# Seam & Web Beans

Jozef Hartinger
JBoss QA Associate, Red Hat

Developer Conference
Sept 10th 2009

## Roadmap

- **Introduction**
- Seam Core
- Seam 3 & Web Beans
- Seam Extensions (by Ondřej Skutka)

## The Seam Framework

- Application framework for developing web applications based on Java EE standards
- Seam consists of:
  - Seam Core (uniform stateful component model)
  - JSF Extensions (pageflow, page parameters, Seam taglib) – influenced JSF 2.0
  - Extensions (jBPM support, Security, Mail, PDF, iText)
  - Tooling (seam-gen, JBoss Tools)
    - Do not miss hands-on lab tomorrow at 9:30

## A brief look into Seam history

- Project founded in September 2005 by Gavin King
- Version 2.0 released by the end of 2007
  - Hot deployment
  - Added many extensions
  - Added seam-gen
  - Added alternative presentation layers
- Latest stable version – 2.2.0.GA
- Version 3.0 in development
  - Radical changes in portability and modularity

## Roadmap

- Introduction
- **Seam Core**
- Seam 3 & Web Beans
- Seam Extensions (by Ondřej Skutka)

## Seam Component

- POJO, JPA entity class, Session Bean, Message-Driven Bean, Web Service Endpoint, Spring Bean
- Metadata (annotations or XML configuration)
- Its instances are provided with:
  - lifecycle management (creation of beans and invocation of lifecycle methods, keeping component reference in context variable)
  - services like bijection and declarative security
  - name

## Seam Component Example

```
@Name("user")
public class User
{
   private String name;
   // some code
}
```

@Name identifies the class as a Seam component

- This component is stored in context variable named "user" and is also accessible from JSF page via EL

```
<h:inputText id="username" value="#{user.name}" />
<h:commandButton type="submit" id="update" value="Update" action="#{userDao.update}"/>
```

## Seam Component Example

```java
@Name("cart")
@Scope(CONVERSATION)
public class ShoppingCart {

   @In private User user;

   @Create
   public void create() {
      // some code
   }

   @Begin
   public void checkout() {

   }
```

## Instantiation of Seam component

- New class instance is created
- Initial property values are applied (static dependency injection)
- @PostConstruct / @Create method is invoked
- Reference to the component instance is stored in a context variable

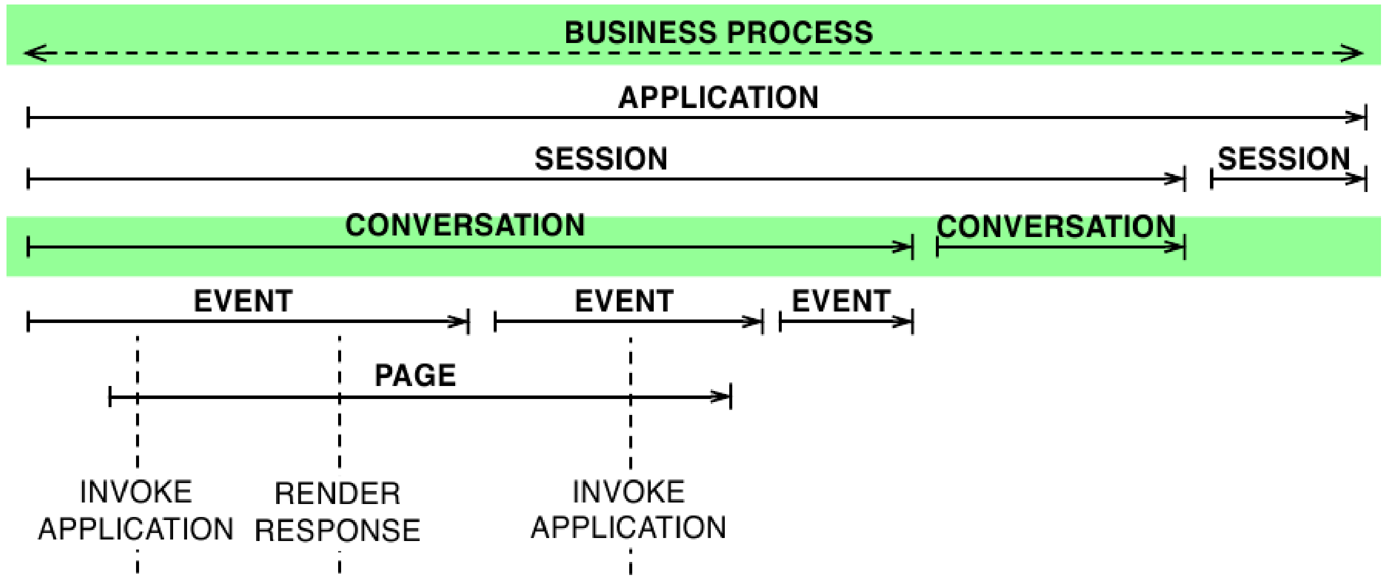
## Seam Component Example

```
@Name("cart")
@Scope(CONVERSATION)
public class ShoppingCart {

    @In private User user;

    @Create
    public void create() {
        // some code
    }

    @Begin
    public void checkout() {

    }
```

