AdventureBuilderSystem

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

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

Contenu

[1. A-Requirements 11](#_Toc71630566)

[1.1 1-!!Use cases view 11](#_Toc71630567)

[1.1.1 Acitivy Provider 11](#_Toc71630568)

[1.1.2 Adventure Builder System 11](#_Toc71630569)

[1.1.3 Airline 11](#_Toc71630570)

[1.1.4 Bank 11](#_Toc71630571)

[1.1.5 Lodging Provider 11](#_Toc71630572)

[1.1.6 UC01 Browse Cataloog 11](#_Toc71630573)

[1.1.7 UC2 Order travel package 11](#_Toc71630574)

[1.1.8 UC3 Track order 11](#_Toc71630575)

[1.1.9 UC4 Update Catalogue 11](#_Toc71630576)

[1.1.10 Vacationer 11](#_Toc71630577)

[2. B-ACOO 12](#_Toc71630578)

[3. C-Modules 12](#_Toc71630579)

[3.1 !!!Top Level Modules Uses View 12](#_Toc71630580)

[3.1.1 ActivityBookingService 12](#_Toc71630581)

[3.1.2 AirlineBookingService 12](#_Toc71630582)

[3.1.3 BankCreditCardService 12](#_Toc71630583)

[3.1.4 ConsumerWebSite 12](#_Toc71630584)

[3.1.5 Dao 12](#_Toc71630585)

[3.1.6 Gwt 12](#_Toc71630586)

[3.1.7 LodgingBookingService 13](#_Toc71630587)

[3.1.8 Logger 13](#_Toc71630588)

[3.1.9 OPCApp 13](#_Toc71630589)

[3.1.10 QAS1 13](#_Toc71630590)

[3.1.11 ServiceLocactor 13](#_Toc71630591)

[3.1.12 Util 13](#_Toc71630592)

[3.1.13 waf 13](#_Toc71630593)

[3.2 !ActivityBookingService 14](#_Toc71630594)

[3.2.1 ActivityBookingAdapter 14](#_Toc71630595)

[3.2.2 IActivityPOService 14](#_Toc71630596)

[3.2.3 QAS1 14](#_Toc71630597)

[3.3 !AdventureCatalogDBDetails class Diagram 15](#_Toc71630598)

[3.3.1 Activity 17](#_Toc71630599)

[3.3.2 Address 17](#_Toc71630600)

[3.3.3 Category 17](#_Toc71630601)

[3.3.4 ContactInfo 17](#_Toc71630602)

[3.3.5 CreditCard 17](#_Toc71630603)

[3.3.6 Document 17](#_Toc71630604)

[3.3.7 Lodging 17](#_Toc71630605)

[3.3.8 OrderDetail 17](#_Toc71630606)

[3.3.9 OrderStatusHistory 17](#_Toc71630607)

[3.3.10 Package 17](#_Toc71630608)

[3.3.11 PurchaseOrder 17](#_Toc71630609)

[3.3.12 Transportation 17](#_Toc71630610)

[3.3.13 UserAccount 17](#_Toc71630611)

[3.4 !AirlineBookingService package detail 17](#_Toc71630612)

[3.4.1 AirlineBookingAdapter 17](#_Toc71630613)

[3.4.2 IAirlinePOService 17](#_Toc71630614)

[3.4.3 QAS1 17](#_Toc71630615)

[3.5 !Bank Credit Card Service class Diagram 18](#_Toc71630616)

[3.5.1 CreditCardService 18](#_Toc71630617)

[3.5.2 ICreditCardService 18](#_Toc71630618)

[3.5.3 PaypayAdapter 18](#_Toc71630619)

[3.5.4 QAS1 18](#_Toc71630620)

[3.5.5 StrypeAdapter 18](#_Toc71630621)

[3.6 !ConsumerWebSite class diagram 18](#_Toc71630622)

[3.6.1 AdventureCatalogueDB 18](#_Toc71630623)

[3.6.2 Controller 18](#_Toc71630624)

[3.6.3 IConsumerWebSite 18](#_Toc71630625)

[3.6.4 Router 18](#_Toc71630626)

[3.6.5 Views 18](#_Toc71630627)

[3.6.6 WebBrowser 18](#_Toc71630628)

[3.7 !Model class diagram 20](#_Toc71630629)

[3.8 !LodgingBookingService package detail 20](#_Toc71630630)

[3.8.1 ILodgingPOService 21](#_Toc71630631)

[3.8.2 LodgingBookingServiceAdapter 21](#_Toc71630632)

[3.8.3 QAS1 21](#_Toc71630633)

[3.9 !OPCApp Module Decomposition View 21](#_Toc71630634)

[3.9.1 Financial 21](#_Toc71630635)

[3.9.2 Invoice 21](#_Toc71630636)

[3.9.3 Mailer 21](#_Toc71630637)

[3.9.4 Manager.ejb1 21](#_Toc71630638)

[3.9.5 OTWebService 21](#_Toc71630639)

[3.9.6 Opc 22](#_Toc71630640)

[3.9.7 OrderFiller 22](#_Toc71630641)

[3.9.8 OrderReceiver 22](#_Toc71630642)

[3.9.9 PoWebService 22](#_Toc71630643)

[3.9.10 ProcessManager 22](#_Toc71630644)

[3.9.11 Provider 22](#_Toc71630645)

[3.9.12 PurchaseOrder 22](#_Toc71630646)

[3.9.13 Requestor 22](#_Toc71630647)

[3.9.14 Utils 22](#_Toc71630648)

[3.9.15 WebServiceBroker 22](#_Toc71630649)

[3.9.16 WorkflowManager 22](#_Toc71630650)

[3.9.17 crm.ejb 22](#_Toc71630651)

[3.9.18 ejb 23](#_Toc71630652)

[3.10 Exception class diagram 23](#_Toc71630653)

[3.10.1 InvalidPOException 23](#_Toc71630654)

[3.10.2 OrderNotFoundException 23](#_Toc71630655)

[3.10.3 ProcessingException 23](#_Toc71630656)

[3.10.4 RemoteException 23](#_Toc71630657)

[3.11 !OPC MODULE Uses View 24](#_Toc71630658)

[3.11.1 ActivityBookingService 24](#_Toc71630659)

[3.11.2 AirlineBookingService 24](#_Toc71630660)

[3.11.3 BankCreditCardService 24](#_Toc71630661)

[3.11.4 ConsumerWebSite 24](#_Toc71630662)

[3.11.5 Financial 24](#_Toc71630663)

[3.11.6 Invoice 25](#_Toc71630664)

[3.11.7 LodgingBookingService 25](#_Toc71630665)

[3.11.8 Mailer 25](#_Toc71630666)

[3.11.9 OTWebService 25](#_Toc71630667)

[3.11.10 OrderFiller 25](#_Toc71630668)

[3.11.11 OrderReceiver 25](#_Toc71630669)

[3.11.12 PoWebService 25](#_Toc71630670)

[3.11.13 ProcessManager 25](#_Toc71630671)

[3.11.14 PurchaseOrder 25](#_Toc71630672)

[3.11.15 ServiceLocator 25](#_Toc71630673)

[3.11.16 Utils 25](#_Toc71630674)

[3.11.17 WebServiceBroker 25](#_Toc71630675)

[3.11.18 WorkflowManager 25](#_Toc71630676)

[3.11.19 crm.ejb 26](#_Toc71630677)

[3.12 !OtWebService package diagram 26](#_Toc71630678)

[3.12.1 IOpcOrderTrackingService 26](#_Toc71630679)

[3.12.2 OrderNotFoundException 26](#_Toc71630680)

[3.12.3 OtWebService 26](#_Toc71630681)

[3.12.4 ProcessManager 26](#_Toc71630682)

[3.12.5 RemoteException 26](#_Toc71630683)

[3.12.6 ServiceLocator 26](#_Toc71630684)

[3.12.7 crm.ejb 26](#_Toc71630685)

[3.12.8 getOrderDetails 27](#_Toc71630686)

[3.13 !POWebService class diagram 27](#_Toc71630687)

[3.13.1 IMessageListener 27](#_Toc71630688)

[3.13.2 IOpcPurchaseOrderService 27](#_Toc71630689)

[3.13.3 InvalidPOException 28](#_Toc71630690)

[3.13.4 POEndpointBean 28](#_Toc71630691)

[3.13.5 ProcessingException 28](#_Toc71630692)

[3.13.6 PurchaseOrder 28](#_Toc71630693)

[3.13.7 WorkflowManager 28](#_Toc71630694)

[3.13.8 cancelPurchaseOrder 28](#_Toc71630695)

[3.13.9 submitPurchaseOrder 29](#_Toc71630696)

[3.14 !!Package WorkflowManager 29](#_Toc71630697)

[3.14.1 BrokerRequestBean 29](#_Toc71630698)

[3.14.2 CreditCardVerifier 29](#_Toc71630699)

[3.14.3 Financial 29](#_Toc71630700)

[3.14.4 Handlers 30](#_Toc71630701)

[3.14.5 Invoice 30](#_Toc71630702)

[3.14.6 Invoice 30](#_Toc71630703)

[3.14.7 InvoiceHandler 30](#_Toc71630704)

[3.14.8 Opc 30](#_Toc71630705)

[3.14.9 OrderFiller 30](#_Toc71630706)

[3.14.10 OrderFillerBean 30](#_Toc71630707)

[3.14.11 OrderReceiver 30](#_Toc71630708)

[3.14.12 POHandler 30](#_Toc71630709)

[3.14.13 POReceiver 30](#_Toc71630710)

[3.14.14 PurchaseOrder 30](#_Toc71630711)

[3.14.15 SErviceException 31](#_Toc71630712)

[3.14.16 Utils 31](#_Toc71630713)

