Seminar on

J2EE Patterns

„Application and System Level Patterns“

Walter Kriha

Goals

- Learn application level patterns like value objects, composite entity etc. to avoid performance problems
- Take a look at how J2EE itself can be extended e.g. through the Connector Architecture

The application level patterns are also called „J2EE best practices patterns“. They are mostly adaptations or straight uses of the GOF patterns.

Roadmap

1. Show the architectural forces in a distributed multi-tier environment
2. Show some specific problems and look for patterns.
3. Learn the J2EE pattern catalog
4. Discuss selected patterns
5. Take a look at the connector architecture
6. Resources: Hints and tips for J2EE.

A current student project e.g. tries to model composite objects with EJBs.

Questioning entity beans

Early EJB designs followed the „domain model“ of design: Heavyweight business objects included business logic and were also designed to be persisted through some OO-relational mapper. Lately the „service model“ of design has been favored with relatively dumb entity beans and the business logic contained in session facades. „Bitter EJB“, by the author of „Bitter Java“ raises some important questions about entity beans and whether they are still useful. The author believes that the domain model would lead to a cleaner and better maintainable architecture – which is somewhat questionable as well. Some other issues:

-with transactions and security done by session facedes, what‘s left for Entity Beans besides persistence? And couldn‘t this be provided by a better mechanism (topcad etc.)?

-Making Entity beans both remote and local just confuses distributed computing with local computing and falls into all the traps that Jim Waldo mentions in his famous „note on distributed computing“.

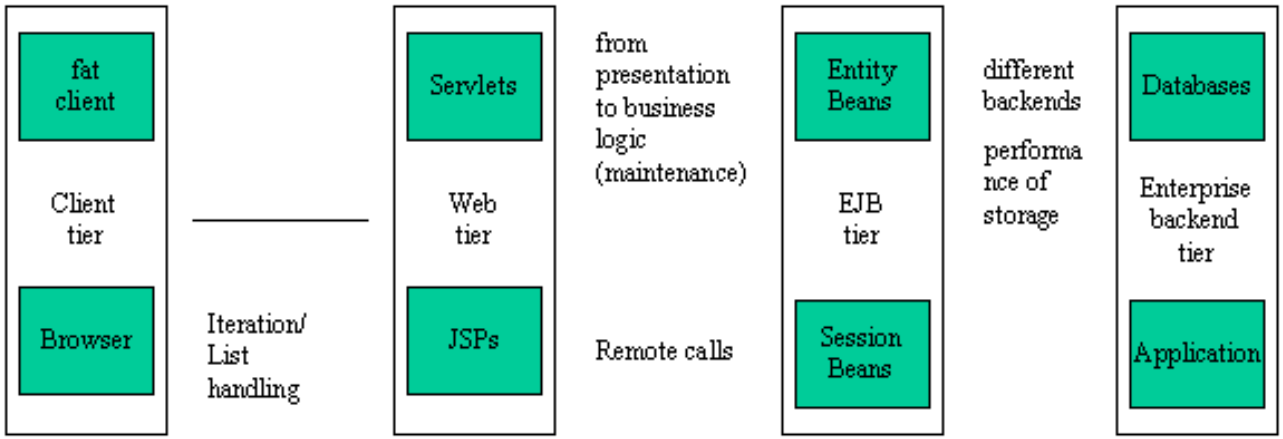
You can find selected chapters of „Bitter EJB“ on www.theserverside.com. In many ways it reflects my own experiences with so called business objects – a concept that was never easy to define or explain. I guess the result is that those heavyweight business objects are simply overloaded with functionality.

Architectural Forces of J2EE

- Distributed Environment (naming, finding etc.)
- Different physical architectures possible
- Persistence of important business objects
- Different user interfaces and channels
- Different backend systems
- Limits of components as not being programming language objects (inheritance in EJBs etc.)

These forces result in endless possibilities to create applications which are either too slow or cannot be maintained.

