WAKESOFT ARCHITECTURE SERVER

An Introduction to Architecture Servers & Frameworks

Date last modified: January 8, 2002 12:17 pm

Wakesoft, Inc.
525 Brannan Street
Suite 404
San Francisco, California 94107
t 415.281.0300
www.wakesoft.com
support@wakesoft.com

TABLE OF CONTENTS

CHAPTER

1

Preface

What is an architecture server?

A common misconception of those new to J2EE is that J2EE by itself provides everything needed to build successful business applications.

The J2EE infrastructure does solve complex technological challenges; it supports implementation of transactions, distribution, and fault tolerance. However, J2EE focuses downward into the technology stack. J2EE abstracts the very low level technologies such as sockets and database vendor class libraries into a still-highly-technical Applications Programming Interface (API).

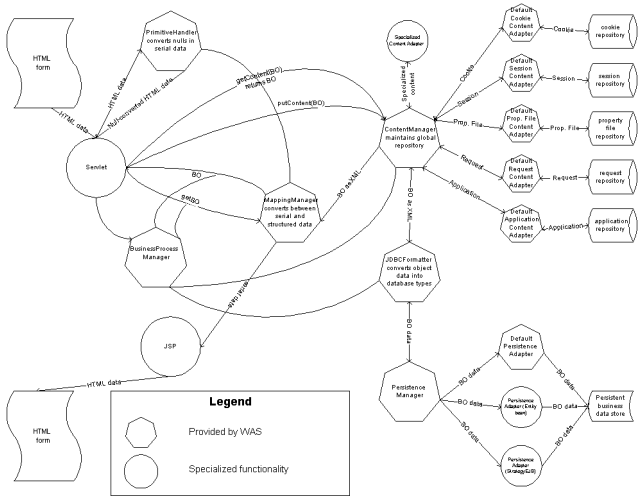
In other words, what J2EE does not do is look at business applications and ask “How do I help a developer build a business application?”

What is an application architecture?

An application architecture bridges the gap between the business-specific code, which is the code specific to your application, the code that only developers familiar with your business needs can write, and the underlying technology infrastructure, which is implemented as the J2EE-compliant application server.

A good application architecture separates the difficult technical challenges from the production of the business application. The architecture accomplishes this by defining how to create business components in a way that is independent of the underlying technologies. Developers who use an application architecture can invest significantly less time becoming familiar with the J2EE API and more time concentrating on implementing business logic. They only have to understand the business level components, an easier task than mastering J2EE for most application developers.

The Wakesoft application architecture focuses developers on the job of building robust business applications on top of J2EE by providing an integrated collection of business application frameworks and application development accelerators that can be used to form the core of your business application.



Why do I need an architecture?

All too often, applications running on top of J2EE mix technical plumbing and business logic code. A codebase in which functionalities are intermingled becomes harder and harder to debug and maintain as the application grows and matures.

The frameworks provided by a good application architecture solve many of the problems that developers face when building applications from scratch. Developers no longer have to re-invent wheels and you don't find yourself staying up at night worrying that key pieces of code were implemented inefficiently -- or worse.

Wakesoft's application architecture provides a well-documented structure that helps developers to isolate business logic, presentation logic, and infrastructure from each other. It provides a clear map of how they need to interact with each other. Working from this sort of clearly defined map makes the development process much easier to manage since each member of the of the development team can be provided with a clear blueprint of what piece of the whole they responsible. As a result, applications are developed faster, perform better, and are dramatically less costly to maintain.

What is the Wakesoft Architecture Server

The Wakesoft Architecture Server is an integrated collection of J2EE frameworks, including an implementation of Sun's J2EE Blueprints Design Patterns. Together,

these frameworks form a complete architecture for any J2EE application. Companies use the Wakesoft Architecture Server in conjunction with J2EE application servers as the foundation for rapidly building robust, scalable and extensible J2EE applications.

The Wakesoft Architecture:

- Separates technical code from business code
- Enables code reuse
- Isolates complexity
- Leverages design patterns
- Builds a maintainable and extensible application

Who should read this manual

This manual serves as an introduction to designing and architecting Wakesoft Architecture Server applications. It begins with an introduction to the philosophy of Wakesoft development. It then provides a thorough discussion of many of the frameworks provided to developers by Wakesoft Architecture Server.

Target audience

The reader of this manual should be a Wakesoft Architecture Server developer or architect.

Conventions followed in this manual

Convention	Used for
boldface	Bold type indicates words that are to be typed exactly as shown.
italics	Italics indicates information that the user or application provides, such as variables in commands.
`computer`	Computer typeface is used for sample command lines and program code. `Bolded code` is used to highlight parts of programs that merit your special attention.
[ ]	Brackets indicate optional items. I chose brackets over <> because <> are used more frequently in HTML.
…	An ellipsis indicates that unimportant lines of code may have been omitted.
|	A vertical bar separates two mutually exclusive choices.

The Wakesoft Architecture Server documentation set

The complete Toontown Matchmaker (dating service) sample application is included in the Wakesoft Architecture Server installation package. Code from this sample application is used throughout the Wakesoft Architecture Server documentation set to demonstrate procedures and principles.

Guide to Installing Wakesoft Architecture Server describes how to install Wakesoft Architecture Server and provides hints about installing and configuring databases and the application server to work with Wakesoft Architecture Server.

Developing Wakesoft Architecture Server Applications (Quick Start) provides experienced Wakesoft Architecture Server programmers with a concise list of the steps required to generate a Wakesoft Architecture Server application. This document is targeted at experienced Wakesoft Architecture Server developers.

The *Wakesoft Architecture Server Toontown Tutorial* uses a dating service example to teach programmers new to Wakesoft Architecture Server how to create an application and highlights many of Wakesoft Architecture Server's most useful features.

The *Wakesoft Architecture Server Reference* provides concise descriptions of the many tools Wakesoft Architecture Server provides. It is targeted at experienced Wakesoft Architecture Server developers and architects.

The Wakesoft Architecture Server JavaDocs provide reference information for each package in the Wakesoft Architecture Server API.

Using Wakesoft Architecture Server with TogetherSoft describes how to generate a Wakesoft Architecture Server application from the TogetherSoft IDE.