[3.14.17 WebServiceBroker 31](#_Toc71630714)

[3.14.18 WorkflowManagrerBean 31](#_Toc71630715)

[3.15 !Workflowmanager Module Use View 31](#_Toc71630716)

[3.15.1 CreditCardVerifier 31](#_Toc71630717)

[3.15.2 CrmBean 31](#_Toc71630718)

[3.15.3 Financial 31](#_Toc71630719)

[3.15.4 Handlers 32](#_Toc71630720)

[3.15.5 IMessageListener 32](#_Toc71630721)

[3.15.6 IMessageListener 32](#_Toc71630722)

[3.15.7 IOrderFiller 32](#_Toc71630723)

[3.15.8 Invoice 32](#_Toc71630724)

[3.15.9 Invoice 32](#_Toc71630725)

[3.15.10 InvoiceHandler 32](#_Toc71630726)

[3.15.11 Mailer 32](#_Toc71630727)

[3.15.12 OrderFiller 32](#_Toc71630728)

[3.15.13 OrderFillerBean 32](#_Toc71630729)

[3.15.14 OrderReceiver 32](#_Toc71630730)

[3.15.15 POEndpointBean 32](#_Toc71630731)

[3.15.16 POHandler 32](#_Toc71630732)

[3.15.17 POReceiver 33](#_Toc71630733)

[3.15.18 PoWebService 33](#_Toc71630734)

[3.15.19 ProcessManager 33](#_Toc71630735)

[3.15.20 ServiceLocator 33](#_Toc71630736)

[3.15.21 WorkflowManagrerBean 33](#_Toc71630737)

[3.15.22 crm.ejb 33](#_Toc71630738)

[3.16 !EJB Container 34](#_Toc71630739)

[3.16.1 ActivityBean 34](#_Toc71630740)

[3.16.2 AdvendureOPCDB 34](#_Toc71630741)

[3.16.3 ArrivalTime 34](#_Toc71630742)

[3.16.4 BrokerServiceBean 34](#_Toc71630743)

[3.16.5 BrowkerRequestorBean 34](#_Toc71630744)

[3.16.6 CRMQueue 34](#_Toc71630745)

[3.16.7 CreditCardBean 34](#_Toc71630746)

[3.16.8 CrmBean 34](#_Toc71630747)

[3.16.9 GetInvoice 35](#_Toc71630748)

[3.16.10 IActivityPOService 35](#_Toc71630749)

[3.16.11 IAirlinePOService 35](#_Toc71630750)

[3.16.12 ILodgingPOService 35](#_Toc71630751)

[3.16.13 LodgingBean 35](#_Toc71630752)

[3.16.14 ManagerBean 35](#_Toc71630753)

[3.16.15 OTEndpointBean 35](#_Toc71630754)

[3.16.16 OrderFillerBean 35](#_Toc71630755)

[3.16.17 OrderFillerQueue 35](#_Toc71630756)

[3.16.18 POEndpointBean 35](#_Toc71630757)

[3.16.19 Price 35](#_Toc71630758)

[3.16.20 ProcessManagerBean 35](#_Toc71630759)

[3.16.21 PurchaseOrderBean 35](#_Toc71630760)

[3.16.22 SendRequest 35](#_Toc71630761)

[3.16.23 TransportationBean 35](#_Toc71630762)

[3.16.24 WebServiceBroker 35](#_Toc71630763)

[3.16.25 WebServiceBrokerQueue 35](#_Toc71630764)

[3.16.26 WorkflowManagerBean 35](#_Toc71630765)

[3.16.27 WorkflowMgrQueue 37](#_Toc71630766)

[3.16.28 arrivalCity 37](#_Toc71630767)

[3.16.29 arrivaltime 37](#_Toc71630768)

[3.16.30 carrier 37](#_Toc71630769)

[3.16.31 creditCardNumber 37](#_Toc71630770)

[3.16.32 dateTime 37](#_Toc71630771)

[3.16.33 departurCity8 37](#_Toc71630772)

[3.16.34 departurTime 37](#_Toc71630773)

[3.16.35 departureTime 37](#_Toc71630774)

[3.16.36 destination 37](#_Toc71630775)

[3.16.37 expirationDate 37](#_Toc71630776)

[3.16.38 flight\_id 37](#_Toc71630777)

[3.16.39 headCount 37](#_Toc71630778)

[3.16.40 headCount 37](#_Toc71630779)

[3.16.41 headCount 37](#_Toc71630780)

[3.16.42 hotel\_id 37](#_Toc71630781)

[3.16.43 id 37](#_Toc71630782)

[3.16.44 id 37](#_Toc71630783)

[3.16.45 id 37](#_Toc71630784)

[3.16.46 location 37](#_Toc71630785)

[3.16.47 numberOfNight 37](#_Toc71630786)

[3.16.48 numberOfRooms 37](#_Toc71630787)

[3.16.49 orderDateTime 37](#_Toc71630788)

[3.16.50 orderOverallStatus 37](#_Toc71630789)

[3.16.51 order\_id 37](#_Toc71630790)

[3.16.52 ordersStatus 37](#_Toc71630791)

[3.16.53 origin 37](#_Toc71630792)

[3.16.54 ratePernight 37](#_Toc71630793)

[3.16.55 startDate 37](#_Toc71630794)

[3.16.56 totalPrice 37](#_Toc71630795)

[3.16.57 travelClass 37](#_Toc71630796)

[3.16.58 type 37](#_Toc71630797)

[3.16.59 user\_id 37](#_Toc71630798)

[3.16.60 validationCode 37](#_Toc71630799)

[4. D-C&C 37](#_Toc71630800)

[4.1 !Top Level SOA View 37](#_Toc71630801)

[4.1.1 AQS4 38](#_Toc71630802)

[4.1.2 ActivityBookingProvider 38](#_Toc71630803)

[4.1.3 AdvendureOPCDB 38](#_Toc71630804)

[4.1.4 AdventureCatalogDB 38](#_Toc71630805)

[4.1.5 AirlineBookingProvider 38](#_Toc71630806)

[4.1.6 BankProvider 39](#_Toc71630807)

[4.1.7 ConsumerWebsite 39](#_Toc71630808)

[4.1.8 LodgingBookingProvider 39](#_Toc71630809)

[4.1.9 OPC 39](#_Toc71630810)

[4.1.10 PortActivityBookingService 40](#_Toc71630811)

[4.1.11 PortActrivityBookService 40](#_Toc71630812)

[4.1.12 PortAirlineBookingProvider 40](#_Toc71630813)

[4.1.13 PortAirlinePoService 40](#_Toc71630814)

[4.1.14 PortCreditCardService 40](#_Toc71630815)

[4.1.15 PortIWebServiceBroker 40](#_Toc71630816)

[4.1.16 PortJdbc 40](#_Toc71630817)

[4.1.17 PortLodgingPoService 40](#_Toc71630818)

[4.1.18 PortOpcOrderTrackingService 40](#_Toc71630819)

[4.1.19 PortOpcPurchaseOrderService 40](#_Toc71630820)

[4.1.20 QAS2 40](#_Toc71630821)

[4.1.21 QAS3 40](#_Toc71630822)

[4.1.22 QAS5 40](#_Toc71630823)

[4.1.23 QAS6 40](#_Toc71630824)

[4.1.24 QAS7 40](#_Toc71630825)

[4.1.25 ServiceRegistry 40](#_Toc71630826)

[4.1.26 UserEmailClient 40](#_Toc71630827)

[4.1.27 WebBrowser 40](#_Toc71630828)

[4.1.28 bankProviderPort 41](#_Toc71630829)

[4.1.29 lodgingBookingProviderPort 41](#_Toc71630830)

[4.1.30 portSmtp 41](#_Toc71630831)

[4.2 !Activity Booking Provider Component Detail 41](#_Toc71630832)

[4.2.1 IActivityBookingService 41](#_Toc71630833)

[4.2.2 PortActrivityBookService 41](#_Toc71630834)

[4.2.3 activityBookingAdapter 41](#_Toc71630835)

[4.2.4 iActivityBookingService 41](#_Toc71630836)

[4.3 !Activity Booking Provider Component Detail 41](#_Toc71630837)

[4.4 !Airline Booking Provider component detail 41](#_Toc71630838)

[4.4.1 PortAirlineBookingProvider 41](#_Toc71630839)

[4.4.2 PortWebServiceBroker 41](#_Toc71630840)

[4.4.3 airlineBookingAdapter0 42](#_Toc71630841)

[4.4.4 iAirlineBookingService0 42](#_Toc71630842)

[4.5 !Airline Booking Provider component detail 42](#_Toc71630843)

[4.6 !Bank Provider Component detail 42](#_Toc71630844)

[4.6.1 bankProviderPort 42](#_Toc71630845)

[4.6.2 iCreditCardService 42](#_Toc71630846)

[4.6.3 paypayAdapter 42](#_Toc71630847)

[4.6.4 strypeAdapter 42](#_Toc71630848)

[4.7 !Bank Provider Component detail 42](#_Toc71630849)

[4.8 !OPC C&C View 42](#_Toc71630850)

[4.8.1 ActivityBean 43](#_Toc71630851)

[4.8.2 BrokerRequestorBean 43](#_Toc71630852)

[4.8.3 BrokerServiceBean 43](#_Toc71630853)

[4.8.4 CRMBean 43](#_Toc71630854)

[4.8.5 CreditCardBean 43](#_Toc71630855)

[4.8.6 LodgingBean 43](#_Toc71630856)

[4.8.7 ManagerBean 43](#_Toc71630857)

[4.8.8 OTEndpointBean 43](#_Toc71630858)

[4.8.9 OrderFillerBean 43](#_Toc71630859)

[4.8.10 POEndpointBean 43](#_Toc71630860)

