AdventureBuilderSystem

Document generated by Gendoc (<http://www.eclipse.org/gendoc>)

Help and support at : <https://www.eclipse.org/forums/index.php/f/286/>

**Aucune entrée d'index n'a été trouvée.**

Contenu

**Aucune entrée de table des matières n'a été trouvée.**

# AdventureBuilderSystem

https://wiki.sei.cmu.edu/confluence/display/SAD/Adventure+Builder+Data+Model
Rationale
\* We have chosen a Service Oriented Architecture for the Adventure Builder because we desire a high degree of interoperability between internal and external entities in the system. This gives us a great deal of flexibility in terms of implementation platforms.
\* For example, the Airline Provider component may be implemented using Java, PHP, .NET, IBM CICS or any other technology that supports SOAP web services.
\* Internally, it's possible, for example, to reimplement and deploy Consumer Website using .NET. OPC doesn't have to change because the SOAP Web services communication allows such level of interoperability.
\* We have also a clear separation between the user interface and the business logic. The Consumer Website (user interface) and OPC (business logic) are two separate applications that together implement the functionality described in the System Overview.
\* We chose to implement the service registry using a simple data repository because of ease of implementation. Creating a UDDI registry was an alternative that was rejected because it would bring increased complexity with no apparent benefit, given that the repository is only queried internally (using a standard format is more relevant when the registry will be accessed by external players).

## C&C

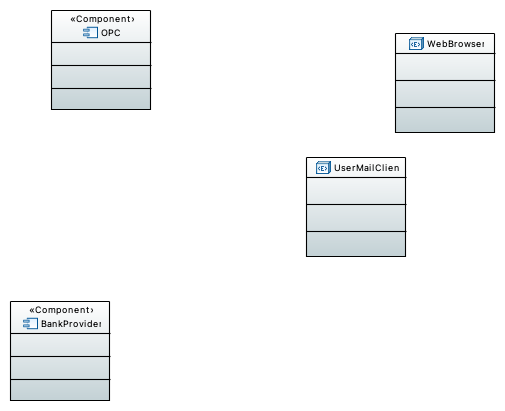


Figure 1 : C&C diagram

## UseCases

The system used as an example in this architecture document is an adapted version of the Adventure Builder Reference application, which was developed in the context of the Java BluePrints program at Sun Microsystems. This application was chosen because the functionality is easy to understand, and the source code, documentation, and other artifacts are publicly available for download. Also available is a book on Web services that explains the design and implementation of the application [Singh 2004].
The architecture documented here does not reflect exactly the implementation provided by Sun. To make it a more interesting and realistic example of an SOA solution, we made several assumptions about the business context and requirements of the system, and documented design elements and relations that deviate from the original implementation. Click for more information.