Frameworks Implemented by the Wakesoft Architecture Server provides an architect's introduction to Wakesoft Architecture Server.

For More Information...

More information about Wakesoft Architecture Server is available on the Wakesoft Web site: `http://www.wakesoft.com`.

Most application server and RDBMS database vendors maintain up-to-date information about their products on the Internet.

For more information about J2EE, see Sun Microsystems' Web site:

`http://java.sun.com/j2ee/`

CHAPTER

2

Using Wakesoft Architecture Server

Wakesoft Architecture Server applications are implemented according to the Inversion of Control design pattern. When a Wakesoft application runs, the Wakesoft Architecture Server takes responsibility for calling custom components as necessary. In this way, the Wakesoft Architecture differs from a standard class library, where custom code is in control and occasionally invokes isolated methods in the class library to accomplish certain specialized functions, such as making a calculation or opening a file.

Writing a Wakesoft Architecture Server application

To write a Wakesoft Architecture Server application, you:

- Design the business objects of your J2EE application.
- Create the business attributes of your objects in Java. At Wakesoft, we call these the skeleton business objects, because you don't have to put any code on the bare-bones descriptions of your business objects.
- Generate business objects using the Wakesoft Application Development Accelerators.
- Generate an application prototype using the Wakesoft Application Development Accelerators.
- Create the database using schemas generated by the Wakesoft Application Development Accelerators.
- Build the application.
- Deploy the application.
- Run your application.

- Iterate to build final application.

Philosophical underpinnings

The architecture underlying the Wakesoft Architecture Server applications provides the following services:

- Independence from underlying technologies via adapters.
- Transparent handling of XML data (serialization).
- Caching of global application data.
- Strong support for persistence.
- Support for business processes and transactions.
- User interaction structure follows the model-view-controller design pattern
- Consistent exception handling.
- Shields you from many J2EE inter-tier issues.
- Isolation of business logic from presentation logic, persistence logic and other platform-dependent considerations.

CHAPTER

3

The Wakesoft framework philosophy

According to FutureSights, Inc. Dictionary of Technical Terms, a framework is “a structure for supporting or enclosing something else, especially a skeletal support used as the basis for something being constructed.”

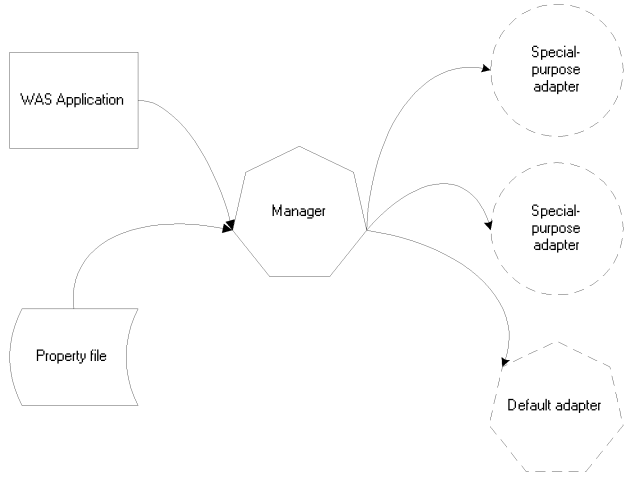
The Wakesoft Architecture Server software implements a set of integrated frameworks. Some Wakesoft frameworks define business level components so that developers can easily determine what needs to be built. Other frameworks define architecture level components that allow developers to change application behavior or code that uses the framework public interfaces without impacting business level components. Core frameworks encapsulate complexity and reduce coupling so that users only have to call public interfaces. In addition, the frameworks encapsulate technical decisions concerning how to use the J2EE infrastructure technology.

The frameworks provide developers with a rigorously structured environment that allows decisions to be made quickly. Better yet, the frameworks ensure that decisions you make today can be revised tomorrow with minimal application impact.

Adapters: independence from the underlying platform

For example, where ever possible, Wakesoft Architecture Server makes use of the technique of interposing a customizable adapter to do work that needs to be performed. We call this the Wakesoft Manager/Adapter Framework Approach.

Where ever the Wakesoft Architecture needs to call into the infrastructure, a *Manager* looks up the correct *Adapter* to perform the work requested.



For example, to access the datastore, rather than call an EJB directly from the business logic, a Wakesoft application calls the PersistenceManager which calls a PersistenceAdapter which can be specified by the developer at runtime. The PersistenceAdapter for one type of business object may call an EJB to write that object to a database, while the PersistenceAdapter for a different type of business object may write to a flat file or to a pipe to a backend legacy system. So, in a typical Wakesoft Architecture Server application, the default PersistenceManager finds the default PersistenceAdapter, which calls a `StrategyEJB` to save the business object. However, because of the architecture of a Wakesoft Architecture Server application, it is quite straightforward to change the behavior so that the PersistenceManager calls an entity bean rather than a StrategyEJB.

All of the components of the Wakesoft architecture -- business objects, adapters, and J2EE infrastructure -- are loosely coupled. To change the framework's behavior without impacting business logic code, all you have to do is to switch the Adapter. Because the components are loosely coupled, developers can create business components without having a thorough understanding of the underlying technology and without worrying about having to change business logic when the underlying technology changes.

When you use the Wakesoft Manager/Adapter Framework Approach, when an underlying platform or data structure changes, business logic can remain unaffected; all that has to be changed is the adapter.

Adapters isolate a Wakesoft Architecture Server application from:

- Change
- Underlying platform complexities

- Data formats

Adherence to the Wakesoft Manager/Adapter Framework Approach is not required of Wakesoft Architecture Server applications. However, Wakesoft recommends you try to implement your application using the Manager/Adapter Framework Approach. We think you will be glad you did the first time your underlying platform changes.