Lifecycle callback – @Create method will be called during component instatiation

## Seam Contexts – state matters

- Event
- **Conversation**
- Application
- **Page**
- Session
- **Business Process**



## Specifying a scope of a component

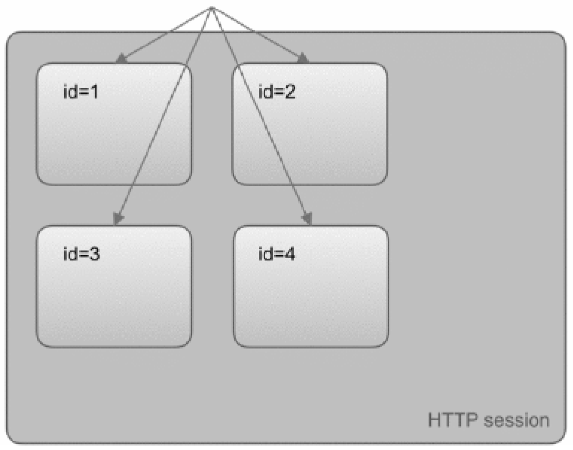
```
@Name("cart")
@Scope(CONVERSATION)
public class ShoppingCart {

    @In private User user;

    @Create
    public void create() {
        // some code
    }

    @Begin
    public void checkout() {

    }
```

@Scope specifies context variable in which the component instance is stored once created

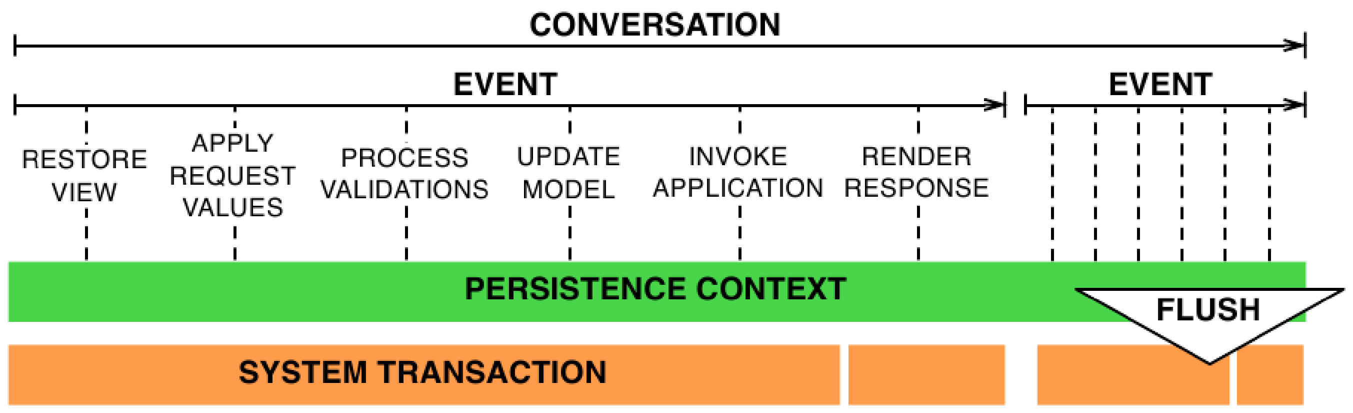
## Conversation

- Context holding state of several interaction needed to achieve a goal
- Bridges the gap between HTTP requests
- This state would normally be held in session – memory leaking, no multitab functionality
- Conversation examples:
  - Wizard for adding a user
  - Hotel booking process (view hotel -> check availability -> enter user information -> confirm booking )
  - Shopping cart checkout (review -> address -> payment information)