[4.8.11 PortActivityBookingService 44](#_Toc71630861)

[4.8.12 PortAirlinePoService 44](#_Toc71630862)

[4.8.13 PortBrokerServiceBean 44](#_Toc71630863)

[4.8.14 PortCreditCardService 44](#_Toc71630864)

[4.8.15 PortDataAccessIntegrity 44](#_Toc71630865)

[4.8.16 PortIWebServiceBroker 44](#_Toc71630866)

[4.8.17 PortJdbc 44](#_Toc71630867)

[4.8.18 PortLodgingPoService 44](#_Toc71630868)

[4.8.19 PortManagerBean 44](#_Toc71630869)

[4.8.20 PortOpcOrderTrackingService 44](#_Toc71630870)

[4.8.21 PortOpcPurchaseOrderService 44](#_Toc71630871)

[4.8.22 PortProcessManagerBean 44](#_Toc71630872)

[4.8.23 PurchaseOrderBean 44](#_Toc71630873)

[4.8.24 TransportationBean 44](#_Toc71630874)

[4.8.25 WorkFlowManagerBean 44](#_Toc71630875)

[4.8.26 portSmtp 44](#_Toc71630876)

[4.8.27 processManagerBean 44](#_Toc71630877)

[4.9 !OPC C&C View 45](#_Toc71630878)

[5. E-Allocation 45](#_Toc71630879)

[5.1 !singleDeployement 45](#_Toc71630880)

[5.1.1 AdminUserMachiine 45](#_Toc71630881)

[5.1.2 AdventureDBCatalog 45](#_Toc71630882)

[5.1.3 ServiceRegistry 45](#_Toc71630883)

[5.1.4 UserMailClient 45](#_Toc71630884)

[5.1.5 WebBrowser 45](#_Toc71630885)

[5.1.6 activityProviderServerMachine 45](#_Toc71630886)

[5.1.7 arilineProviderServerMachine 45](#_Toc71630887)

[5.1.8 bankServerMachine 45](#_Toc71630888)

[5.1.9 endUserMachine 46](#_Toc71630889)

[5.1.10 lodgingProviderServerMachine 46](#_Toc71630890)

[5.1.11 svr-opc 46](#_Toc71630891)

[5.1.12 svr-web1 46](#_Toc71630892)

[5.1.13 web1B 46](#_Toc71630893)

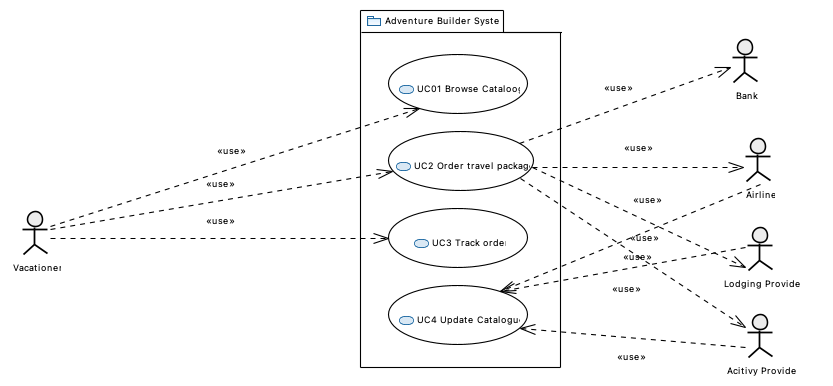
[5.1.14 web1a 46](#_Toc71630894)

[5.1.15 website.ear 46](#_Toc71630895)

[6. F-Comportement 46](#_Toc71630896)

# A-Requirements

## 1-!!Use cases view



### Acitivy Provider

### Adventure Builder System

### Airline

### Bank

### Lodging Provider

### UC01 Browse Cataloog

The user can visit the Adventure Builder Web site and browse the catalog of travel packages, which include flights to specific destinations, lodging options, and activities that can be purchased in advance. Activities include mountain biking, fishing, surfing classes, hot air balloon tours, and scuba diving.

### UC2 Order travel package

The user can place an order for a vacation package. To process this order, the system has to interact with several external entities. A bank will approve the customer payment, airline companies will provide the flights, lodging providers will book the hotel rooms, and businesses that provide vacation activities will schedule the activities selected by the customer.

### UC3 Track order

After an order is placed, the user can return to check the status of his/her order. This is necessary because some interactions with external entities are processed in the background and may take hours or days to complete.

After an order is placed, the user can return to check the status of his/her order. This is necessary because some interactions with external entities are processed in the background and may take hours or days to complete.

### UC4 Update Catalogue

The internal system periodically interacts with its business partners (transportation, lodging, and activity providers) to update the catalog with the most recent offerings.

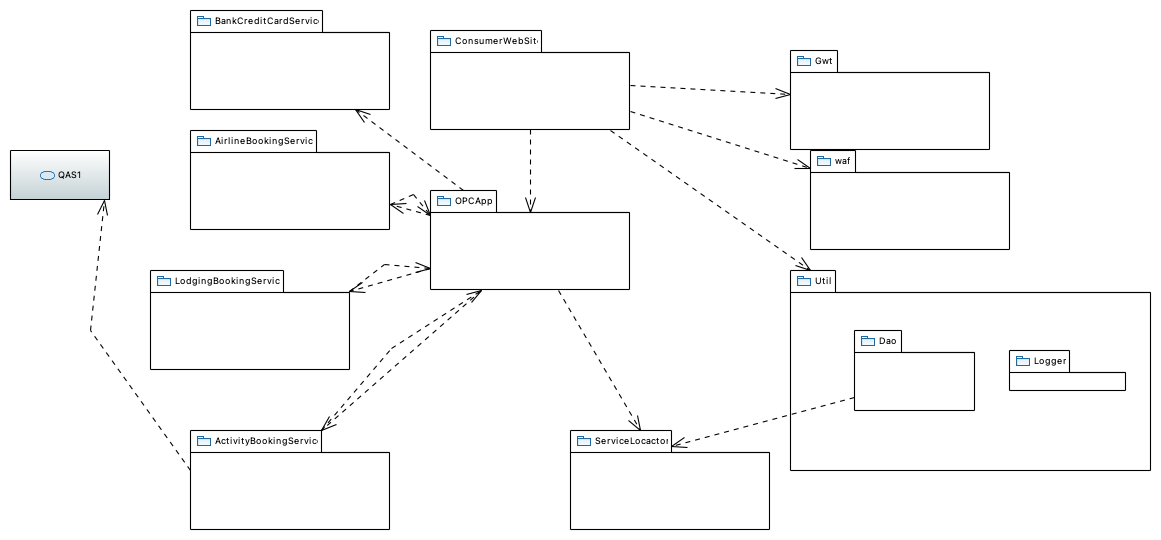
### Vacationer

The user can visit the Adventure Builder Web site and browse the catalog of travel packages, which include flights to specific destinations, lodging options, and activities that can be purchased in advance. Activities include mountain biking, fishing, surfing classes, hot air balloon tours, and scuba diving. The user can select transportation, accommodation, and various activities to build his/her own adventure trip.  
   
 The user can place an order for a vacation package. To process this order, the system has to interact with several external entities. A bank will approve the customer payment, airline companies will provide the flights, lodging providers will book the hotel rooms, and businesses that provide vacation activities will schedule the activities selected by the customer.  
   
 After an order is placed, the user can return to check the status of his/her order. This is necessary because some interactions with external entities are processed in the background and may take hours or days to complete.  
   
 The internal system periodically interacts with its business partners (transportation, lodging, and activity providers) to update the catalog with the most recent offerings.

# B-ACOO

# C-Modules

## !!!Top Level Modules Uses View



### ActivityBookingService

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

### AirlineBookingService

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

### BankCreditCardService

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

Must support mecanism to connect Adventure builder to multiple bank service provider for billing purpose

### 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.

### 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.

### 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.

### LodgingBookingService

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

### Logger

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

### OPCApp

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

### QAS1

. A new business partner (airline, lodging, or activity provider) that uses its own web services interface is added to the system in no more than 10 person-days of effort for the implementation. The business goal is easy integration with new business partners.

### ServiceLocactor

This module is an implementation of the ServiceLocator design pattern.

### Util

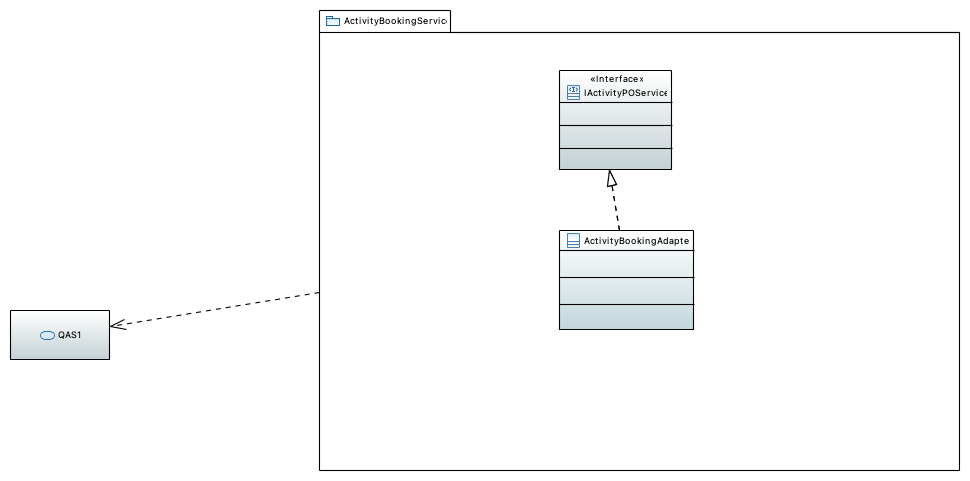
See JMS tutorial at

This module contains utilities used by the Adventure Builder system.