## QualityAttributes

### Modifiability

### Performance

### Reliability

### Security

### Availability

## UsageLink

## Modules

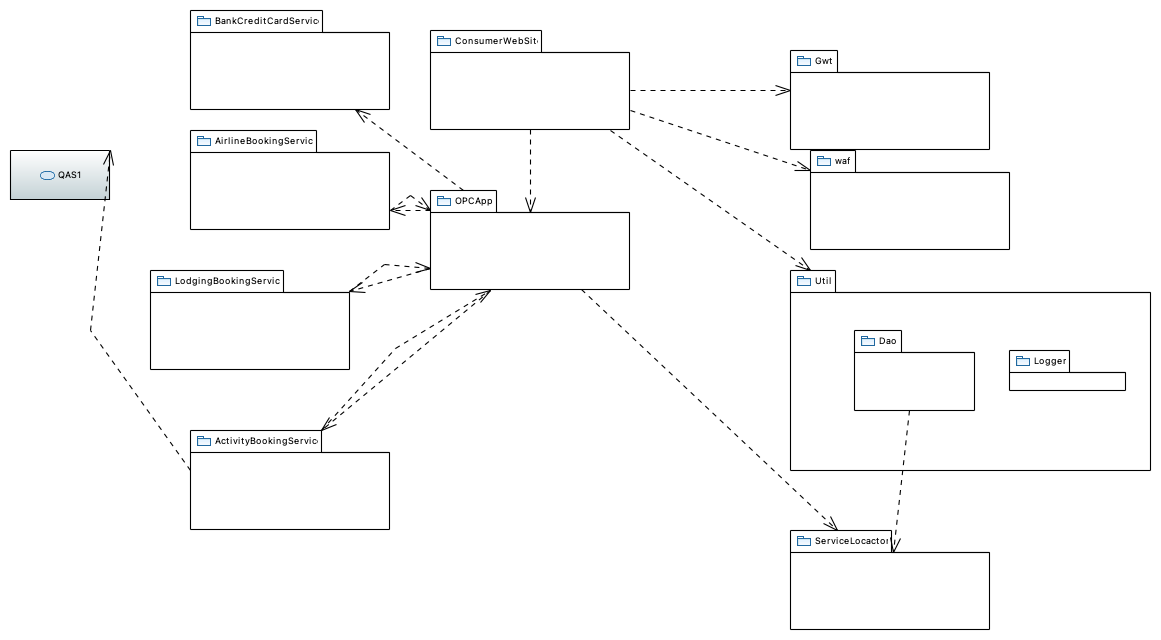


Figure 1 : TopLevelUseView

### OPCApp

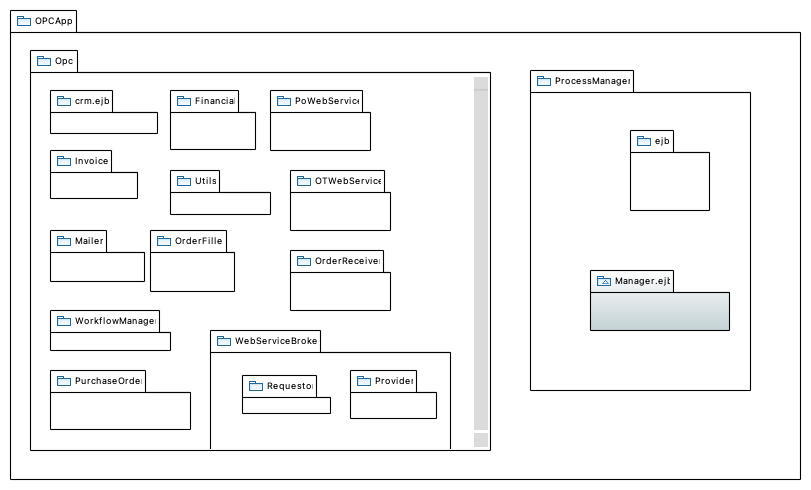


Figure 1 : OPC Module Decomposition View

OpcApp stands for Order Processing Center Application. The business logic of the Adventure Builder is implemented in this module. It provides the following functionality:
\* Accepting purchase order requests from the ConsumerWebsite for processing by hosting the Purchase Order Web Service
\* Provide a mechanism for the Consumer Website to query the current status of a purchase order by hosting the Order Tracking Web Service
\* Communicate with external suppliers to process and maintain the status of a purchase order.
\* Upon completion of processing a purchase order, send an email to the customer of its success or failure.
A module decomposition refined view and a module uses refined view of OpcApp are available.