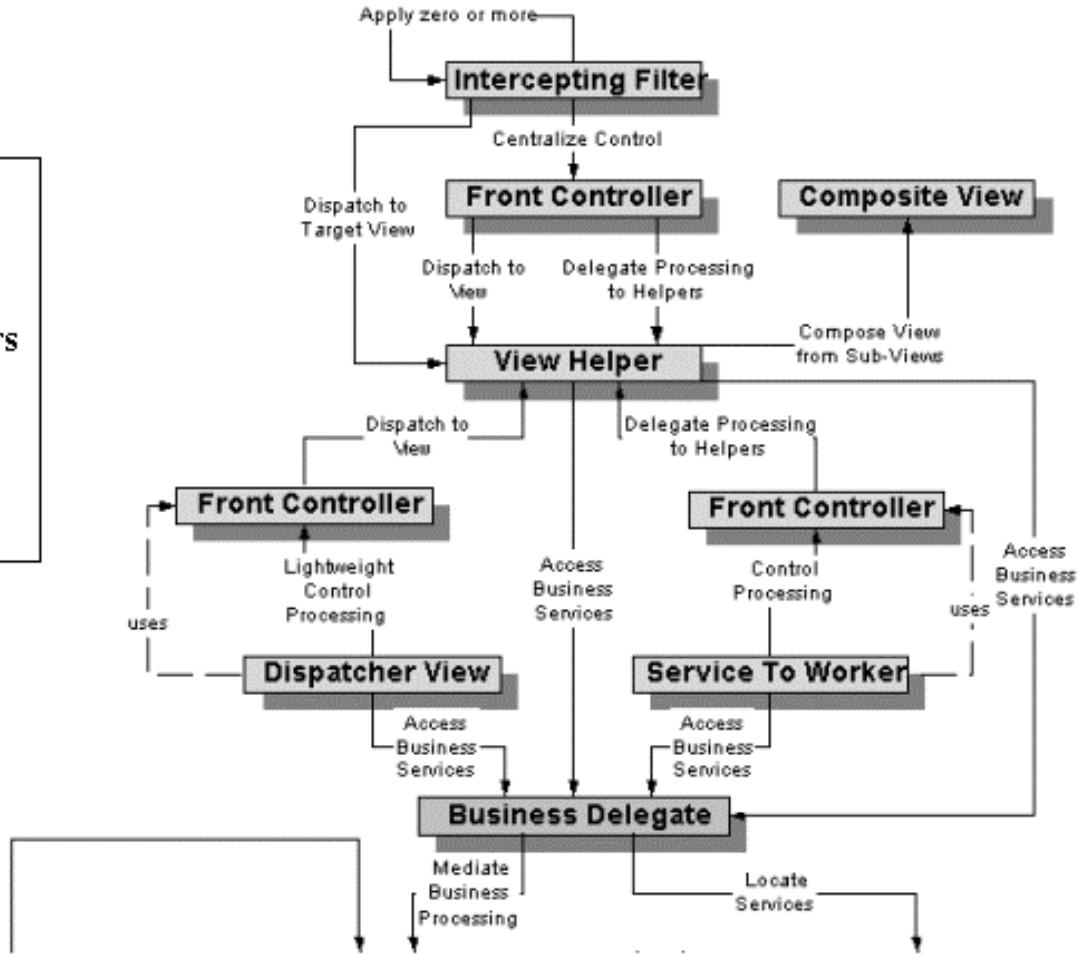
Basic physical components of J2EE



Each interface between tiers has unique problems. E.g. can the number of remote calls slow down the applicatoin. If too many internal interfaces are exposed to clients we get an application that cannot be changed anymore. Large amounts of objects stored in a relational DB can also become a performance problem.