### 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.

## !ActivityBookingService



### ActivityBookingAdapter

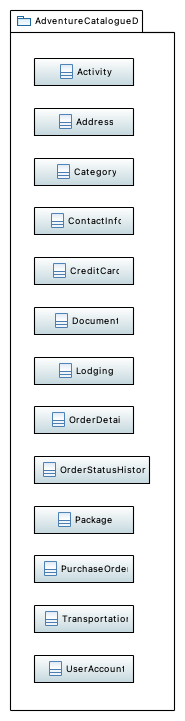
### IActivityPOService

- This is used to send purchaseorders to external activity suppliers.

### QAS1

. A new business partner (airline, lodging, or activity provider) that uses its own web services interface is added to the system in no more than 10 person-days of effort for the implementation. The business goal is easy integration with new business partners.

## !AdventureCatalogDBDetails class Diagram



### Activity

### Address

### Category

### ContactInfo

### CreditCard

### Document

### Lodging

### OrderDetail

### OrderStatusHistory

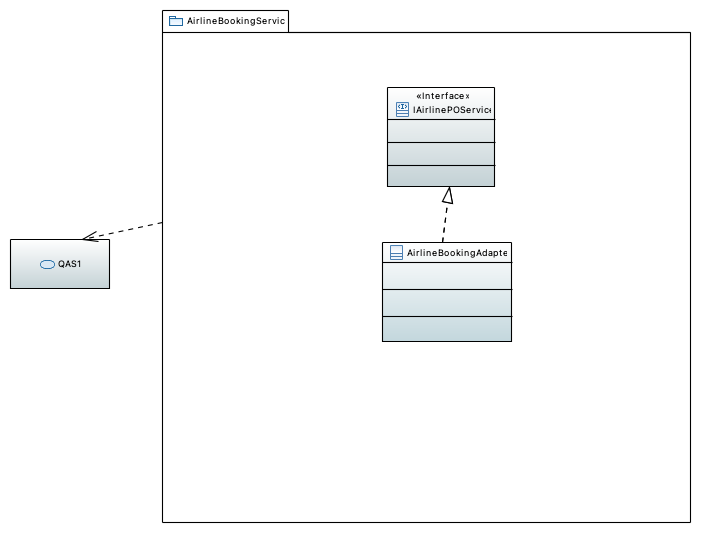
### Package

### PurchaseOrder

### Transportation

### UserAccount

## !AirlineBookingService package detail



### AirlineBookingAdapter

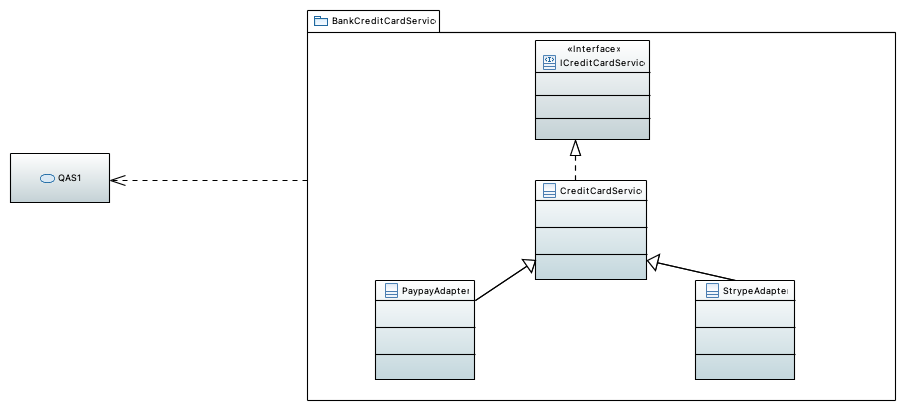
### IAirlinePOService

This is used to send purchaseorders to external airline suppliers.

### QAS1

. A new business partner (airline, lodging, or activity provider) that uses its own web services interface is added to the system in no more than 10 person-days of effort for the implementation. The business goal is easy integration with new business partners.

## !Bank Credit Card Service class Diagram



### CreditCardService

### ICreditCardService

### PaypayAdapter

Adapter to use Paypal.com service for billing

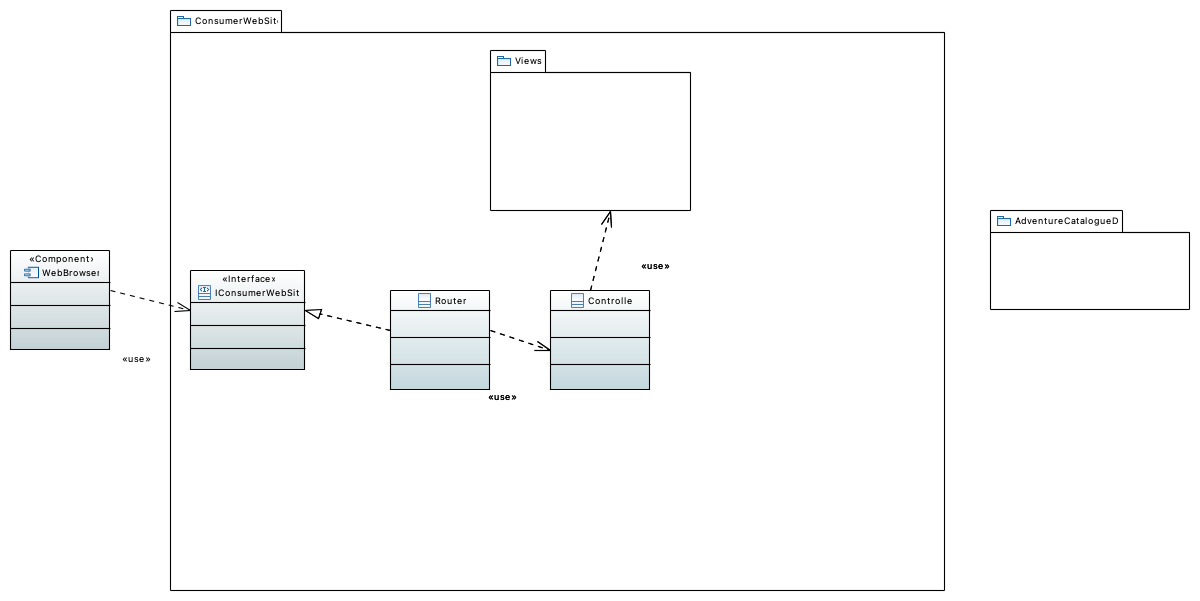
### QAS1

. A new business partner (airline, lodging, or activity provider) that uses its own web services interface is added to the system in no more than 10 person-days of effort for the implementation. The business goal is easy integration with new business partners.