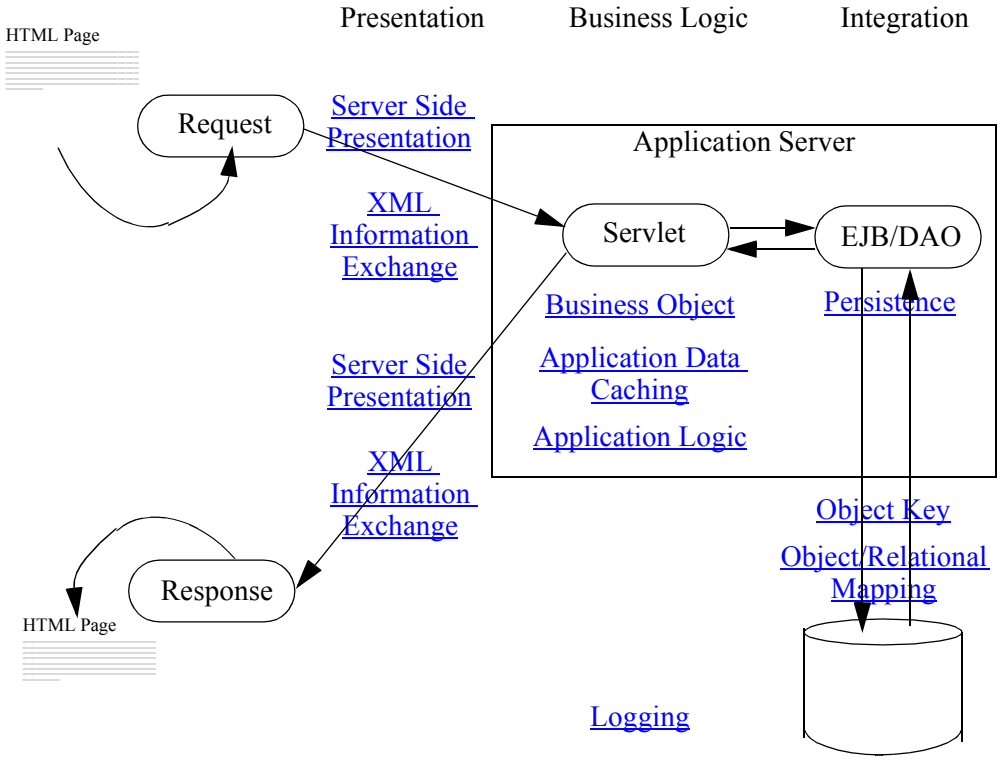
CHAPTER

4

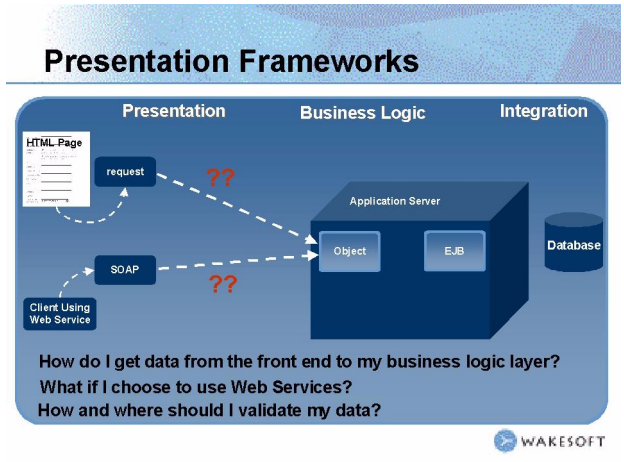
Using the Wakesoft frameworks

Because the Wakesoft Architecture Server incorporates the use of frameworks, whenever you implement functionality in your application according to the Wakesoft way, you are using tried and tested design patterns.

The following illustrates how some of the Wakesoft frameworks are used in a Wakesoft Architecture Server application:



The Wakesoft presentation layer frameworks



When you use the Wakesoft presentation frameworks:

- A simple call instantiates request data as business object data.
- Business object data is isolated from persistence. You won't necessarily have to change the code that retrieves and saves data when you change the data that is presented to end users.
- You can easily change the type of request you are passing into the servlet. (E.g., by converting your request into an XML document.)
- Wakesoft Architecture Server centralizes data validation across the entire application.

How code changes when Wakesoft frameworks are used

Typically, code that interacts with end users ends up including lots of other kinds of functionality as well. For example,

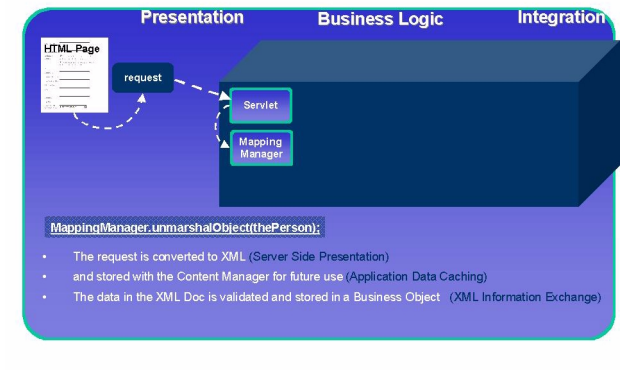
Without Wakesoft

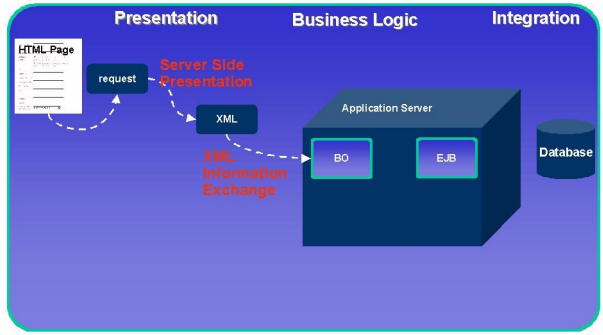
```
public void service(HttpServletRequest req, HttpServletResponse res)
     throws IOException {
   PersonBO person = new PersonBO();
   String starWarsChar = req.getParameter("starWarsChar");
   String birthDateString = req.getParameter("birthDate");
   DateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
   Date birthDate = null;
```

```
   try {
      birthDate = formatter.parse(birthDateString);
   }
   catch (ParseException pe) {
   // deal with exception
   }

   // You need code here to search database to find out if the name entered
   // is a Star Wars character
   // redisplay JSP
   // how will I get my error message into the JSP??
   }

   person.setStarWarsChar(starWarsChar);
   Person.setBirthDate(birthDate);
}
```

With Wakesoft

```
public void doWork() throws ApplicationException {
   PersonBO thePerson = new PersonBO();
   MappingManager.unmarshalObject(thePerson);
   displayPage("Success.jsp");
}
```





The Wakesoft presentation layer frameworks are:

- Server Side Presentation
- XML Information Exchange

The Wakesoft business logic layer frameworks

The Wakesoft business logic frameworks are:

- Business Object
- Application Data Caching
- Application Logic
- Logging

The Wakesoft integration layer frameworks

The Wakesoft integration frameworks are:

- Persistence
- Object/Relational Mapping
- Object Key

CHAPTER

5

Server Side Presentation

Problem

J2EE JavaServer Pages (JSPs) provide a programmatic way to format data for display on HTML pages. Mixing business logic into the presentation logic that is supposed to comprise JSPs leads to unfortunate results, including:

- User interfaces that are hard to extend.
- Skills mismatches because the user interface specialists don't know enough Java to maintain the JSPs.
- Code redundancy -- the same functions are included in both the business logic and presentation layers of the application.