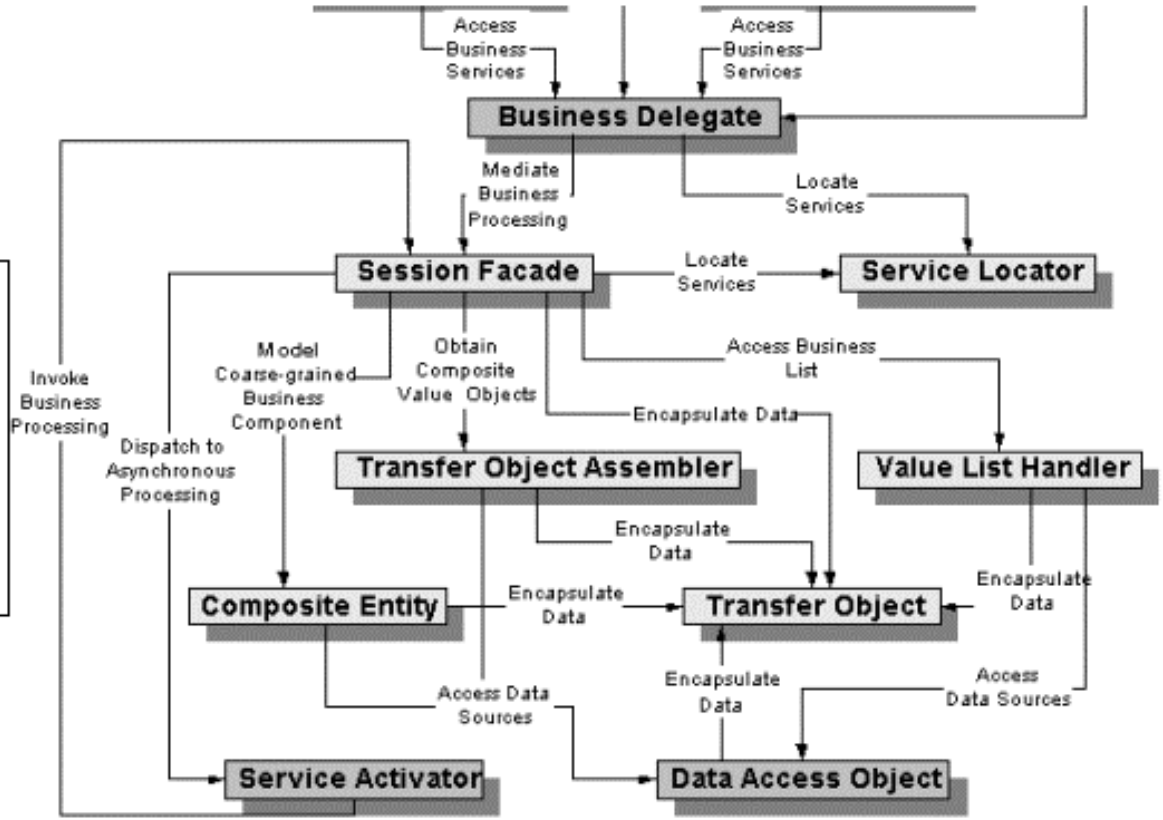
The J2EE Pattern Catalog (1)

The upper half of the pattern catalog covers mostly presentation and initial access



The J2EE Pattern Catalog (2)

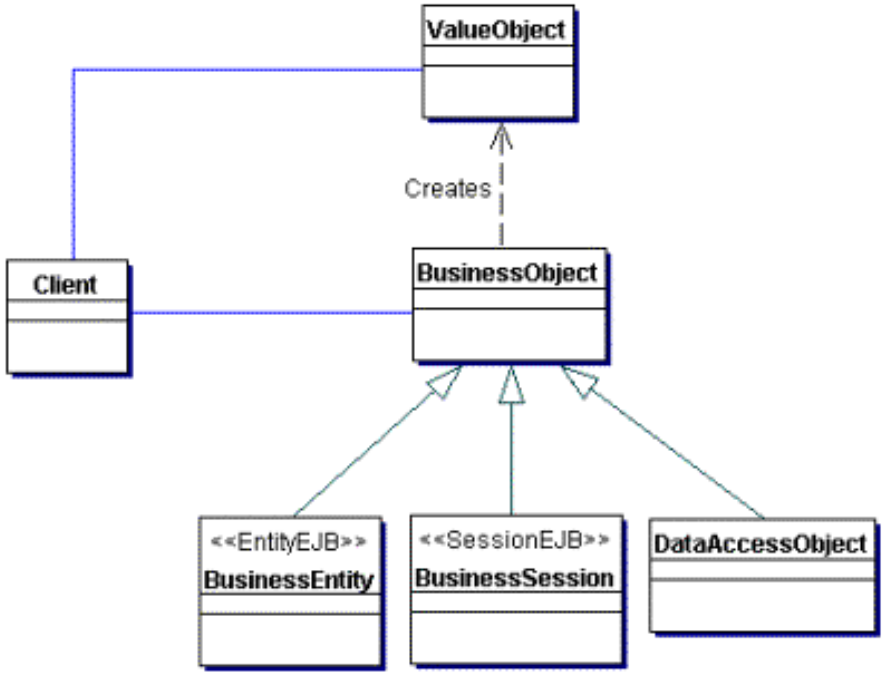
The lower half of the pattern catalog covers mostly business logic, storage and data transfer issues



Transfer/Value Object Pattern

Forces : remote bean acces expensive, usually more attributes of an object needed.