All other modules within Opc dan use these modules

#### ServiceLocator

#### Opc

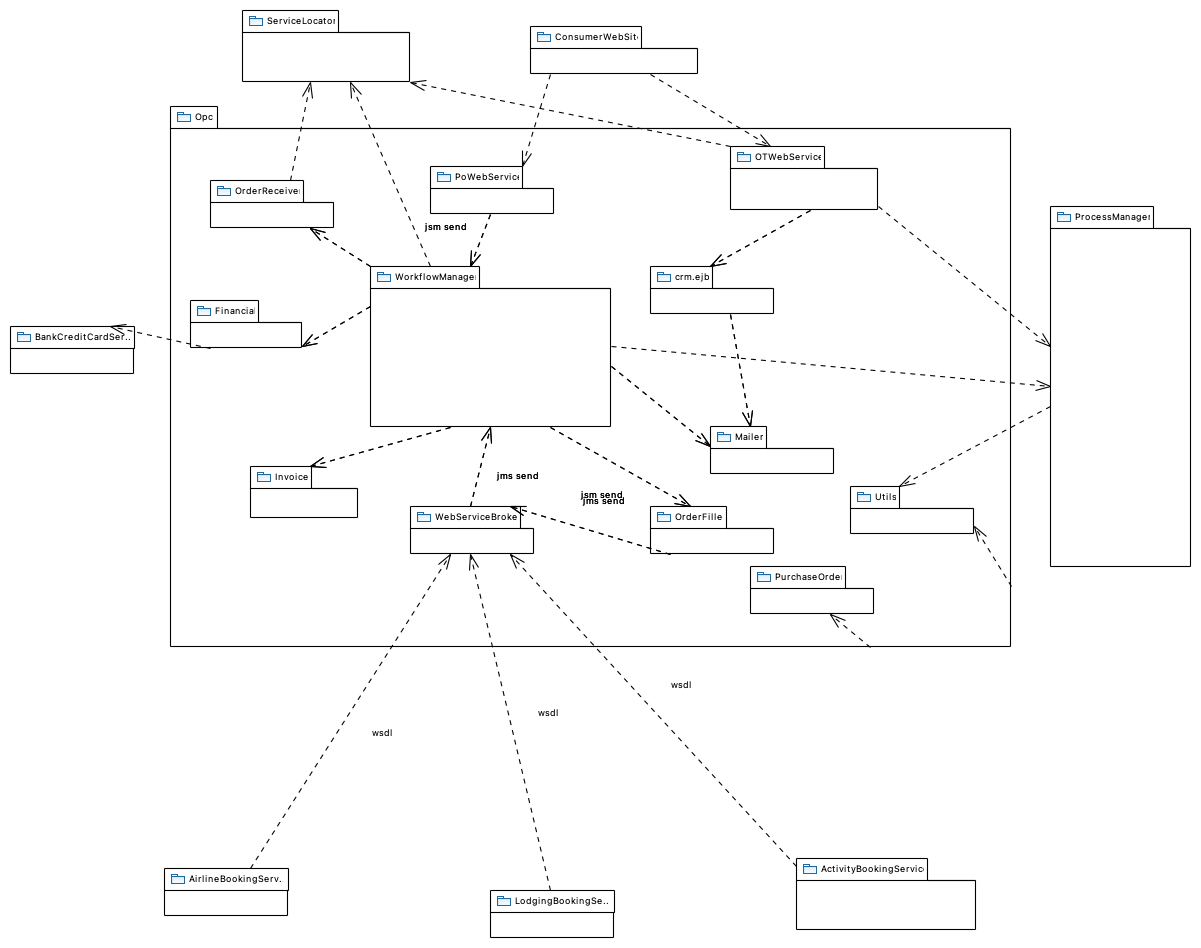


Figure 1 : OPC MODULE Uses View

This package contains all the order processing logic, including the workflow, internal queues used for communication between elements, and interaction with external web services.

##### crm.ejb

This is the Customer Relationship Manager (CRM) module. The job of this module is to send out an email once an order has been completely and successfully processed. It is implemented as a message-driven bean. In the future this module can hold additional information about customers that could assist in providing the customers with a better experience. This could include things like a history of a particular customer's purchases, or sending out periodic emails to customers regarding new and fresh deals.

##### Financial

The financial module is responsible for verifying and charging the customer's credit card. For this purpose it calls the external web services provided by the bank. The verification of the credit card happens in asynchronous manner and the OPC application waits for the external web service to reply before moving on. If the response from the banking service is not positive, the OPC application does not further process the order.