Solution

The Wakesoft Server Side Presentation Framework provides:

- High level page abstraction that enables developers to build pages using reusable components.
- Reusable view components that can be used within the page.
- An XML-based approach for handling data input and output.

High level page abstraction

In the Wakesoft Server Side Presentation Framework, high-level page abstractions are called *views*. *Composite views* combine individual views. The framework

provides a PageManager/ PageAdapter interface for constructing these reusable views. The views themselves are implemented as JSPs.

Reusable view components

To assist developers in constructing views (aka JSPs), the framework provides a set of reusable view components. For example, the Wakesoft Application Development Accelerators streamline the process of creating of lookup values for lists of data. Developers can choose to display these lookup values as HTML drop downs, radio buttons, etc. and can easily change their representation (for example, change a drop down into a radio button).

XML-based data handling

Like many of the Wakesoft Architecture Server frameworks, the Server Side Presentation Framework handles HTML-based user presentation data formatted as XML. Many benefits result when developers standardize all data handling using the XML Information Exchange Framework, so that HTML form data handling is consolidated with XML data handling. Centralizing data input and validation, object-to-field mapping, error-handling and data formatting for display streamlines code and results in consistency throughout the application.

The Server Side Presentation Framework utilizes the MappingManager (part of the XML Information Exchange Framework) and the ContentManager (part of the Application Data Caching Framework).

The Server Side Presentation Framework converts data from incoming HTTP requests into XML using a configurable ContentAdapter in the Application Data Caching Framework. (This conversion takes place only if requested and is done only once and then cached.) The request data, previously just name/value pairs, is now in a hierarchical XML structure that the XML Information Exchange Framework can handle as if it were any other XML data. Invalid data and other exceptions pertaining to form data can be redisplayed easily using the JSP view components. Erroneous fields can be highlighted, localized error messages can be automatically displayed, and invalid data can be automatically redisplayed.

J2EE Design Patterns Implemented

- Business Delegate
- Composite View
- Dispatcher View

- Intercepting Filter
- Service to Worker
- View Helper

Benefits

Using the Wakesoft Server Side Presentation Framework results in cleaner user interface code that is easier to maintain and extend. When developers use the Wakesoft Server Side Presentation Framework:

- The user interface is decoupled from the business logic so that it can be modified without impacting the other parts of the application.
- User interfaces can now be developed by user interface specialists rather than experienced Java programmers.

CHAPTER

6

XML Information Exchange

Problem

XML is the de facto standard for transporting serialized data. However, within an application, developers find that working with objects is easier than working with XML. This means that within applications, developers must often convert data from XML into objects and back.

Conversion between these two formats is complicated because while XML data has a hierarchical structure, Java data has an object structure. Standard low-level packages for interacting with XML such as DOM and SAX do not resolve the XML-Java structure mismatch. Therefore, using the low-level APIs to convert directly between XML and objects can result in non-reusable validation and data conversion logic.

Solution

The Wakesoft XML Information Exchange Framework shields developers from the intricacies of XML-Java mapping so that they can focus on creating reusable business components.

The Wakesoft Architecture Server XML Information Exchange Framework encapsulates the functionality of XML-to-object data binding and object-to-XML formatting. The Framework consists of a MappingManager, which controls the process. The MappingManager calls on MappingAdapters, which encapsulate the logic that traverses the XML and Java structures to identify business-object-level matches. The MappingAdapters in turn call Mappers, which validate data, report errors, if they are found, in XML formatted for display, and load valid data into the XML document or object.

The Framework can map single objects, collections of objects, object graphs (groups of related objects), and matching XML and objects using identifiers.

The MappingManager can serve as a façade for the XML Information Exchange Framework. Because the Manager delegates calls to a MappingAdapter, developers can easily configure the behavior of the framework. For example, the developer can easily change the algorithm used to compare business objects without impacting the code which calls the MappingManager. The MappingAdapter is responsible for navigating the XML or the objects, depending on whether it is marshaling into XML or unmarshaling into a data object . As matches are found between the XML and objects, the MappingAdapter calls Mappers to perform the actual translation, including validation .

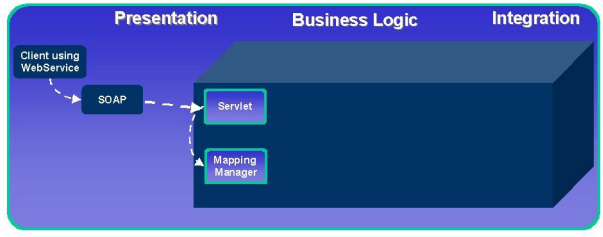
The Mapper called can be either a custom component or a prepackaged component. A custom Mapper would contain application-specific functionality, such as determining required fields when unmarshaling the XML, or special formatting that needs to occur when creating the XML from objects, such as combining the person's first name, middle name, and last name into a single name XML node.

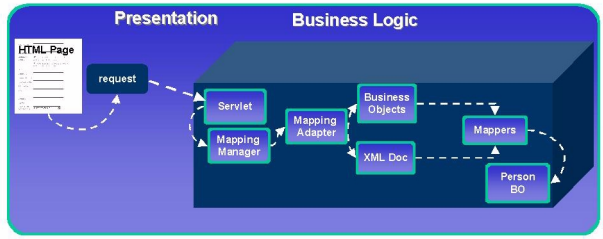
The prepackaged Mapper performs the work without requiring custom code by using introspection. This is possible by following conventions to allow for the matching to happen between the XML and object.