Solution: Use a Transfer Object to encapsulate the business data. A single method call is used to send and retrieve the Transfer Object. When the client requests the enterprise bean for the business data, the enterprise bean can construct the Transfer Object, populate it with its attribute values, and pass it by value to the client.

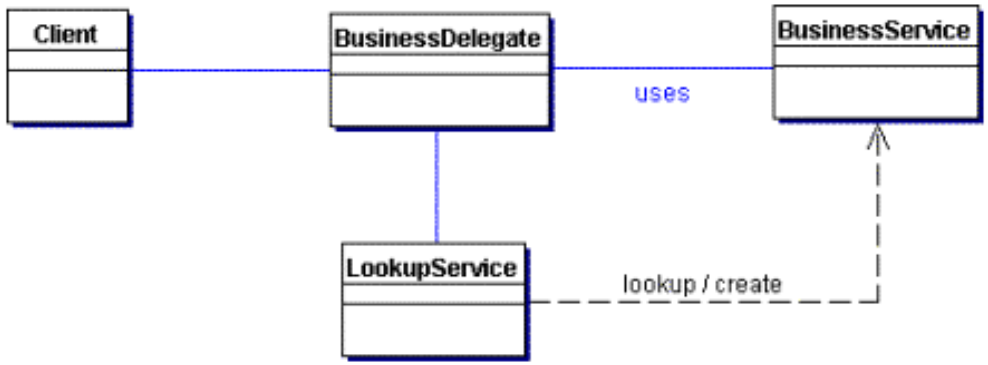


Value objects can be generic (composite message pattern) or domain specific (in this case they should be generated if possible). An interesting use is a partial value object that can prohibit access to certain fields for certain users. (http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html)

Business Delegate Pattern

Forces: Presentation-tier clients need access to business services but should not learn the internals of a business service (how to find which objects). It is desirable to reduce network traffic between client and business services.

Solution: Use a Business Delegate to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service, such as lookup and access details of the EJB architecture.

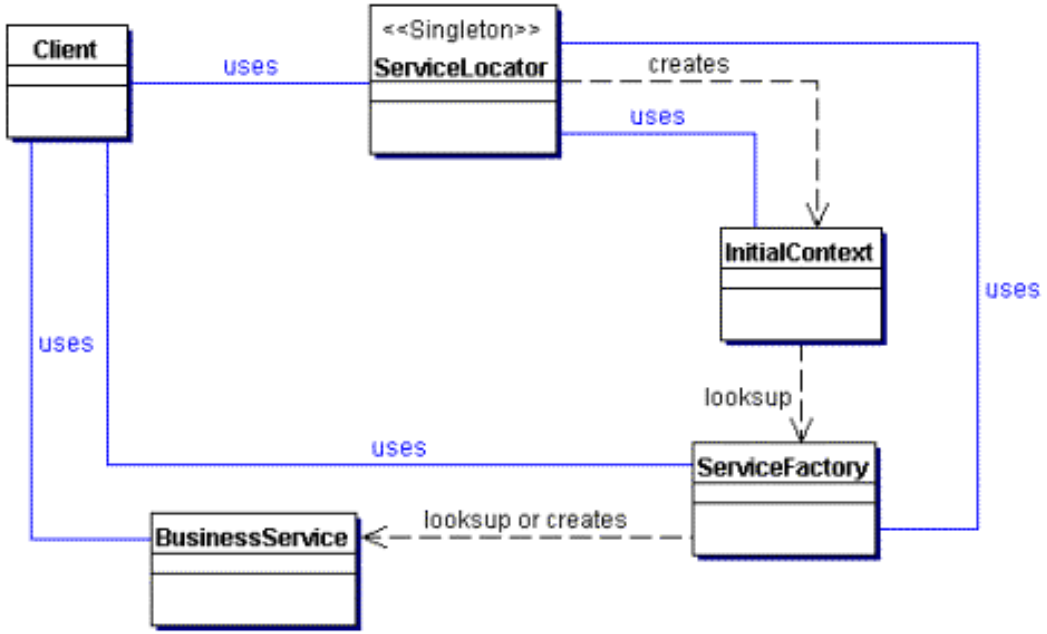


Business Delegate is related to session facade which also hides internals. The major point here is to allow the business logic to evolve and change without affecting clients. Facedes can also enforce a common way to access a system. Mainframe transaction sytems use those facades as general entry mechanism to the system.