##### PoWebService

This module provides a web service that is used by ConsumerWebsite to communicate the details of a purchase order to the OPC for processing. The web service interface is OpcPurchaseOrderService:
OpcPurchaseOrderService interface documentation

##### Invoice

This package contains a data structure that holds information that the OPC uses to communicate with external suppliers. The data structure also holds the status of an order in the invoice (for more information on the lifecycle of an order see the state diagram in the OPC C&C view).

##### Utils

This package contains utility classes that are used by the OPC application, such as glue code for JMSAPI. In the future, this module will be moved to the utils package outside opc.

##### OTWebService

This module provides a web service that is used by ConsumerWebsite to query the status of an order by providing the order id. The web service interface is OpcOrderTrackingService:
OpcOrderTrackingService interface documentation

##### Mailer

The mailer is a helper module and its primary responsibility is to send out emails using the Java Mail API.It is provided with a message and email addresses to send out emails. In the future this module will be moved to the utils package outside opc.

##### PurchaseOrder

This package contains the classes that correspond in memory to the data entities required to create a purchase order. For each data entity, there is a POJO and an entity bean. The POJOs are used throughout the application as data transfer objects. The data entities in this package are:
\* Activity
\* CreditCard
\* Lodging
\* PurchaseOrder
\* Transportation
\* ContactInfo
\* Address
See the data model for a description of what each entity represents.

##### OrderFiller

This module reads an internal queue of order requests. When an order arrives, it decomposes the order into requests to the different providers involved. These requests are sent in XML format to internal queues.

##### OrderReceiver

The orderreceiver helps in persisting the purchase order in a relational database.

##### WebServiceBroker

The webservicebroker is responsible for the interaction via web services with the airline, lodging and activity providers. This module is divided into two sub-modules described below.

###### Requestor

Contains classes that can invoke the external web services provided by airline, lodging and activity supplier partners. It also contains a message-driven bean that can receive messages sent through an internal queue. These messages contain exactly the requests to be sent to the external web services.

###### Provider

This module provides a web service that is visible to airline, lodging and activity supplier partners. This web service is implemented in a session bean and is used by the external partners to submit the result of processing requests made to them. Calls to this web service are forwarded to an internal queue.

