungsobjekt `entityManager` ist auch mit der Standardannotation `@PersistenceContext` möglich. Seam bietet aber auch ein eigenes Verwaltungsobjekt. Dem Entwickler bieten sich dadurch unterschiedliche Alternativen für Transaktionen. Erstens kann eine Transaktion vom Anfang der Bearbeitung der Anfrage an die Webanwendung bis zur Erzeugung der Antwort dauern. Dadurch werden die Schreiboperationen in der gleichen Transaktion ausgeführt wie die Leseoperationen für die Erstellung der Antwort. Die Transaktion wird somit länger als notwendig. Sperren auf der Datenbank reduzieren die Skalierbarkeit und führen zu Zeitüberschreitungen und Abbrüchen.

Daher sollte im Allgemeinen die zweite Alternative eingesetzt werden. Bei dieser finden die Datenbankzugriffe für die Ausführung der Operation und die Zugriffe für die Erzeugung der Antwort in getrennten Transaktionen statt. Die obigen Probleme werden vermieden.

Die dritte Alternative besteht darin, eine Transaktion erst mit der Beendigung einer Konversation festzuschreiben. Dies ermöglicht z. B. einfach einen Assistenten zu entwickeln, der in mehreren Schritten Geschäftsobjekte manipuliert. Die Änderungen werden aber erst gespeichert, wenn der Anwender den Assistenten abschließt.

Dieses Beispiel verdeutlicht, dass Seam es dem Entwickler ermöglicht, sich auf die Fachlichkeiten zu konzentrieren, und ihn in die Lage versetzt, schnell Masken für die Verarbeitung von Geschäftsobjekten zu entwickeln und zu pflegen. Auf den ersten Blick scheint Seam damit die Dreischichtenarchitektur aufzubrechen, da die GUI direkt auf die Geschäftsobjekte zugreift. Dies ist aber nicht ganz richtig.

Die GUI benutzt nur reine Java-Objekte (Plain Old Java Objects, POJOs), sie greift nicht auf die JPA-Schnittstellen zu und sie manipuliert nicht die konkrete O/R-Abbildung. Ersteres wird nur von der Anwendungslogik getan, letzteres mittels



Annotationen nur in der Persistenzschicht. Die Schichtenarchitektur ist also immer noch gegeben. Änderungen in einer Schicht wirken sich nicht auf andere Schichten aus. Allerdings lässt sich, wie oben schon erwähnt, die GUI-Schicht nicht mehr einfach durch eine andere Technologie austauschen.

Was bietet Seam sonst noch?

Die dargestellten Fähigkeiten von Seam genügen schon für die Erstellung von Geschäftsanwendungen, welche die drei Haupttechnologien der JEE einsetzen. Seam bietet aber noch viele weitere Möglichkeiten für die Entwicklung von Webanwendungen, wie z. B. den Einsatz von Prozessen für die Seitennavigation und ein regelbasierter Ansatz für Zugriffsschutz. Da Seam auf JSF aufsetzt, kann Ajax-Funktionalität mit dem Framework `ajax4jsf` realisiert werden. Zusätzlich werden auch die Komponentenframeworks Rich Faces und ICEFaces unterstützt, welche Komponenten mit Ajax-Funktionalität enthalten.

Neben der Integration von EJBs, JSF und JPA unterstützt Seam noch weitere Technologien. Hierzu gehören insbesondere die Technologien der JBoss Enterprise Middleware Suite (JEMS), wie z.B. `jbpm` zur Entwicklung von Anwendungsprozessen und JBoss Rules für Geschäftsregeln. Es werden aber auch Technologien anderer Gruppen unterstützt, wie z.B. `iText` zur Erzeugung von Berichten in PDF oder eine umfassende Integration mit Spring. Zusätzlich werden Werkzeuge angeboten, um schnell CRUD-Anwendungen aufzubauen (Create, Retrieve, Update, Delete). Diese einfache Integration von weiteren Technologien, die für die Webentwicklung hilfreich sind, zeigt die Stärke der Architektur von Seam. Das Framework entwickelt sich aktuell mit großer Geschwindigkeit. Die Kernfunktionalität ist inzwischen ausgereift, die neueren Fähigkeiten sollten vor dem Einsatz in einem Projekt zuerst evaluiert werden.

Die technischen Anforderungen

Das Seam-Framework wurde für einen JEE-Applikationsserver konzipiert, der sowohl einen Web-Container, wie z. B. Tom-

cat, als auch einen EJB-3-Container umfasst. Erfolgreich getestet wurde Seam in den Applikationsservern WebSphere, WebLogic, Glassfish und JBoss. Für den Einsatz von Seam in einer Umgebung ohne EJB-3-Unterstützung, wie z. B. Tomcat, lässt sich der einbettbare EJB-3-Container von JBoss benutzen.

Es ist in Seam aber auch möglich, Webanwendungen ohne EJBs zu entwickeln. In diesem Fall werden einfache Java-Objekte eingesetzt. Die Vorteile von EJBs bzgl. Transaktionalität, Skalierbarkeit usw. können dann nicht genutzt werden. Statt der JPA unterstützt Seam auch Hibernate direkt.

Fazit

Seam bietet eine neue Architektur für Webanwendungen, die auf einfache Art den Einsatz von leistungsfähigen JEE-Technologien ermöglicht und so ein zuverlässiges, skalierbares und sicheres Fundament bietet. Formularorientierte Geschäftsanwendungen lassen sich schnell erstellen. Der Kern von Seam mit seiner Interaktion zwischen Komponenten und unterschiedlichen Kontexten ist einfach zu verstehen und zu benutzen und hinterlässt einen ausgereiften Eindruck. Die weiter reichenden Funktionalitäten, wie z. B. der Einsatz von Prozessen, benötigen aber eine deutliche Einarbeitungszeit.



Dr. Torsten Fink ist Leiter des JBoss Competence Centers der akquinet AG. Neben Koordinierungstätigkeiten führt er Architektur- und Technologieberatungen durch und unterstützt laufende JEE-Projekte. Er ist zertifizierter JEMS Middleware Expert.
E-Mail: torsten.fink@akquinet.de.



Weitere Informationsquelle

<http://labs.jboss.com/portal/jbosseam>