## Conversation



- Multiple conversations can be active during single session
  - Each for every browser window
- Persistence context remains active for the entire conversation – entities stay managed



## Seam Component Example

```
@Name("cart")
@Scope(CONVERSATION)
public class ShoppingCart {

   @In private User user;

   @Create
   public void create() {
      // some code
   }

   @Begin
   public void checkout() {

   }
```

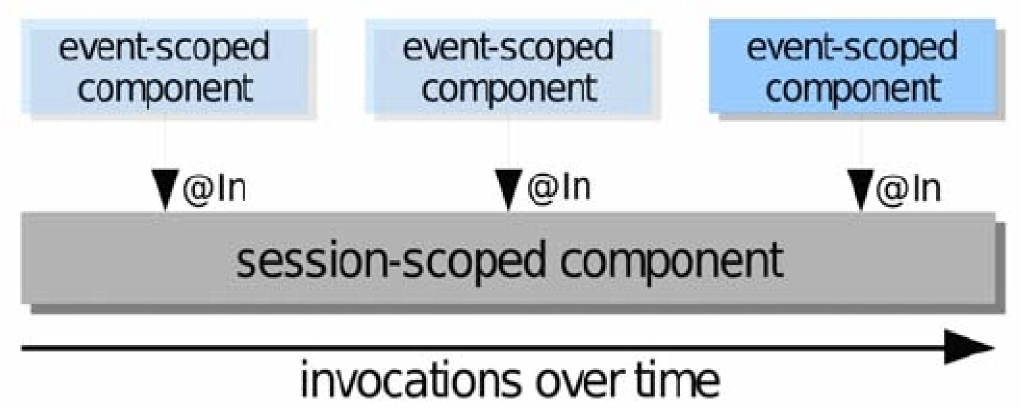
Invocation of @Begin method "starts" a long-running conversation

## Dependency Injection – usual approach

- Static – DI performed on component instantiation only
- One-way
- Non-contextual – injected beans are stateless

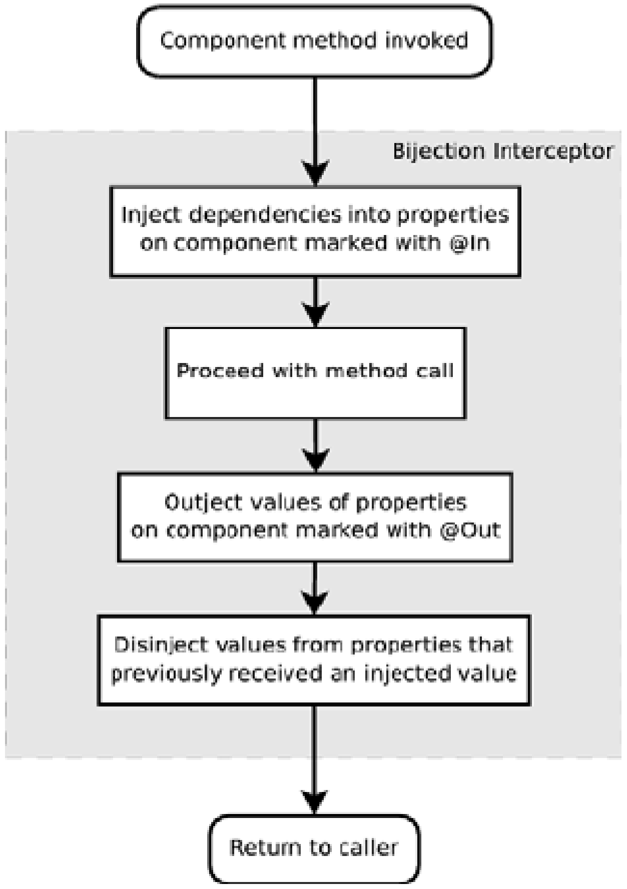
## Injection evolved – Dependency Bijection