Service Locator Pattern

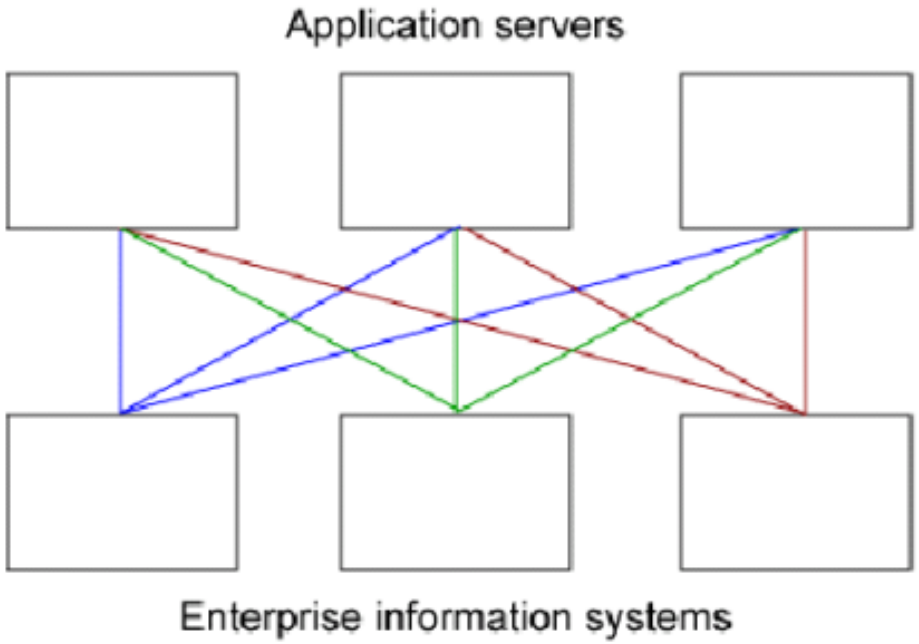
Forces: Lookup of EJB or J2EE objects is expensive and tedious. JNDI needs to be asked for objects references, finders used to find homes and homes finally used to create objects. No caching of factories.

Solution: Use a Service Locator object to abstract all JNDI usage and to hide the complexities of initial context creation, EJB home object lookup, and EJB object re-creation. Multiple clients can reuse the Service Locator object to reduce code complexity, provide a single point of control, and improve performance by providing a caching facility.



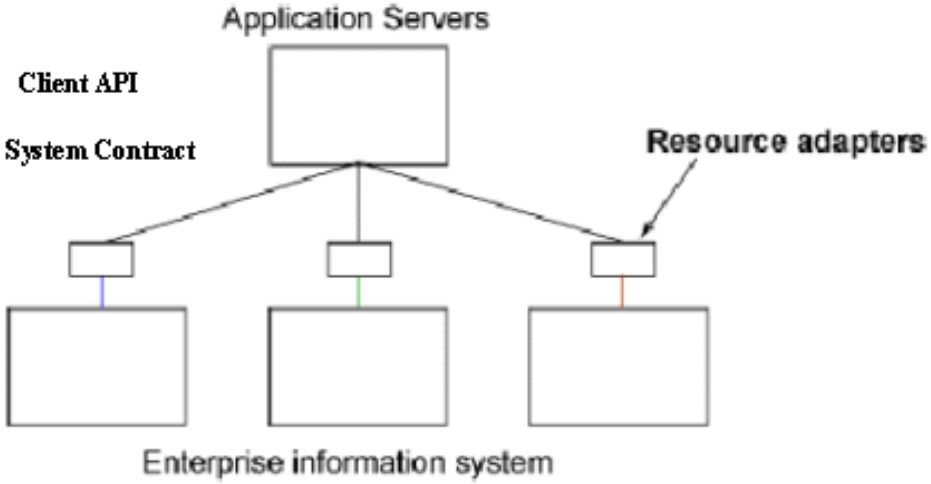
Service locator is simply a convenience pattern that allows caching of repeatedly uses factories. Once you have created you first EJB „hello-world“ you know how tedious the creation of objects through JNDI/finders/factories can be.

J2EE Architecture Patterns



A system like J2EE uses a lot of patterns internally. We will look at one example: The Java Connector Architecture. It solves the problem of integrating different backend systems with application servers. from: Will Farrell, Introduction to the J2EE Connector Architecture, www.ibm.com/developerworks

Main JCA Components



Resource adapters are usually written by EIS providers. They need to guarantee the system contract with the application. In other words: the system contract defines a protocol of interfaces that the resource adapter needs to implement. A client API is only a higher level interface that allows a more convenient access to the backends. The system contract includes flow of transaction, security and pooling information.

