Open containers and dynamic adaptability of services in the EJB model

Julien Blass

julien.blass@emn.fr

Ecole des Mines de Nantes

Ecole Polytechnique de l’Université de Nantes

Supervisors: Shigeru Chiba, TITECH (Japan)- Jacques Noyé, EMN (France)

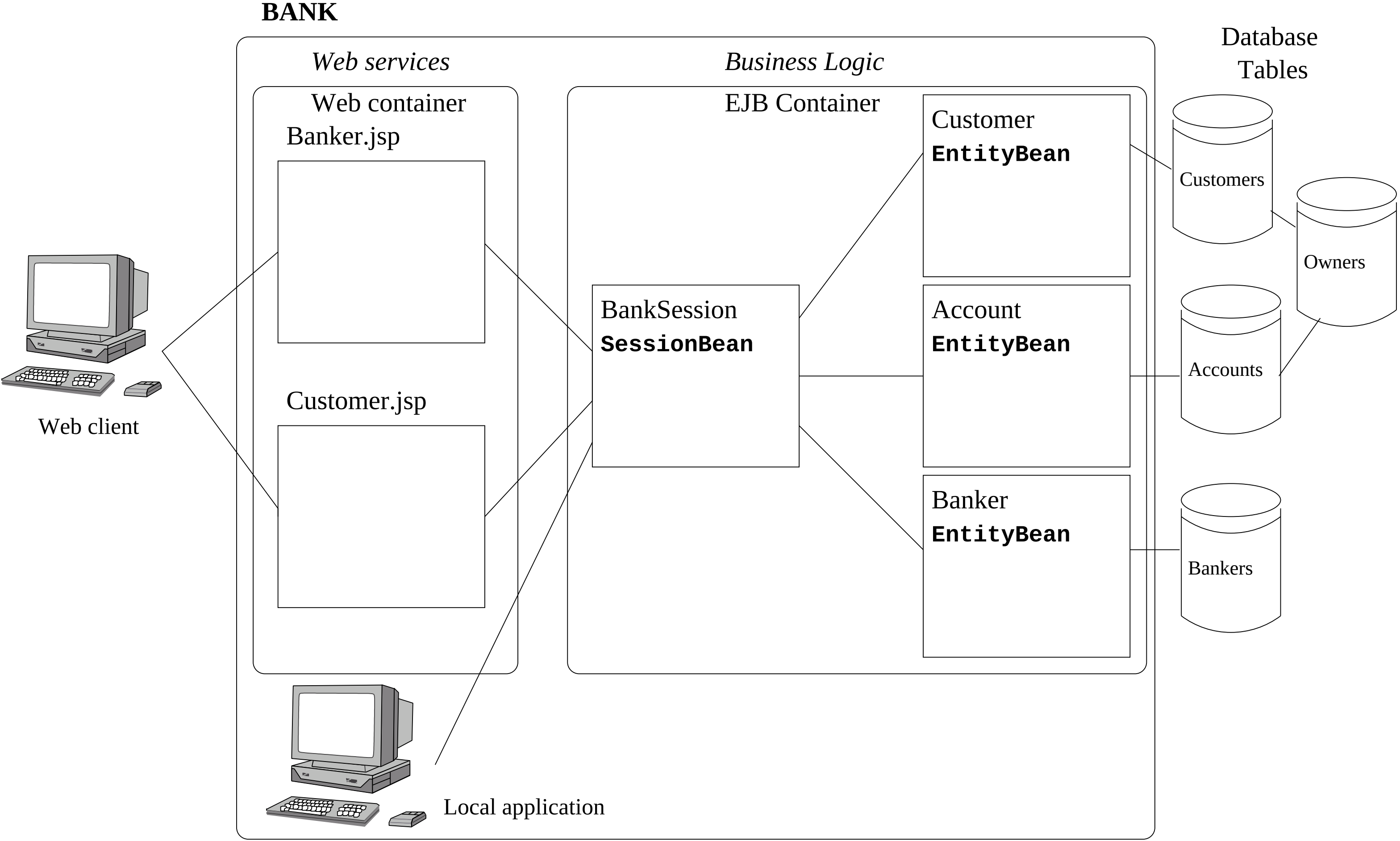
Motivation and goals

Some definitions

- Application servers
  - AS are component-based products providing middleware services.
- Containers
  - They encapsulate components and manages their execution. It is in charge of executing the services.
- Services
  - Low-level platform-specific functionnalities + high-level business functionnalities

An exemple of J2EE application

A simple bank application



Objectives of my work

- Issues of the present model
  - The present model is limited. No possibility to change/add/remove services provided by the EJB container at execution time.
- Goal
  - Proposing an open container model for EJB-based applications.

Related work

- Classical EJB platforms
  - Usually, no way to integrate new services.
- Models of adaptable and extensible components
  - Open ORB Python Prototype, JavaPOD, JAC (Java Aspect Components), . . .
- Models of adaptable and extensible EJB containers
  - Using a meta-object model on the interposition objects. No dynamic adaptability of services.

My work

My proposal

- Inserting hooks in the code at load time
- Triggering events at specific moment of execution
- Monitoring the application and catching events
- Analyzing events and performing actions

This model permits to satisfy the three mechanisms that define an open container architecture: *interception*, *coordination* and *control*.

Interception: “eventifying the code”

At load time, hooks are inserted in EJBs, using the Javassist library, in order to trigger events at specific moment of execution time.

Example: method `addCustomer` *of BankSession*

```
// Renaming the original method:
addCustomer(String name, String password)
   -> addCustomerOrg(String name, String password)
// Creating a new method:
boolean addCustomer(String name, String password) {
  EventCall evt = new EventCall(method info);
  boolean preResult = MONITOR.invoking(MONITOR.catching(evt));
  boolean result = MONITOR.catching(new
           EventReturn(preResult,method info));
  return result; }
```

Coordination: the monitors

- A monitor monitors a class.
- The monitor interface permits to add/remove dynamically a *Service*.
- *Services* are coordinated following the chain model.
- A service programmer can ask for a specific order in the services to a monitor.

Control: analyze and perform

The control mechanism is managed by the services.

Services are implemented by a *service developer*, according to a specific interface.

Services can:

- receive *extended events* from a monitor,
- *analyse* the information contained in an event,
- *perform* actions using this information
- store/get information in the object referred by the event (simulates the behaviour of an object monitor)

A simple example of service

```
public class TraceEventService implements Service {
  ...
  public ExtendedEvtCall performing(ExtendedEvtCall eevt)
   throws ServicePerformingException {
     out.println("--> Method call of "+eevt.getEvent().getSrcName());
     return eevt;
  }
  public ExtendedEvtReturn performing(ExtendedEvtReturn eevt)
   throws ServicePerformingException {
     out.println("<-- Method return of "+eevt.getEvent().getSrcName());
     return eevt;
  }
  ...
}
```

Remaining tasks

A remaining issue

Cooperation of monitors

Offering the possibility for monitors to cooperate would permit to develop entities monitoring the whole application.

Solutions:

- Using a meta-meta-level: eventifying services and monitoring their activities. Costly: implies decreasing performance.
- Using a same service for many class monitors. Limited and less powerfull than the first solution.

Remaining work

- Integrating the tools I developed for this model into JOnAS
  - Hooking the EJBs at load time, when the server loads the beans
  - Using JNDI to register the monitors
  - Load/Unload the services using the JAdmin tool
- Evaluating the performances of the model