The prepackaged Mappers also provide field-level data validation for datatypes, such as validating that a field that should contain an integer in an XML node is actually an integer before trying to populate a Java int field with the value.

Both the custom and prepackaged Mappers conform to a standardized error handling convention for flagging errors in the XML. This allows for location-specific error indication, discovery and correction.

Figure 6.1 You can invoke the XML Information Exchange Framework directly (bypassing the Server Side Presentation Framework) by passing it an XML source such as an XML document.





J2EE Design Patterns Implemented

None apply.

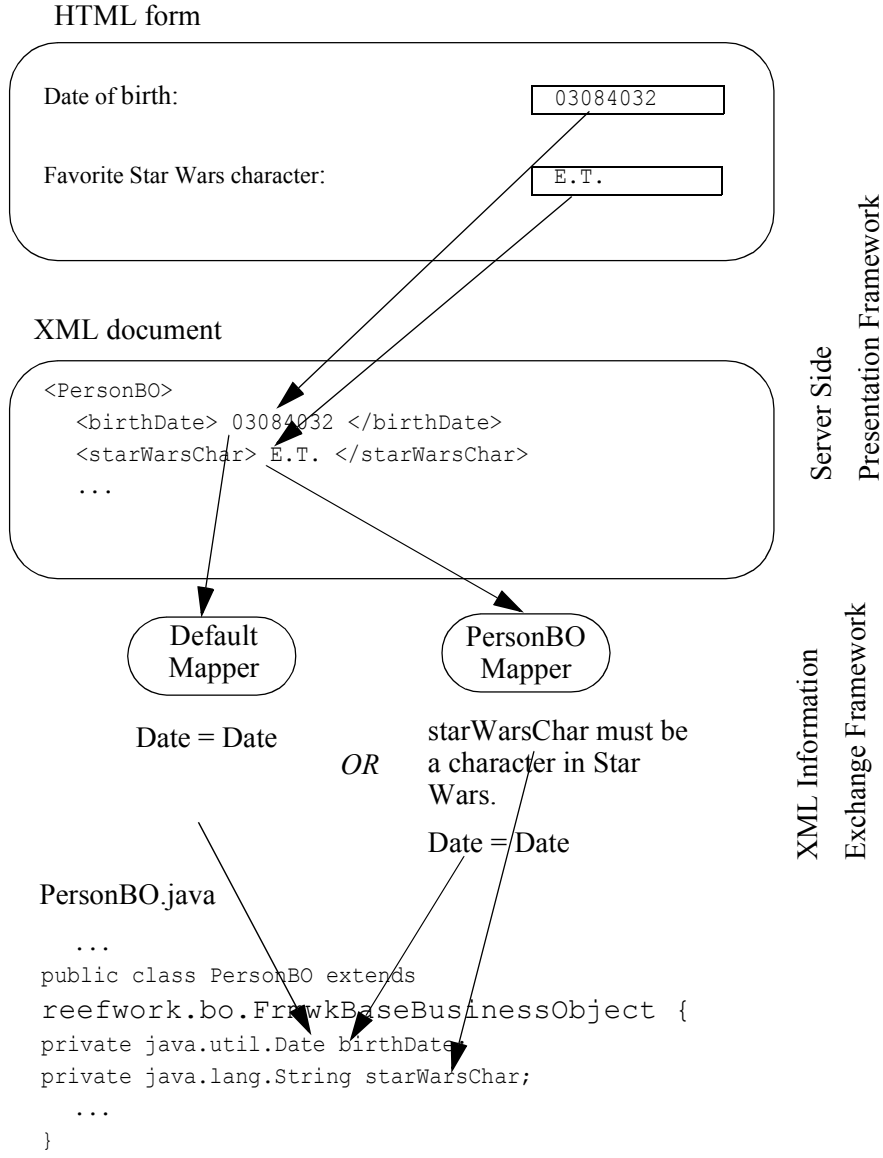
Benefits

- Encapsulation of XML-Java data binding.
- Centralized data validation, data mapping from XML to objects, data formatting from object to XML.
- Creation of business components that are reusable across applications.

Example

In the following example, an HTML data entry form is submitted to the XML Information Exchange Framework. The PersonBO Mapper is a custom mapper that validates application-specific data. If a custom mapper is specified for a business

object, the default mapper is not called. However, the custom mapper can extend an existing mapper to provide additional capability.



Before validation

```
<PersonBO>
   <starWarsChar> E.T. </starWarsChar>
   ...
```

After validation

```
<PersonBO>
   <starWarsChar error = E.T. is not a Star Wars character> E.T.
   </starWarsChar>
   ...
```

CHAPTER

7

Business Object

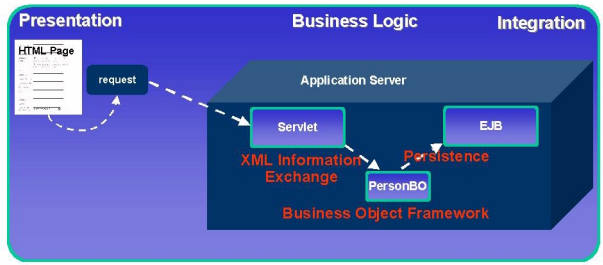
Problem

A business object is an instance of a class that represents a business entity such as a customer, address, or person.

If business data is not structured in a meaningful way, data fields can be interacted with directly (as would be the case in Perl or C). When business data is accessible at the field level, different parts of the application may handle it differently than other parts do.

When the structure of business data is dependent on the underlying technologies (such as directly referencing EntityEJBs from within the business logic), the structure of the business data will have to change when technical decisions (e.g. how do I persist my data) change. Therefore, the structure of business data should be independent of the underlying persistence and distribution mechanisms.

Solution



The Wakesoft Business Object Framework makes it possible for you to define single objects (Business Object) and collections of objects (Business Object Collection), to maintain business objects and collections of business objects in object hierarchies and to establish and maintain relationships between them.

The Framework provides a Business Object Introspector that can be used to retrieve detailed information about Business Objects. The Business Object itself also serves as a façade for some of the other Frameworks, such as Persistence and Application Logic.

The Business Object Collections support Java Collections behavior. They can be ordered using the BusinessObjectComparator and iterated over using the standard Iterator.