System contracts

System contracts define the connection between the application server and the EIS. The EIS side of the system contract is implemented by a *resource adapter* -- a system-level software driver specific to the EIS. The application server and the resource adapter collaborate by means of the system contract to provide secure, robust, scalable access to the EIS.

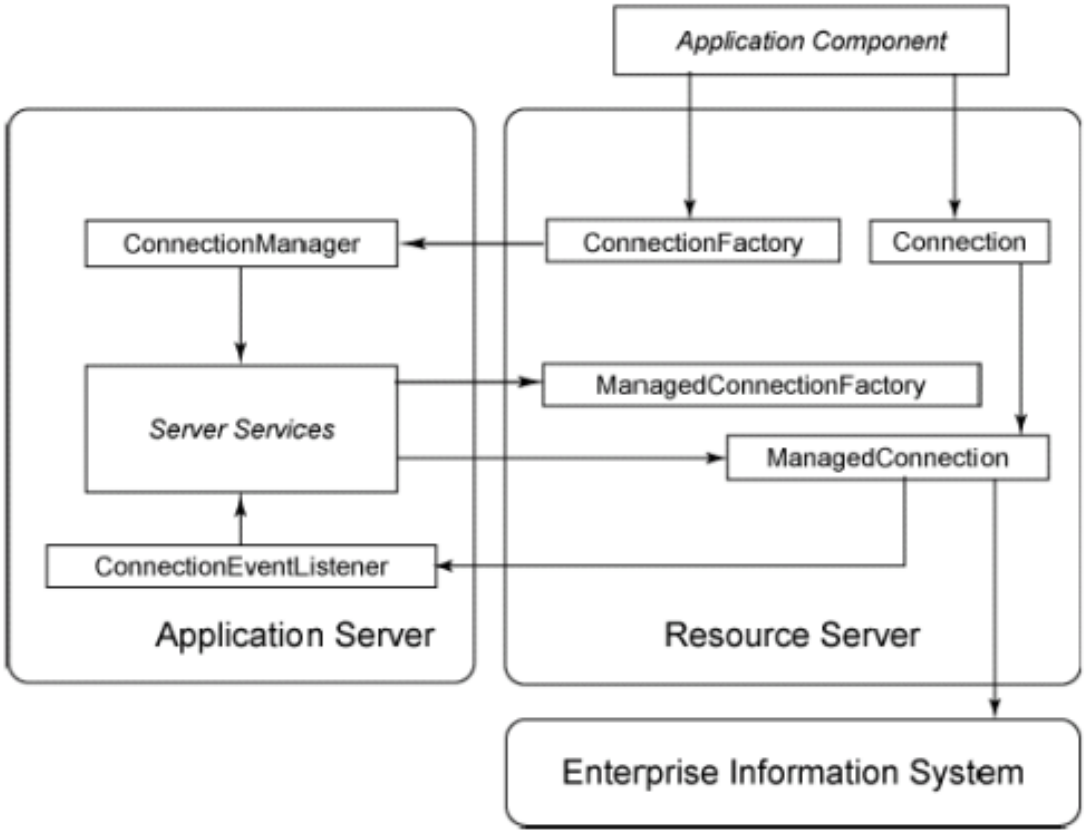
Three types of system contracts are defined:

- The *connection management* contract enables physical connections to the EIS and provides a mechanism for the application server to pool those connections.
- The *transaction management* contract supports access to an EIS in a transactional context. Transactions can be managed by the application server, providing transactions that incorporate other resources besides the EIS, or they can be internal to the EIS resource manager, in which case no transaction manager is required.
- The *security* contract supports secure access to the EIS.

Will Farrell, Introduction to the J2EE Connector Architecture, www.ibm.com/developerworks

System Contract Architecture

The application server can intercept calls to the resource adapter because the RA objects implement app. server interface (template/hook pattern). Clients do not get real connection objects, only proxies to managed connection objects in the RA.