### StrypeAdapter

Adapteur to use Stripe.com service for billing

## !ConsumerWebSite class diagram



### AdventureCatalogueDB

### Controller

### IConsumerWebSite

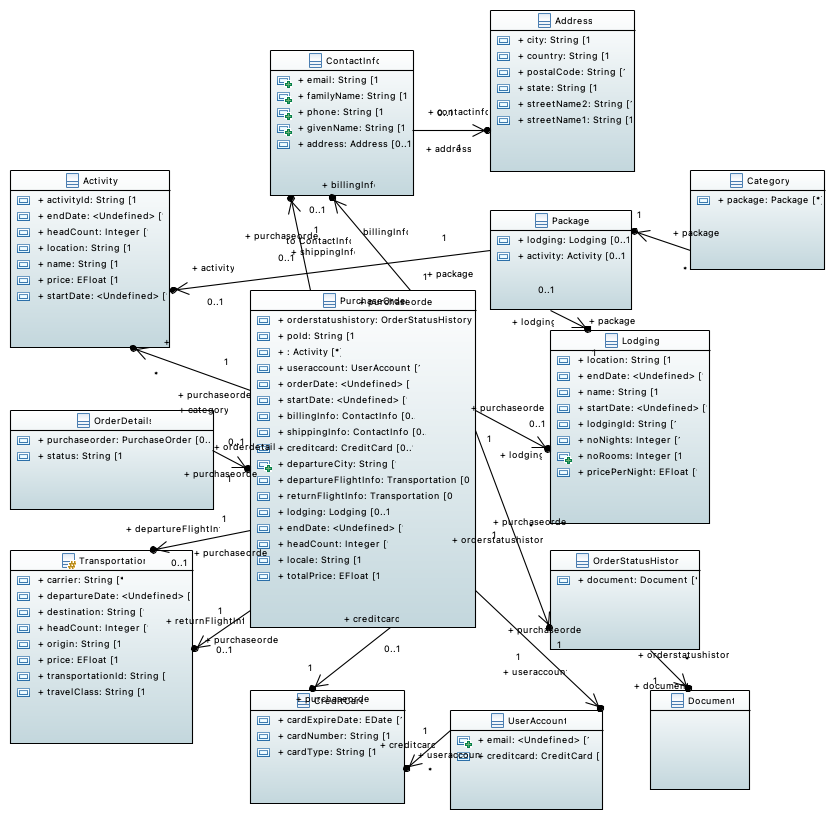
### Router

### Views

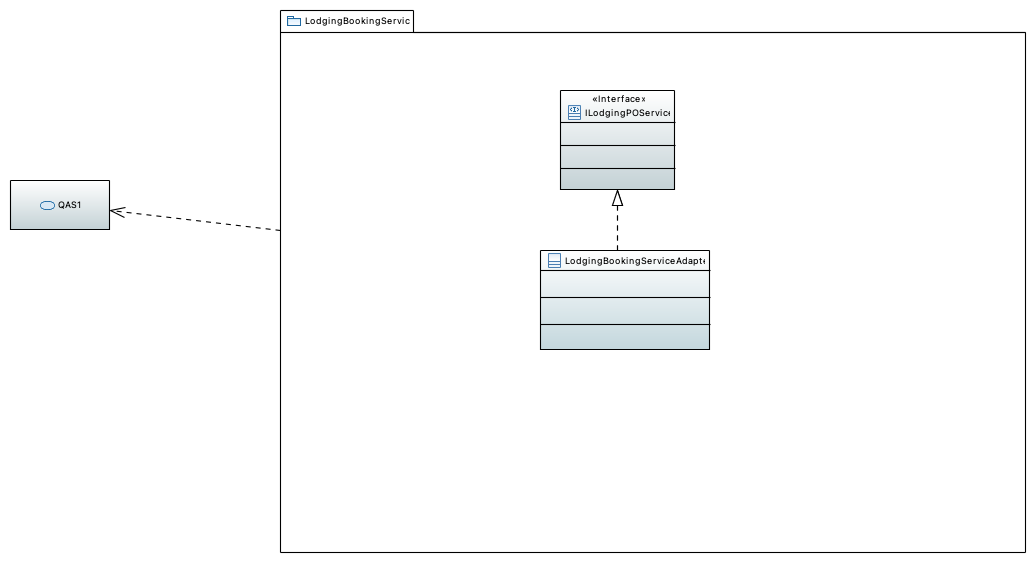
### WebBrowser

This component represents the user interface of the application running on a web browser. Consumer Website is a Web 2.0 application implemented using GWT. Therefore, in addition to HTML, the web browser runs JavaScript code that uses Ajax to communicate with the server. Using the web browser, a customer of Adventure Builder opens the web site, browses through the existing catalog of adventure packages, places orders, and tracks the status of existing orders.

## !Model class diagram



## !LodgingBookingService package detail



### ILodgingPOService

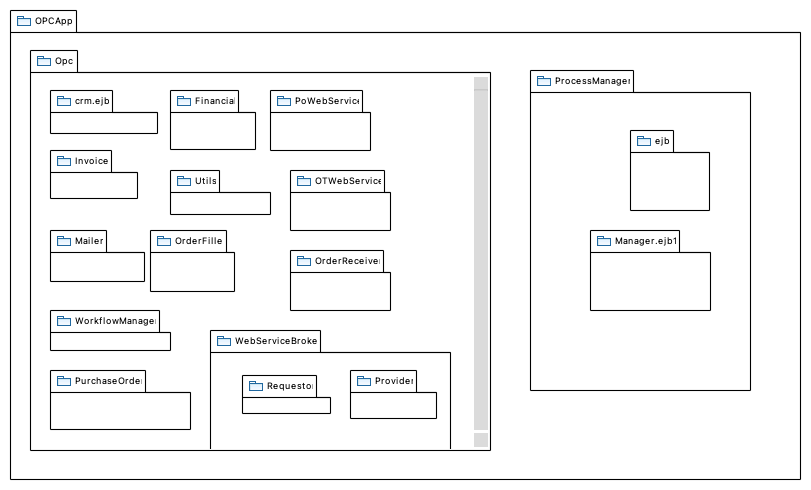
This is used to send purchaseorders to external lodging suppliers.

### LodgingBookingServiceAdapter

### QAS1

. A new business partner (airline, lodging, or activity provider) that uses its own web services interface is added to the system in no more than 10 person-days of effort for the implementation. The business goal is easy integration with new business partners.

## !OPCApp Module Decomposition View



### 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.

### 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).

### 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.

### Manager.ejb1

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.

### 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

### Opc

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

Order processing center

### 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.

### 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

### 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.

### 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.

### 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.

### 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.

### 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.

### 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.

### 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.

### crm.ejb

\* minimum (and initial) number of bean instances in the pool

For each EJb, a pool of bean instances is provided by the application server. There are three parameters that can be configured separately for each EJB via deployment descriptor:

\* maximum number of bean instances in the pool

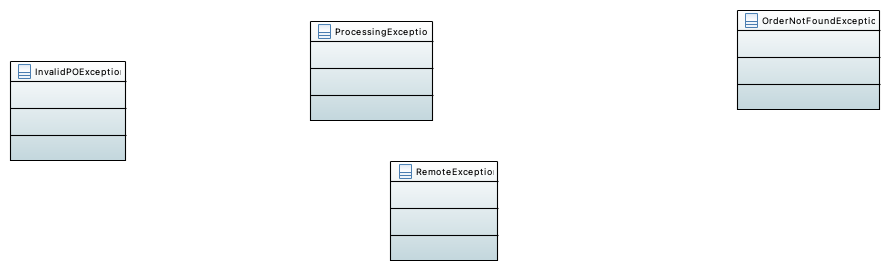
This is the Customer Relationship Manager (CRM) module. The job of this module is to send out an email once an order has been completely and successfully processed. It is implemented as a message-driven bean. In the future this module can hold additional information about customers that could assist 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.

\* timeout for an idle instance to be passivated or deleted

### 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).

## Exception class diagram



### InvalidPOException

The service throws this exception if the purchase order is null, or if any of the components that make up the purchase order is null.

### OrderNotFoundException

The service throws this exception if no purchase order with the given order id was found, or the orderId was null or ill-formed.

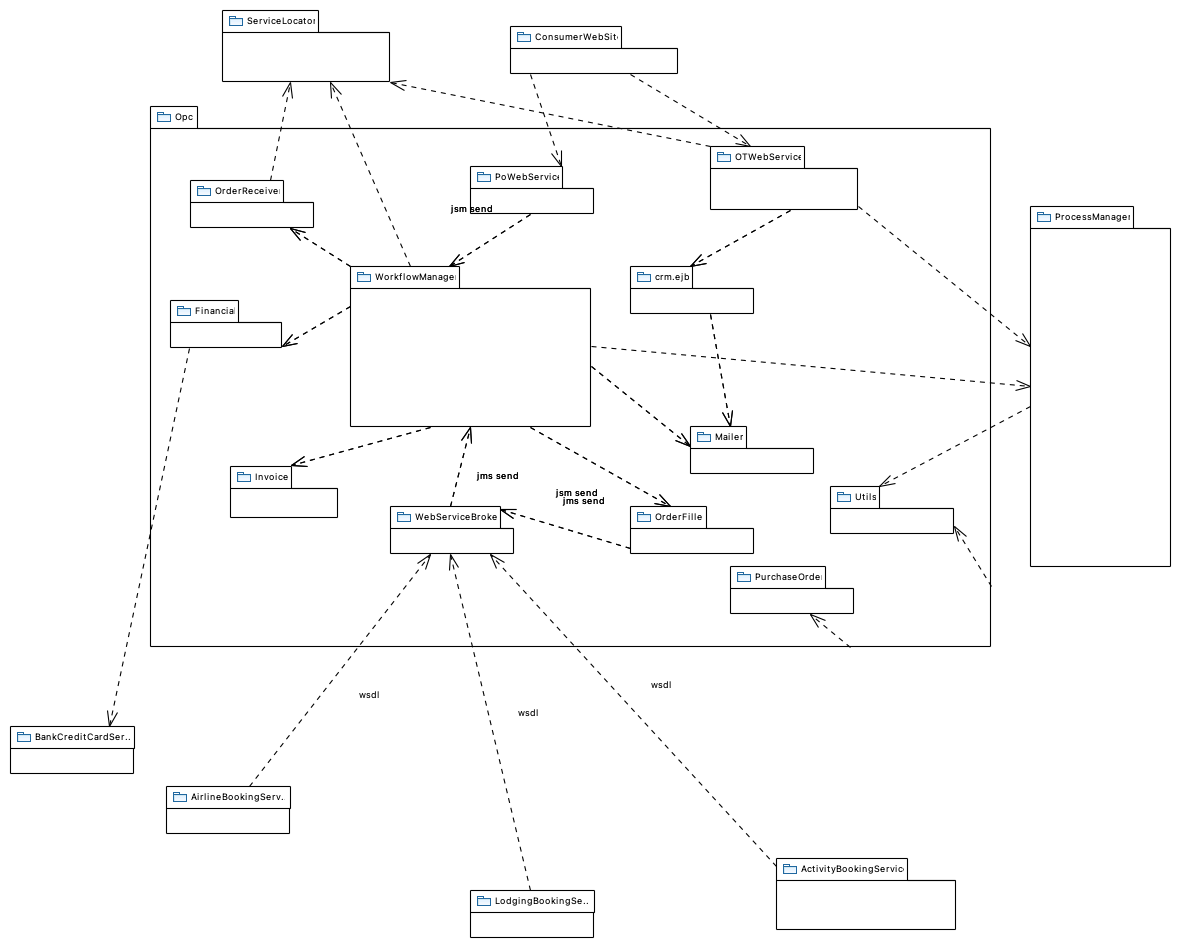
### ProcessingException

If the processing of the purchase order fails after it has passed its validation (i.e., there is no InvalidPOException), then a ProcessingException is thrown. This indicates that the purchase order is valid but something has gone wrong while placing the order.

### RemoteException

The caller receives a RemoteException when there is a communication problem with the service provider implementing this interface.