##### WorkflowManager

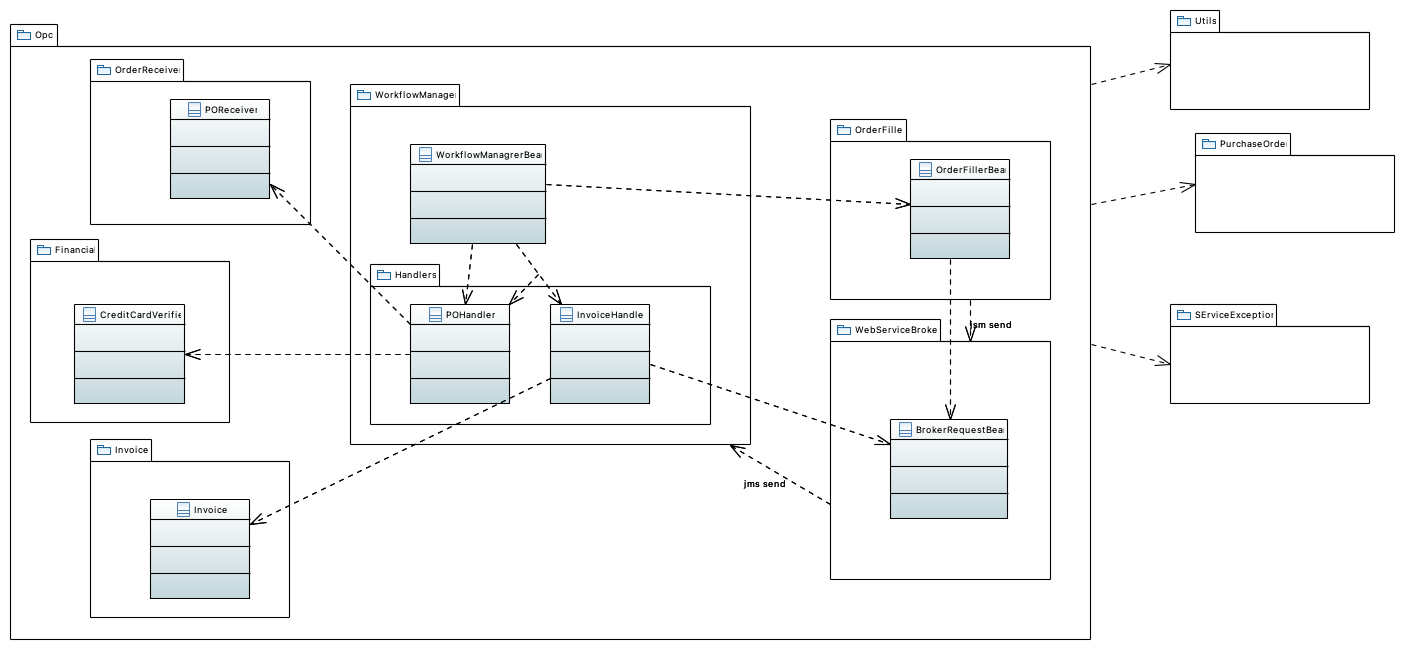


Figure 1 : Package WorkflowManager

SMTP host, message body contents, subject, sender address and reply-to address for all email messages sent by CrmBean are configurable via text files that can be changed at run time, but require reinitialization of the application to take effect. These files are implemented using standard Java i18n support.

###### Handlers

###### SErviceException

#### ProcessManager

The processmanager is used by the otwebservice module to retrieve from the database adventure package purchase orders and their updated status. It is also used by the workflowmanager to retrieve the orders placed with the external providers and persist their status. This module contains the submodules (ejb and manager.ejb) described below.

##### ejb

Contains a session bean that offers operations to retrieve orders and update the status of a given order(both the adventure purchase order and the orders placed with the partner suppliers).

##### Manager.ejb

Contains an entity bean to persist a purchase order. The entity bean uses container-managed persistence (CMP).
Rational
The choice of EJBs in the implementation, including session beans, message-driven beans and entity beans is based on:
\* Developers are familiar with EJB development and component-based development.
\* These highly modular EJB components promote reuse.

### BankCreditCardService

This module represents an external service provided by a partner bank to validate credit card transactions.

### AirlineBookingService

This module represents an external service provided by an airline partner company to book air travel.

### LodgingBookingService

This module represents an external service provided by a lodging partner company to book hotel rooms.

### ActivityBookingService

This module represents an external service provided by an activity supplier company to book different activities.

### ConsumerWebSite

The web-based user interface of the Adventure Builder is implemented in this module. The user interface lets the user browse the catalog of travel packages, place a new purchase order, and track the status of existing orders. This module creates the purchase orders based on user input and passes them to OpcApp for processing. It uses an implementation of the Model View Controller pattern called the Web Application Framework (waf). The model is implemented using Entity beans, the controller is implemented using servlets, and the view is a collection of JSPs and static HTML pages. Part of the client-facing code is implemented using the GWT framework.

### Gwt

Google Web Toolkit (gwt) is an open source framework for development of rich internet applications based on Ajax.
Rational
\* The GWT framework was chosen for the following reasons:
\* It is open source, which allows us to go under the hood and fix things when needed.
\* It provides a rich development environment with powerful trace/debug, IDE integration and build management.
\* It integrates well with any other frontend technologies since it does not use proprietary standards.
\* It offers powerful widget construction features.
\* There are several extension libraries of widget components available.
\* Code is written in Java (and translated to JavaScript during build), which is an OO language that is familiar to the majority of the team.
\* Code is compiled into Javascript, which is available on a very high percentage of browsers.
\* It has a reasonably large support community and is backed by a major player in the industry (Google).
The table below shows the comparative analysis of GWT and competing technologies based on the specific needs of our project.