Installing Connectors

connector with implementation classes and deployment descriptor

deploy code:

```
...
public static void main(String[] args) throws NamingException {

 Properties properties = new Properties();
 properties.put(Context.INITIAL_CONTEXT_FACTORY, INITIAL_CONTEXT_FACTORY);
 InitialContext context = new InitialContext(properties);
 HelloWorldInteractionSpecImpl ispec = new HelloWorldInteractionSpecImpl();
 ispec.setFunctionName(HelloWorldInteractionSpec.SAY_HELLO_FUNCTION);
 context.bind("jca/HelloWorldISpec", ispec);
}
...
```

Depending on your tooling you need not write this code by yourself. A deploytool will use information from the deployment descriptor to automatically install your new connector.

Using Connectors

client lookup code:

```
InitialContext context = new InitialContext();
ConnectionFactory cxFactory =
   (ConnectionFactory) context.lookup("java:comp/env/HelloWorld");
RecordFactory recordFactory = cxFactory.getRecordFactory();
IndexedRecord input =
   recordFactory.createIndexedRecord(HelloWorldIndexedRecord.INPUT);
IndexedRecord output =
   recordFactory.createIndexedRecord(HelloWorldIndexedRecord.OUTPUT);
```

First a ConnectionFactory needs to be found. Then special connections can be created from it. All lookup is done through JNDI – a good example how a naming service decouples clients and service providers. BTW: there is a nice JNDI browser available from sourceforge.com.

Resources (1)

- Adam Bien, J2EE Patterns, Entwurfsmuster für J2EE. Am besten einzelne pattern herausgreifen und bearbeiten.
- Adam Bien, Enterprise Java Frameworks, Das Zusammenspiel der Java-Architekturen. Einführung in Framework Technology am Beispiel J2EE
- Assorted links to J2EE patterns: http://www.javaworld.com/javaworld/jw-06-2002/jw-0607-j2eepattern.html and http://www.javaworld.com/javaworld/jw-01-2002/jw-0111-facade.html
- Improve your application's workflow with the Dispatcher design pattern and XSL http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-dispatcher.html?

Resources (2)

- Implement a Data Access Object pattern framework http://click.idg.email-publisher.com/maaah3RaaRlW0a9JUqkb/
- J2EE Architecture and Development Introduction to the J2EE Platform - JDC (Monica Pawlan) http://developer.java.sun.com/developer/technicalArticles/J2EE/Intro/index.html Still probably the best short introduction even though it is from the last year.
- Portals: http://java.sun.com/j2ee/ .The main portal to all of J2EE. especially useful is the page http://java.sun.com/j2ee/docs.html with links to all specifications (JMS, EJB, XML, etc., FAQs and SDKs.
- For quality articles on all serverside processing and free books on EJB patterns: www.theserverside.com

Resources (3)

- Introduction News & Articles A walking tour of J2EE http://www.javaworld.com/javaworld/jw-07-2001/jw-0727-enterprisejava.html J2EE Tutorial - jsc
- http://java.sun.com/j2ee/tutorial/ J2EE Platform Quiz – JDC
- http://developer.java.sun.com/developer/Quizzes/j2ee/ Writing J2EE Enterprise Apps – jsc
- http://developer.java.sun.com/developer/onlineTraining/J2EE/Intro/ J2EE: Developing Multi-Tier Enterprise Applications – JR
- http://www.javareport.com/html/from_pages/view_recent_articles_jr.cfm?ArticleID=689 Draft of J2EE Connector Architecture – jsc
- http://java.sun.com/aboutJava/communityprocess/review/jsr016/index.html http://www.cetus-links.org/oo_patterns.html

Resources (4)

- Books: What beginners need is an overview of the whole architecture and its components and the best practices and patterns needed to develop something. The J2EE architecture has enough complexity to warrant a design pattern driven approach. The whole (free) book on J2EE: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/index.html#chapters
- Professional Java Server Programming J2EE, 1.3 Edition by Subrahmanyam Allamaraju (Editor), et al Aimed at the working developer or IT manager tackling server-side and Web-based enterprise Java applications, Professional Java Server Programming J2EE 1.3 Edition offers a truly excellent guide to the fast-changing world of today's Java 2 Enterprise Edition (J2EE) APIs and programming techniques....
- Don't start with J2EE without design patterns for it: Core J2EE Patterns: Best Practices and Design Strategies by Deepak Alur, John Crupi, Dan Malks EJB Design Patterns: Advanced Patterns, Processes, and Idioms von Floyd Marinescu

Resources (5)

- Bitter EJB. Selected chapters at www.theserverside.com very good.