## !OPC MODULE Uses View



### ActivityBookingService

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

### AirlineBookingService

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

### BankCreditCardService

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

Must support mecanism to connect Adventure builder to multiple bank service provider for billing purpose

### 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.

### 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.

### 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).

### LodgingBookingService

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

### 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.

### 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

### 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.

### 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

### 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.

### 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.

### ServiceLocator

TODO

### 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.

### 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.

### 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.

### crm.ejb

\* minimum (and initial) number of bean instances in the pool

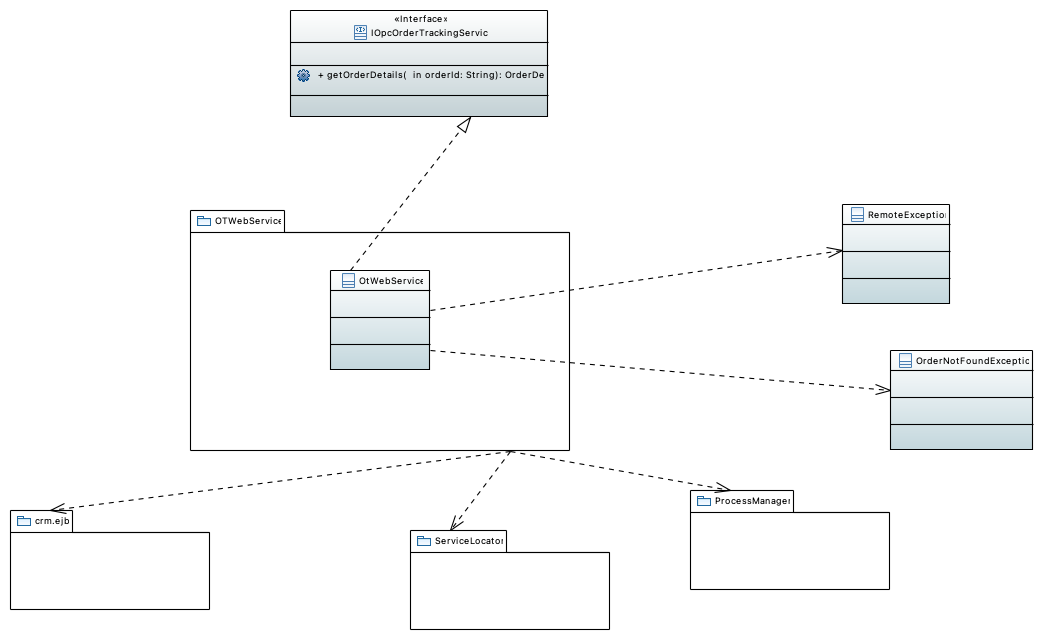
For each EJb, a pool of bean instances is provided by the application server. There are three parameters that can be configured separately for each EJB via deployment descriptor:

\* maximum number of bean instances in the pool

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.

\* timeout for an idle instance to be passivated or deleted

## !OtWebService package diagram



### IOpcOrderTrackingService

This is a SOAP web service interface named OpcOrderTrackingService. The main purpose of this web service is to track the status of a purchase order with a given order id.

### OrderNotFoundException

The service throws this exception if no purchase order with the given order id was found, or the orderId was null or ill-formed.

### OtWebService

### 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.

### RemoteException

The caller receives a RemoteException when there is a communication problem with the service provider implementing this interface.

### ServiceLocator

TODO

### crm.ejb

\* minimum (and initial) number of bean instances in the pool

For each EJb, a pool of bean instances is provided by the application server. There are three parameters that can be configured separately for each EJB via deployment descriptor:

\* maximum number of bean instances in the pool

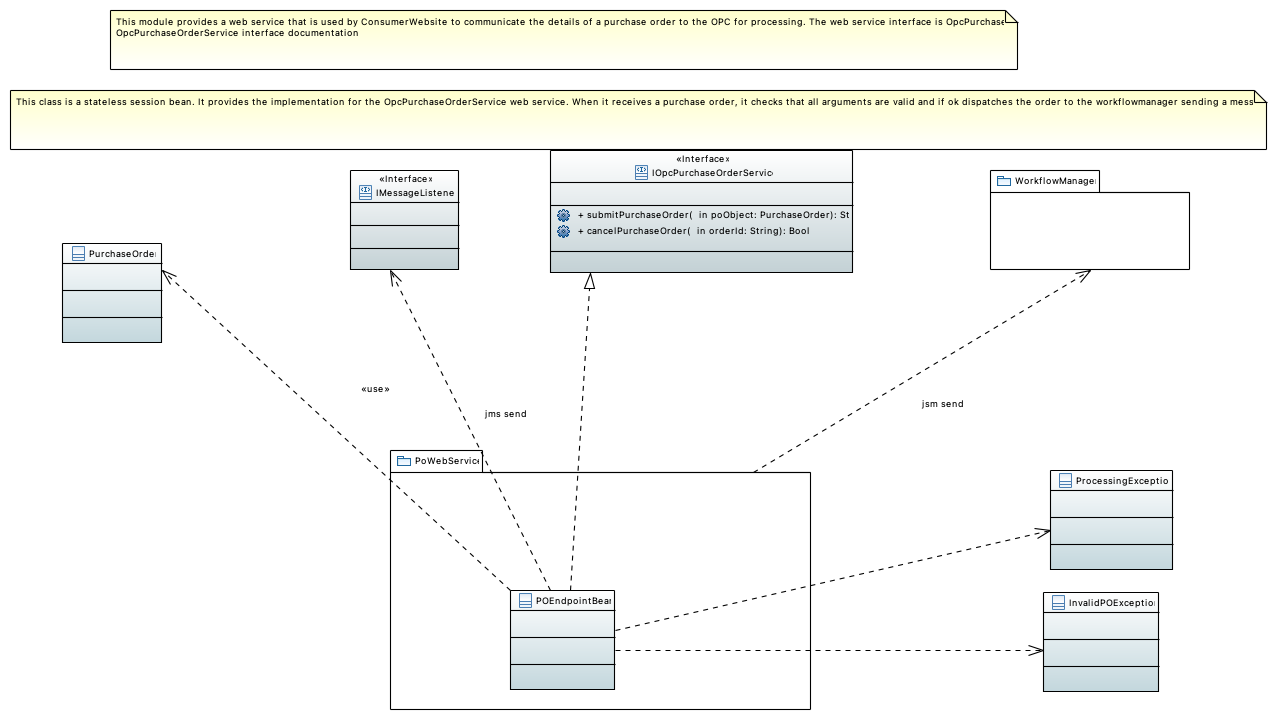
This is the Customer Relationship Manager (CRM) module. The job of this module is to send out an email once an order has been completely and successfully processed. It is implemented as a message-driven bean. In the future this module can hold additional information about customers that could assist 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.

\* timeout for an idle instance to be passivated or deleted

### getOrderDetails

Pre-conditions \* The orderID argument must not be null. \* The orderID must be a valid order id. Post-conditions \* Return the details of the purchase order, which include the status of each of the airline requests (both departing and returning flights), hotel booking request and activity requests. Usage restrictions \* Authorization. Only a signed in user is authorized to call this operation. A customer user can only get information about orders he created. A sales representative user can get information about any order. \* Concurrent access. There isn't any limitation on the number of concurrent accesses of this interface. The service is implemented by using EJBs and concurrent calls to it are managed by the EJB container.

## !POWebService class diagram



### IMessageListener

### IOpcPurchaseOrderService

Error handling The exceptions raised by this interface are: \* InvalidPOException. The service throws this exception if the purchase order is null, or if any of the components that make up the purchase order are null. \* ProcessingException. If the processing of the purchase order fails after it has passed its validation, i.e. there is no InvalidPOException, then a ProcessingException is thrown. This indicates that the purchase order is valid but something has gone wrong while placing the order. \* RemoteException. A RemoteException is thrown when there is an error in connection. Quality attribute characteristics of the interface \* Scalability. This interface is internally implemented using Enterprise Java Beans and therefore it is scalable (allows scaling of number of requests it can handle). Also, this service is asynchronous in nature, which means the client can continue after making a call to it. \* Security. Security is important since credit card information is part of the PurchaseOrder argument. We choose https for our security requirements because that increases inter-operability (TODO: Not sure if this is a good enough reason, although we did use web services to increase inter-operability. So, in that sense it is a valid reason). Rationale and design issues Java-WSDL approach for design of interface We chose the Java-WSDL approach for design of this interface because we don't care much for the stability of the web service. This is because both the consumer website and the order processing center reside in the same enterprise. Using the EJB endpoint type We chose the EJB endpoint type because the order processing center is implemented using a set of session beans. Coarse-grained granularity of service Currently we only expose placing a purchase order as a single web service that includes placing orders for activities, transportation and lodging. We don't provide these services individually and hence have no reason to provide fine-grained versions of the web services. Using JAX-RPC for passing parameters Since the consumer website and the order processing center reside in the same enterprise, we avoid using complex XML processing and pass parameters as Java objects. The return type is a simple String that is the purchase order's unique id. Publishing the web service as a WSDL This web service is published as a WSDL in a well known location since it is not available for general public use. It is only used by the consumer website.

### InvalidPOException

The service throws this exception if the purchase order is null, or if any of the components that make up the purchase order is null.

### POEndpointBean

This class is a stateless session bean. It provides the implementation for the OpcPurchaseOrderService web service. When it receives a purchase order, it checks that all arguments are valid and if ok dispatches the order to the workflowmanager sending a message to a JMS queue.

### ProcessingException

If the processing of the purchase order fails after it has passed its validation (i.e., there is no InvalidPOException), then a ProcessingException is thrown. This indicates that the purchase order is valid but something has gone wrong while placing the order.

### PurchaseOrder

### 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.

### cancelPurchaseOrder

Cancel the given purchase order. The operation returns true if the order was canceled successfully; it returns false when the purchase order processing is in a state where some bookings are already confirmed and cannot be canceled.

<b>Pre-conditions</b>

\* The orderId argument is not null and corresponds to an existing purchase order. <b>Post-conditions</b>

\* If the operation returns true, the order is canceled and all reservations corresponding to this order are canceled. Charges to the customer credit card, if any were made, are also canceled.