### waf

Waf stands for Web Application Framework. It is a Model View Controller framework similar to Struts. It allows you to specify in configuration files the web screens and action that are associated to user clicks on specific elements of the screen. The configuration file has the mapping of screens and actions to Java classes. The framework provides the engine to display the proper screens and invoke the proper actions.
Rationale
The WAF framework was chosen because it facilitates the implementation of the Consumer Website code by providing template classes for using the MVC pattern. For a given user operation, the developer implements an action class (controller) and JSP pages that correspond to the user screens (view). The developer also uses configuration files to provide a configurable mapping between actions, action classes, events and screens. The WAF infrastructure can then automatically take http requests and invoke the action classes and JSP screens. WAF also provides support for event-based communication and internationalization.
WAF also provides support for event-based communication and internationalization.
An alternative to WAF was to use the Spring framework. Spring was a more robust and rich solution from a technical standpoint, but it was rejected because the development team is not familiar with Spring and very familiar with WAF.

### Util

This module contains utilities used by the Adventure Builder system.

#### Dao

This module contains Data Access Object (DAO) utilities, such as a DAO factory. It does not contain the actual DAO classes that access the database--these classes are inside the ConsumerWebsite module.

#### Logger

This module contains tracing and debugging utilities. The original name was tracer.

### ServiceLocactor

This module is an implementation of the ServiceLocator design pattern.

### DataModel

## Deploiement

All the components that are SOAP web service providers in the Adventure Builder System are implemented using stateless session beans. This type of EJB component can be replicated across machines in a cluster and each machine can also make available a pool of such EJBs. This feature allows scaling of the number of requests it can handle. Horizontal scalability is achievable by adding more machines to the cluster. Vertical scalability is also an option by increasing the capacity of any machine (and then increasing the number of EJB instances in each EJB pool). See EJB configuration discussion in OPC C&C View#Variability Guide.

Nevertheless, the transactions performed by these stateless session beans ultimately require access to the database. The database cannot be easily replicated and may become a performance bottleneck if the number of calls increases beyond expectation.

## Comportement

## Package11

## Package12

### Package1

#### Package1

## Experimentation

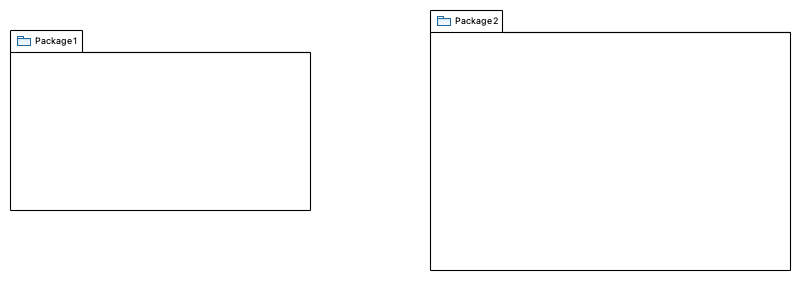


Figure 1 : NewPackageDiagram

### Package1

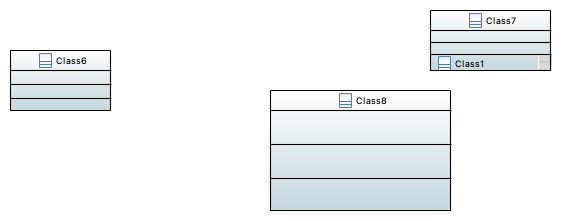


Figure 1 : NewClassDiagram

Allo mon coco #333333

Ceci est la documentatin du package #1 qui est transformé en commentaire XXXXXX

### Package2



Figure 1 : NewClassDiagram

Ceci est la documentation du package no 2
aks;dlkf;asf
asdfasd
f
as
f
sdaf
asd
f
sad

### Package3

Ceci est la documentation du package 33

## Requirements