J2EE Design Patterns Implemented

Business Delegate

Value Object

Composite Entity

Value List Handler

Benefits

- Structures application data so that it can be reused across applications.
- Isolates creation of data structures and relationships so that they are not dependent on underlying technologies that may be used for persistence or distribution.
- Isolates code in servlets from the underlying persistence mechanism.

- Isolates data structures from the underlying persistence mechanism.

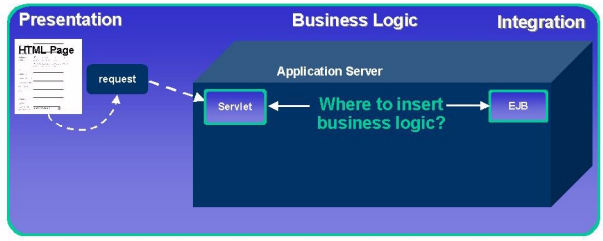
CHAPTER

8

Application Logic

Problem

J2EE provides just two places to execute application logic: within servlets (including JSPs) or within EJBs.

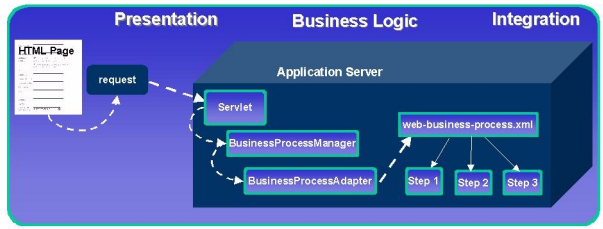


Unfortunately,

- Application logic stored in a servlet cannot easily be executed from a non-servlet.
- Unless carefully designed, servlets tend to suffer from code bloat and spaghetti code.
- Code reuse is only through superclasses which limits it to J2EE component types.
- Business components are often implemented as EJBs, even though EJBs introduce considerable overhead.

Solution

The Wakesoft Application Logic Framework structures application logic into reusable components that can then be assembled to respond to an event.



The Framework consists of the BusinessProcessManager, BusinessProcessAdapters, BusinessProcessEvents, and BusinessProcessSteps.

When a call is made from within the servlet to handle a business process (e.g., `handleBusinessProcess("PersonProcess");`) the appropriate BusinessProcessAdapter loads up the XML document (`Web-business-process.xml` or `ejb-business-process.xml`) that lists the steps to be executed to perform the business process. These reusable `BusinessProcessSteps` are executed in the order in which they are defined in the document.

The BusinessProcessManager is a façade for the application logic behavior with a simplified interface that takes a BusinessProcessEvent and executes the correct logic for that event. The Manager delegates process calls to the BusinessProcessAdapter which then responds to the event. The BusinessProcessAdapter uses an XML file containing the process configuration indicating the BusinessProcessSteps that should be executed to handle the BusinessProcessEvent. This structure is based on the Chain of Responsibility design pattern from the Gang of Four Design Patterns book. The BusinessProcessSteps are reusable across processes.

An application can use this Framework on both the Web Tier and EJB Tier. Because the same application code runs on both tiers, functionality can be executed on the Web Tier in cases where enhanced performance is required and migrated to the EJB Tier when EJB Tier behaviors, such as distribution or container-managed transactions, are needed.

J2EE Design Patterns Implemented

Business Delegate

Intercepting Filter

Service Locator

Service to Worker

Session Façade

Benefits

- Business logic is encapsulated in low-barrier components that are easily understood.
- With the Wakesoft Approach, your code is decoupled from both the Web Tier and the EJB Tier allowing you to reuse it across both tiers: Business logic can be executed anywhere within your J2EE application for optimal performance without requiring overhead of heavy weight J2EE components.
- Code is reusable at the class level; you need not cut and paste it.
- Application logic is rapidly assembled and modified from reusable components, allowing for changing requirements and technology.

Example

PersonSV.java

```
  handleBusinessProcess("PersonProcess");
```

Web-business-process.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<business-process-definitions>
   <business-process name="AddressProcess">
      <bp-step class="reefwork.bp.step.LogStep" />
      <bp-step class="com.wakesoft.sample.applogic.webstep.LoginStep" />
      <bp-step class="com.wakesoft.sample.applogic.webstep.AddressStep" />
   </business-process>

   <business-process name="PersonProcess">
      <bp-step class="com.wakesoft.sample.applogic.webstep.PersonStep" />
   </business-process>

</business-process-definitions>

Web-business-process.xml
   <?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<business-process-definitions>
```

To change application logic, simply add new steps. Recompilation is not necessary.