- Dynamic – performed around every method invocation
  - Seam also provides static DI



- Bidirectional – Allows component to change the context state by outjecting a reference

```
@Out private User user;
```


Component method invoked
Bijection Interceptor
Inject dependencies into properties on component marked with @In
Proceed with method call
Outject values of properties on component marked with @Out
Disinject values from properties that previously received an injected value
Return to caller

## Injection evolved – Dependency Bijection

- Contextual
  - Container looks for matching context variable first
  - If not found, new component instance can be created and injected

## Declaring injection points

```
@Name("cart")
@Scope(CONVERSATION)
public class ShoppingCart {

    @In private User user;

    @Create
    public void create() {
        // some code
    }

    @Begin
    public void checkout() {

    }
```

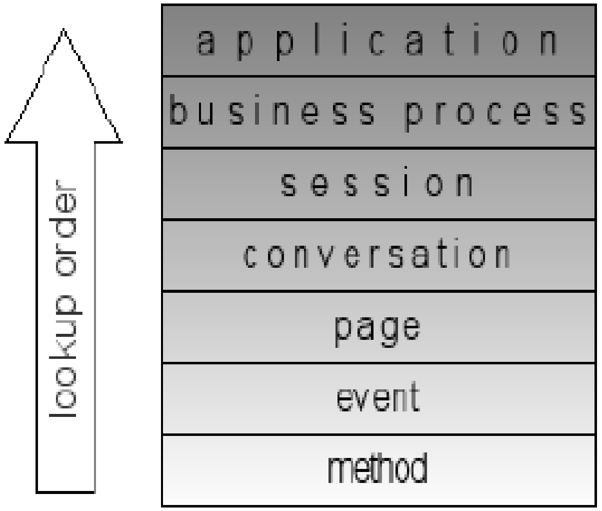
@In indicates dependency
Injection point

# Contextual lookup

- By default, the container looks for a context variable with name identical to the field name

```
@In private User user;
```

- Contexts are searched in the following order



- If no context variable is found, Seam creates a component instance if ordered to do so

```
@In(create=true) private User user;
```

## Roadmap

- Introduction
- Seam Core
- **Seam 3 & Web Beans**
- Seam Extensions (by Ondřej Skutka)

## The JSR-299 Specification

- Specification formally known as Webbeans
- Contexts and Dependency Injection for the Java EE Platform
- Still in development
- Part of Java EE 6
- Spec and reference implementation (Web Beans) are led by Red Hat
- Two alternative implementations (Apache OpenWebBeans and Resin)

## JSR-299 Contexts

- Improved lifecycle for stateful components, bound to well-defined contexts
  - Request
  - Conversation
  - Session
  - Application
  - Dependent (pseudo context)

- Specification supports adding custom contexts (business process)

## Typesafe dependency injection

- Dependency injection based on String identifiers is not perfect

```
@In private User user;
```

  - Issues discovered at runtime

- JSR-299 dependency resolution is based on Java type and Qualifiers

```
@Inject PaymentProcessor paymentProcessor;
```

  - Container is able to detect typesafe resolution issues (like unsatisfied and ambiguous dependencies) at initialization time

```
@Inject @Synchronous PaymentProcessor paymentProcessor;
```

## Qualifier

- Special annotation used to distinguish between various component instances and implementations of an interface

```
@Target( { FIELD, PARAMETER, METHOD, TYPE })
@Retention(RUNTIME)
@Qualifier
public @interface Synchronous
{

}
```

## Typesafe dependency injection

- The following bean:

```
@Synchronous
public class SimplePaymentProcessor
   implements PaymentProcessor
{
   //some code
}
```