\* If the operation returns false, the state of the order remains unchanged.   
 <b>Usage restrictions</b>

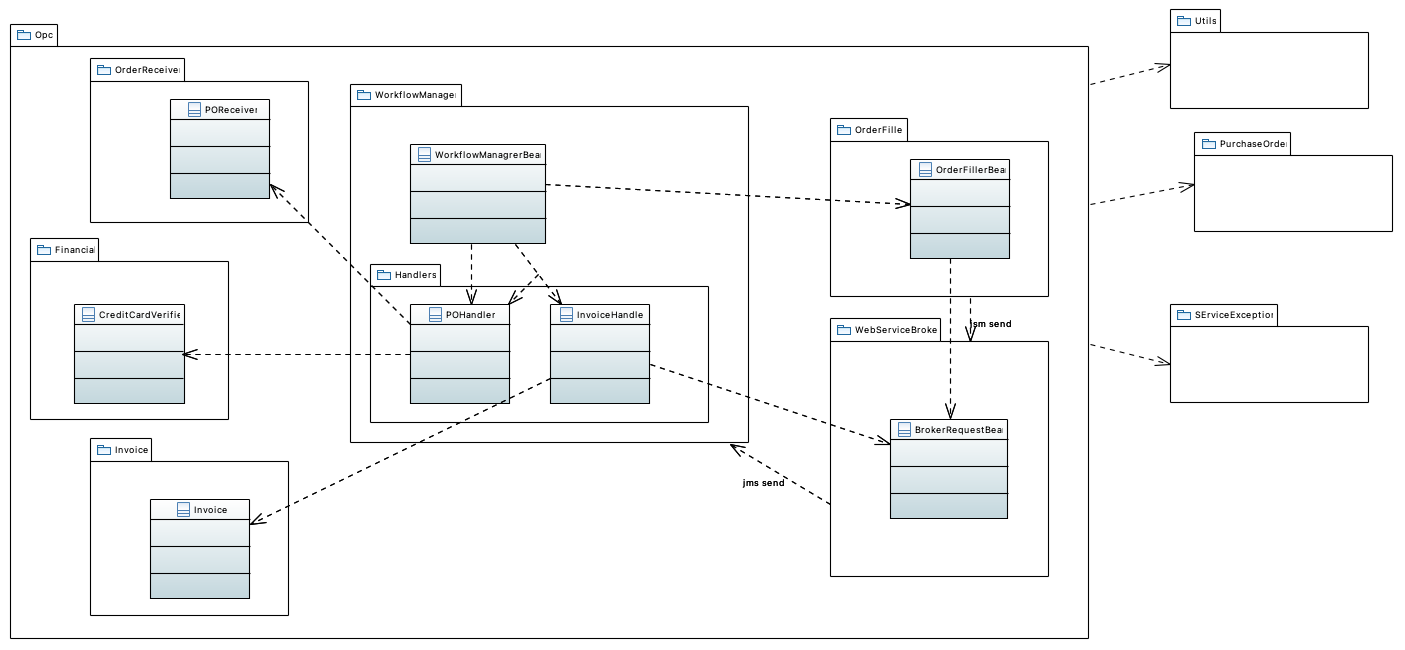
\* Authorization. This operation can only be called by a signed in user. A customer user can only cancel purchase orders he created. A sales representative user can cancel any order.

<b>Error handling</b> \* <b>InvalidPOException.</b> The service throws this exception if the order ID is null or the corresponding order doesn't exist. \* <b>ProcessingException.</b> Thrown if the order exists but is already canceled.

### submitPurchaseOrder

Submit a complete purchase order for an adventure package. This operation returns the order ID as a String. The operation simply checks if the PurchaseOrder argument is valid. If it is valid, the order is created in the database and this operation returns to the caller. The actual processing of the purchase order is carried on in background. If there is some unforeseen error during the processing of the purchase order after this operation has returned to the caller, the error is handled elsewhere and is not the responsibility of this interface. <b>Pre-conditions</b> \* The PurchaseOrder argument must not be null. \* The following components of the PurchaseOrder argument must not be null: user-id, email-id, locale, order date, shipping information, billing information, total price, credit card information, head count information, start date, end date and departure city <b>Post-conditions</b> \* A successful call to this interface will return a unique purchase order id. <b>Usage restrictions</b> \* Authorization. Can only be called by signed in users. \* Concurrent access. There isn't any limitation on the number of concurrent accesses of this interface. The service is implemented by using EJBs and concurrent calls to it are managed by the EJB container. <b>Error handling</b> \* InvalidPOException. The service throws this exception if the purchase order is null, or if any of the components that make up the purchase order is null. \* ProcessingException. If the processing of the purchase order fails after it has passed its validation (i.e., there is no InvalidPOException), then a ProcessingException is thrown. This indicates that the purchase order is valid but something has gone wrong while placing the order.

## !!Package WorkflowManager



### BrokerRequestBean

### CreditCardVerifier

This class is used to verify the credit information for a user. It contacts an external web service to get the banking related information for the credit card number specified.

### 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.

### Handlers

### 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).

### Invoice

This "value object" class holds the data for an invoice that arrives from any of the external suppliers. It has a method to provide an XML representation of the invoice.

### InvoiceHandler

The InvoiceHandler is not visible outside the workflowmanager package. It acts as a delegate of the WorkflowManagerBean class to handle any invoice it receives from any of the suppliers. When an invoice is received, InvoiceHandler basically uses processmanager to update the status of the corresponding order. \* A given adventure package order may consist of: \* zero or one hotel booking \* zero, one or two airline flights (departing and returning flight) \* zero or more activity items When InvoiceHandler receives the last invoice confirming completion of the reservation, it sends a JMS message to CrmBean to notify the customer via email.

### Opc

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

Order processing center

### 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.

### OrderFillerBean

This class is a message-driven bean. It is responsible for sending out all the booking requests to the airline, lodging and activity suppliers involved in a given purchase order.

### OrderReceiver

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

### POHandler

The PoHandler class is not visible outside the workflowmanager package. It acts as a delegate of the WorkFlowManagerBean class to handle any purchase order request. When processing a request, it first uses POReceiver and processmanager to insert the order in the database in the Pending state. Then it calls CreditCardVerifier synchronously to charge the customer's credit card. If the credit card is OK, it sends a message to a JMS queue to be processed by OrderFillerBean. Finally, it sends another JMS message to CrmBean, which will create and send an email to the customer informing about the status of his order.

### POReceiver

The purchase order receiver's responsibility is to create a purchase order entity bean and persist it in a relational database. The POReceiver is implemented as a Java class and it creates a PurchaseOrder entity bean.

### 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.

### SErviceException

### 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.

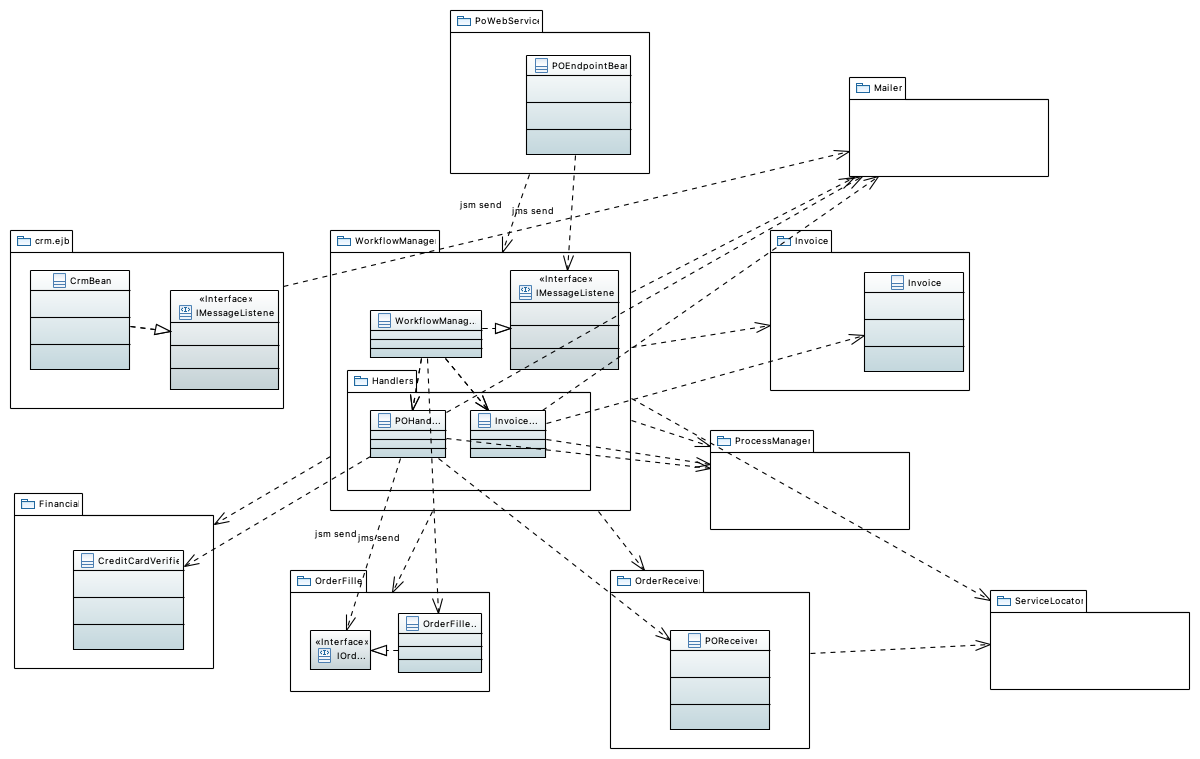
### WebServiceBroker

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.