```
<business-process-definitions>
  <business-process name="AddressProcess">
    <bp-step class="reefwork.bp.step.LogStep" />
    <bp-step class="com.wakesoft.sample.applogic.webstep.LoginStep" />
    <bp-step class="com.wakesoft.sample.applogic.webstep.AddressStep" />
  </business-process>
  <business-process name="PersonProcess">
    <bp-step class="com.wakesoft.sample.applogic.webstep.LoginStep" />
    <bp-step class="com.wakesoft.sample.applogic.webstep.GetLastVisitedStep"
/>
    <bp-step class="com.wakesoft.sample.applogic.webstep.PersonStep" />
  </business-process>
</business-process-definitions>
```

CHAPTER

9

Application Data Caching

Problem

Code that reflects the current data caching and storage strategies is sometimes spread throughout the application. The technical intricacies of interacting with cached and stored data are exposed throughout the application.

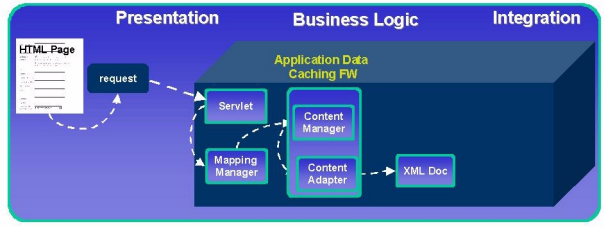
Solution

The Wakesoft Application Data Caching Framework simplifies support for storage and retrieval of application data and allows for changing the implementation without impacting the rest of the application. The classes that make up the Application Data Caching Framework are the ContentManager and ContentAdapters.

The ContentManager serves as a façade for the many possible data caching techniques provided by this framework. The ContentAdapters do the work of exchanging data with the various repositories, such as the HttpSession, Stateful Session EJBs, HttpRequest, and JMS repositories. Because a Manager/Adapter implementation is provided, developers may construct new data caching implementations.

Figure 9.1 When the MappingManager is called to unmarshal the business object (e.g., `MappingManager.unmarshalObject(thePerson);`), the Mapping Manager uses the Application Data Caching Framework to convert the request into an

XML Document. The XML Document is returned to the Mapping Manager for use by the XML Information Exchange Framework.



The ContentManager uses a composite key for caching data. Part of the key indicates the lifetime of the cached data, the other part is a unique identifier for the data. The ContentManager uses the lifetime key to determine which adapter to use. The adapter uses the unique identifier to find requested the data within its repository. The adapter can be configured at runtime without any code changes. This allows for changing the caching implementation, say from using the HttpSession to using a Stateful Entity EJB, without any impact.

J2EE Design Patterns Implemented

None apply.

Benefits

- Encapsulates complexity of dealing with multiple data caching repositories.
- Allows for deploy-time decisions of which repositories to use based on performance, availability, or other deploy-time dependencies.
- Allows for greater code portability and reuse because code no longer contains J2EE tier-dependent code. For example, without this encapsulation, a component might directly interact with the HttpSession, which would prevent that code from being moved to the EJB tier for execution. Using this Framework, the code would interact with the ContentManager, which on the Web tier may use the HttpSession but on the EJB tier might use a Stateful Session EJB. The same code could be executed on either tier; the ContentManager configuration would handle the dependencies.

CHAPTER

10

Logging

Problem

Logging is the outputting of messages from within an application. Logging can be useful for many purposes, including security, debugging, auditing, and monitoring. Inconsistent logging can adversely affect runtime behavior and complicate efforts to debug and improve applications.It is therefore very important that a standard for logging be set early in the application development lifecycle. It is also very important that the logging mechanism be independent of underlying technologies. For example, it is not a good idea to use a logging approach that is tied to a particular application server.

Solution

The Wakesoft Architecture Server LogManager provides a centralized means of logging messages from your Wakesoft Architecture Server applications. Like the ContentManager and its ContentAdapters, the LogManager delegates its work to a helper class: LogAdapter. One LogAdapter is registered with the LogManager to handle message logging.

The default LogAdapter allows you to specify:

- A single output destination and
- Multiple levels of logging messages.

Because you can set these properties in the `wakesoft_archsvr.properties` file, you do not have to change code or re-compile to change LogManager behavior. For example to redirect logging from the WebLogic console to a flat file, change the configured adapter to FileLA.

The LogManager provides a standard way for logging throughout a Wakesoft Architecture Server application.

The behavior of the LogManager can be configured at runtime to use a variety of mechanisms to perform the actual logging. This dynamic behavior is accomplished via the Wakesoft LogAdapter. There are prepackaged adapters that support generic logging behavior, just as to a log file or to the standard output, as well as application server specific adapters, such as to the WebLogic logging facility.

J2EE Design Patterns Implemented

None apply.

Benefits

- Logging is done consistently throughout the application, making it easier to maintain.
- You can change logging behavior without impacting code. This simplifies adaptation for changing requirements.

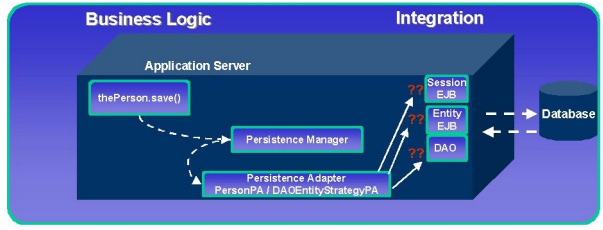
CHAPTER X

11

Persistence

Problem

Designing and implementing the persistence solution for a distributed application is a complex problem. For persistence, J2EE provides EntityEJBs, which can be implemented using Bean-Managed Persistence (BMP) or Container-Managed Persistence (CMP). Application developers have learned from experience that additional options are required. These include using SessionEJBs as facades for accessing EntityEJBs and using Data Access Objects to replace the more heavyweight EntityEJBs. The architect has to decide which standard J2EE features to use and whether to build or use third-party products for functionality that J2EE does not provide out of the box.

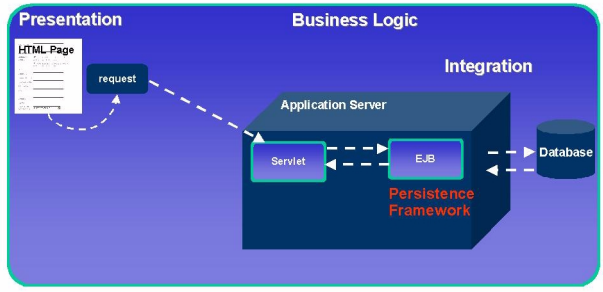


After the persistence mechanism has been chosen comes the hard work: getting it to work. Finally, as requirements change or bottlenecks surface, the persistence mechanism may need to be altered.

One of the most pressing problems with the way persistence is implemented is that, all too often, code that exposes the persistence mechanism is distributed throughout the application code. As a result:

- What should be clean business code is infused with technical code - the developer creating that code has to know too much about the underlying persistence mechanism.
- When changes to the persistence mechanism are required, code throughout the application must also change.

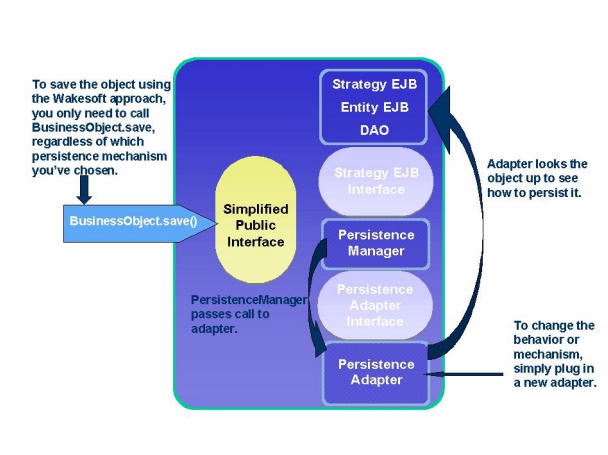
Solution



The Wakesoft Persistence Framework encapsulates the object persistence. What this means for the Wakesoft application developer is that instead of embedding calls to any particular persistence mechanism in business logic, the developer always places a call to a Wakesoft PersistenceManager method such as save, select, delete, or selectCollection. The Wakesoft Persistence Framework determines which mechanism to use to satisfy persistence requests at runtime.

The classes that the Framework uses are the PersistenceManager and PersistenceAdapters. The PersistenceManager serves as a façade for the persistence

technology. It calls various PersistenceAdapters that encapsulate the persistence mechanisms.



The PersistenceManager can be configured to use multiple PersistenceAdapters, so a single persistence mechanism need not be used throughout. This allows for different behaviors based on which object is being persisted or other criteria.

PersistenceAdapters prepackaged with Wakesoft Architecture Server support Session EJBs, Entity EJBs, and XML integration. The PersistenceAdapter configuration can be changed without impacting the code that uses the PersistenceManager. This makes it possible to change the approach to persistence due to requirements, performance, or other reasons later in development without impacting the application.

J2EE Design Patterns Implemented

Business Delegate

Composite Entity

Service Locator

Session Façade

Value List Handler

Value Object Assembler

Benefits

- The persistence mechanism is encapsulated in reusable components. This shields the team from having to understand the intricacies of the object persistence mechanism chosen.
- Make decisions today knowing that they can be changed easily tomorrow. Changing the persistence approach in response to new requirements, to improve performance or for any other reason, will not require a major application overhaul of your application.
- Persistence logic is cleanly separated from business logic.
- You can use existing Wakesoft adapters or quickly build your own.

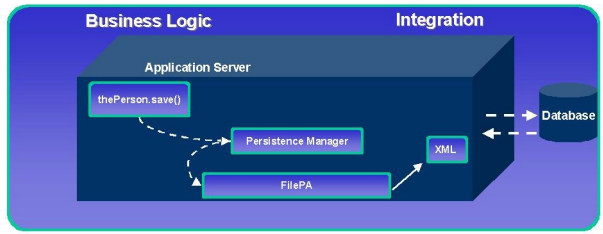
Example: Saving using Wakesoft's Persistence Framework

A call is made from anywhere within the application to save the object, for example, a PersonBO:

```
public void doWork() throws ApplicationException {
PersonBO thePerson = new PersonBO();
  MappingManager.unmarshalObject(thePerson);
  thePerson.save();
}
```

The PersistenceManager looks for the adapter registered for the PersonBO. If there is no adapter specified for this type of object, the PersistenceManager uses the default adapter specified by the developer. The default adapter provided by Wakesoft Architecture Server `(DAOEntityStrategyPA)` searches, in order, for a Strategy, Entity, or DAO with which to persist the PersonBO data.

Example: Changing the persistence mechanism



To persist your data (to an XML file, for example) simply change the adapter used.

CHAPTER

12

Object/Relational Mapping

Problem

Data which, in an application, is accessed in objects is often persisted in a relational database, such as Oracle. If the relationship between object data and relational data is not well-thought, the object/relational mapping code may end up being exposed throughout the application. The code can have an ad-hoc structure that is hard to maintain and understand. There are many decisions to be made when mapping objects to the database. These include how to handle null values and how to handle object-datatype-to-database datatype mismatches. Without a plan, these decisions may end up being made inconsistently.

Solution

The Wakesoft Object/Relational Mapping Framework gives structure to the object relational mapping code. The Wakesoft Object/Relational Mapping Framework provides easy-to-understand methods such as `doUpdate`, `doInsert`, `doDelete`, and `doSelect`. These methods are called at the correct time by the Framework; the developer need not understand the technical details.

It also provides a configurable Manager to perform Java-datatype-to-relational-database-datatype mapping. For example, using the Wakesoft Object/Relational Mapping Framework, it is easy to ensure that Java Booleans are stored consistently throughout your application's database tables as either '`T`'/'`F`' or '`true`'/'`false`'. The datatype mapping framework also handles null value situations. For example, you may set it up so that an integer that is stored as a 'null' in a database consistently correlates to a Java `int` with the value `-9999` when assigned to an object.

J2EE Design Patterns Implemented

Composite Entity

Data Access Object

Benefits

- The object/relational mapping functionality is developed in a clean and consistent manner.
- Java object/relational database datatype mismatches are handled by configurable components. This results in a consistent solution to the datatype mismatches that can be easily changed without impacting the application code.

CHAPTER

13

Object Key

Problem

Developers often find it necessary to give objects unique identifiers within applications. There can be many reasons for this, the most common being that the object represents some domain entity, such as a person or a company, that must be consistently referenced and eventually synchronized with other systems or with the persistence storage. If the developer chooses to use a persistent storage mechanism, such as an Oracle sequence generator, for creating unique keys, this decision can permeate throughout an application. Using these kinds of unique key generators often results in hard-coding the application to rely on the Oracle database. When it is discovered that there is a performance impact using the approach, or that the application must use fixed length strings because integer values can grow infinitely large, or the backend database changes, the entire application must be overhauled.

Solution

The Wakesoft Object Key Framework provides a simplified interface for retrieving unique keys for objects. The Framework consists of the Wakesoft KeyManager and KeyAdapter classes. The Manager calls the Adapters that will return the actual keys. The KeyManager serves as a façade that hides the complexity of the key-generation technology.

Wakesoft Architecture Server includes prepackaged Adapters that support use of the underlying database as source of unique keys. These Adapters can either call the database directly or utilize EJBs to access the database. Other Adapters can generate keys based on the GUID (Globally Unique ID) approach which does not depend on a central database to generate unique keys.

J2EE Design Patterns Implemented

None apply.

Benefits

The Wakesoft Object Key Framework:

- Isolates complex key-generating code from application code.
- Allows the developer to easily change the way keys are generated without impacting application code.
- Supports multiple approaches simultaneously to accommodate objects with different requirements. For example, some objects may need to be uniquely identified but will never be persisted; these can use the GUID KeyAdapter. Other objects may need keys that are valid when/if they eventually get persisted to the database; these could use the Oracle sequence generator KeyAdapter.

INDEX

SYMBOLS

A

B

C

D

E

F

G

H

I

X