- is suitable for injection to this injection point

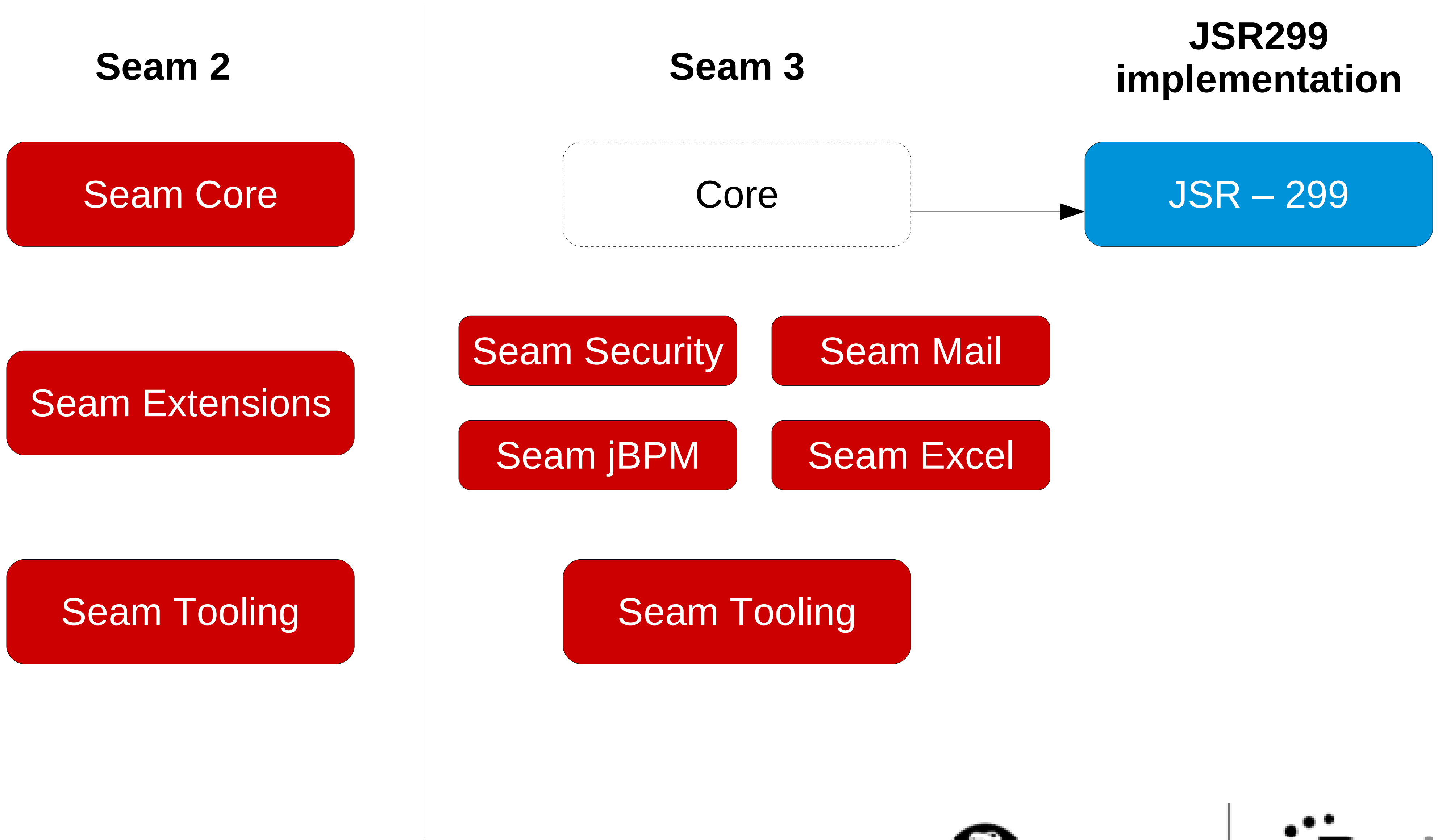
```
@Inject @Synchronous PaymentProcessor paymentProcessor;
```

- because it satisfies both Java type and qualifier requirements

## Other features

- Interaction via an event notification facility
- Better approach to binding interceptors to components, along with a new kind of interceptor, called a decorator, that is more appropriate for use in solving business problems
- Extensibility

## Seam 2 vs Seam 3



## Roadmap

- Introduction
- Seam Core
- Seam 3 & Web Beans
- **Seam Extensions (by Ondřej Skutka)**