### WorkflowManagrerBean

This class is a message-driven bean. It is activated when there is a message in the queue. It processes two kinds of messages: \* purchase order message: in this case it calls the PoHandler class. The sequence diagram down below shows the interactions involved in processing a purchase order. \* invoice message: this is a message that came from one of the external suppliers in response for a booking order. WorkFlowManagerBean calls the InvoiceHandler class to take care of this messages. WorkFlowManagerBean also sets a timer with the EJB container so that it is activated periodically to check the status of all pending orders.

## !Workflowmanager Module Use View



### CreditCardVerifier

This class is used to verify the credit information for a user. It contacts an external web service to get the banking related information for the credit card number specified.

### CrmBean

This component is used for customer relationship management. For this application it is only used to communicate with the user. It reads messages from a queue, creates the corresponding email messages according to templates and I18N requirements, and sent them to users.

This class is a message-driven bean. It is responsible for handling the communication with customers via email.

### 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.

### Handlers

### IMessageListener

### IMessageListener

### IOrderFiller

### 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).

### Invoice

This "value object" class holds the data for an invoice that arrives from any of the external suppliers. It has a method to provide an XML representation of the invoice.

### InvoiceHandler

The InvoiceHandler is not visible outside the workflowmanager package. It acts as a delegate of the WorkflowManagerBean class to handle any invoice it receives from any of the suppliers. When an invoice is received, InvoiceHandler basically uses processmanager to update the status of the corresponding order. \* A given adventure package order may consist of: \* zero or one hotel booking \* zero, one or two airline flights (departing and returning flight) \* zero or more activity items When InvoiceHandler receives the last invoice confirming completion of the reservation, it sends a JMS message to CrmBean to notify the customer via email.

### 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.

### 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.

### OrderFillerBean

This class is a message-driven bean. It is responsible for sending out all the booking requests to the airline, lodging and activity suppliers involved in a given purchase order.

### OrderReceiver

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

### POEndpointBean

This class is a stateless session bean. It provides the implementation for the OpcPurchaseOrderService web service. When it receives a purchase order, it checks that all arguments are valid and if ok dispatches the order to the workflowmanager sending a message to a JMS queue.

### POHandler

The PoHandler class is not visible outside the workflowmanager package. It acts as a delegate of the WorkFlowManagerBean class to handle any purchase order request. When processing a request, it first uses POReceiver and processmanager to insert the order in the database in the Pending state. Then it calls CreditCardVerifier synchronously to charge the customer's credit card. If the credit card is OK, it sends a message to a JMS queue to be processed by OrderFillerBean. Finally, it sends another JMS message to CrmBean, which will create and send an email to the customer informing about the status of his order.

### POReceiver

The purchase order receiver's responsibility is to create a purchase order entity bean and persist it in a relational database. The POReceiver is implemented as a Java class and it creates a PurchaseOrder entity bean.

### 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

### 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.

### ServiceLocator

TODO

### WorkflowManagrerBean

This class is a message-driven bean. It is activated when there is a message in the queue. It processes two kinds of messages: \* purchase order message: in this case it calls the PoHandler class. The sequence diagram down below shows the interactions involved in processing a purchase order. \* invoice message: this is a message that came from one of the external suppliers in response for a booking order. WorkFlowManagerBean calls the InvoiceHandler class to take care of this messages. WorkFlowManagerBean also sets a timer with the EJB container so that it is activated periodically to check the status of all pending orders.

### crm.ejb

\* minimum (and initial) number of bean instances in the pool

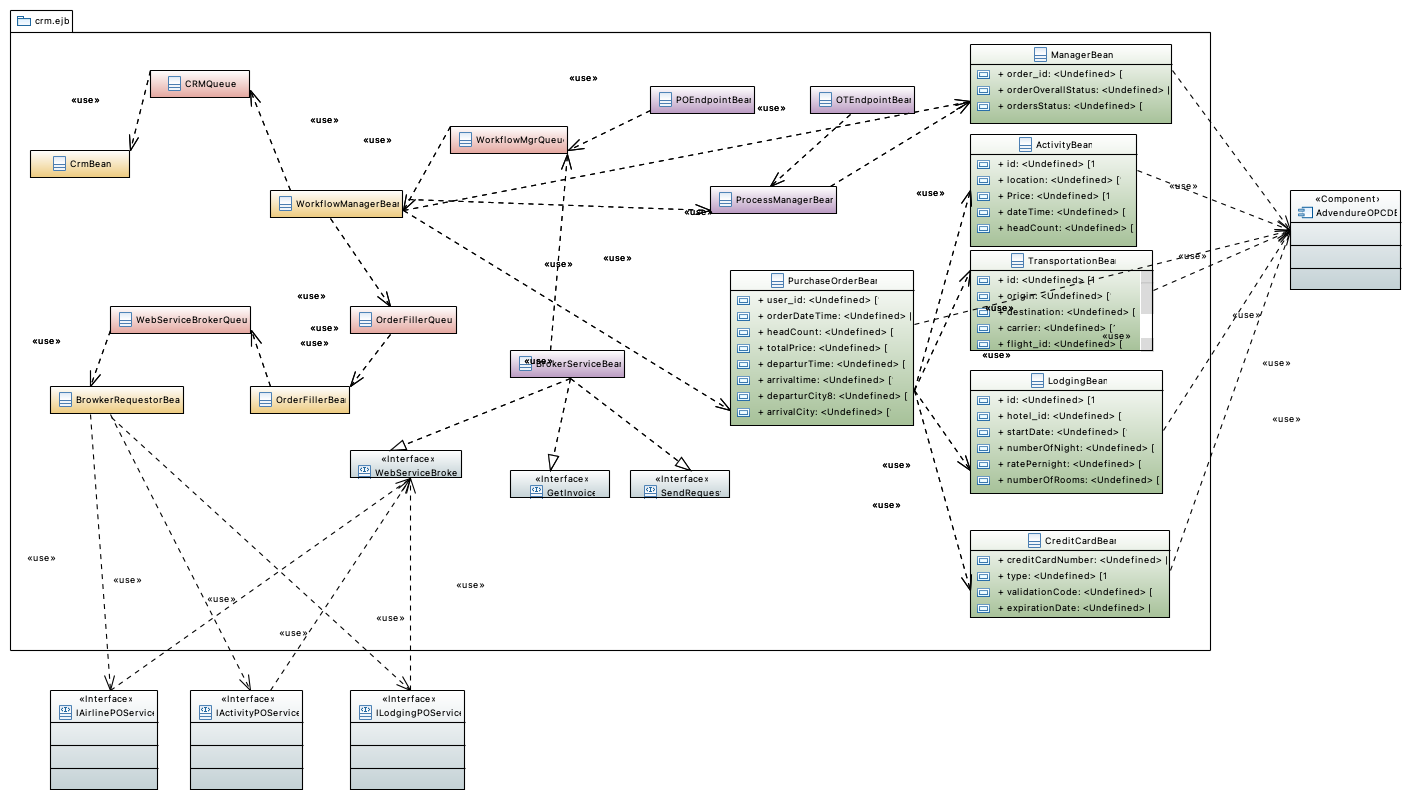
For each EJb, a pool of bean instances is provided by the application server. There are three parameters that can be configured separately for each EJB via deployment descriptor:

\* maximum number of bean instances in the pool

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.

\* timeout for an idle instance to be passivated or deleted

## !EJB Container



### ActivityBean

This entity bean persists the details of an activity reservation (activity id, location, price, date/time, head count).

### AdvendureOPCDB

This relational database stores purchase orders, invoices coming from the external suppliers and related information. The database server is MySQL configured to use the InnoDB engine. Variability guide To configure data storage such that it supports different database vendors, we don't want to use any SQL statements in our queries that use vendor specific syntax and can potentially hinder migrating databases. We also use the DAO pattern to abstract away clients from accessing data directly.

### ArrivalTime

### BrokerServiceBean

\* getInvoice - This interface can provide the invoices that are sent by external suppliers.

\* WebServiceBroker - This is a web service that is used by the external suppliers to submit invoices back to the OPC.

\* sendRequest - This interface can be used to contact external suppliers and place orders with them.

This component is the OPC's window to external suppliers for activities, lodging and transportation. It provides the following interfaces:

### BrowkerRequestorBean

This message driven bean makes requests to external suppliers. It requires the following interfaces:

\* LodgingPOService - This is used to send purchaseorders to external lodging suppliers.

\* AirlinePOService - This is used to send purchaseorders to external airline suppliers.

\* ActivityPOService - This is used to send purchaseorders to external activity suppliers.

### CRMQueue

### CreditCardBean

This entity bean persists the credit card information (credit card number, type, expiration date, etc.) of a user.

### CrmBean

This component is used for customer relationship management. For this application it is only used to communicate with the user. It reads messages from a queue, creates the corresponding email messages according to templates and I18N requirements, and sent them to users.

This class is a message-driven bean. It is responsible for handling the communication with customers via email.

### GetInvoice

getInvoice - This interface can provide the invoices that are sent by external suppliers.

### IActivityPOService

- This is used to send purchaseorders to external activity suppliers.

### IAirlinePOService

This is used to send purchaseorders to external airline suppliers.

### ILodgingPOService

This is used to send purchaseorders to external lodging suppliers.

### LodgingBean

This entity bean persists the details of a hotel reservation (hotel id, start date, number of nights, rate per night, number of rooms).

### ManagerBean

This entity bean is used by WorkFlowManagerBean to persist the status of the purchase order containing order id, order overall status, and the status of each separate supplier order.

### OTEndpointBean

### OrderFillerBean

\* It provides an interface called sendPO that is used by the workflowmanager to send purchaseorders.

\* It requires an interface sendRequest that it uses to send purchaseorders to each individual supplier.

This component is used to process the purchaseorder. It reads the purchaseorder and splits it into smaller purchaseorders, one each for activities, transportation and lodging. It then sends each of these smaller purchaseorders to the webservicebroker that in turn sends them to external suppliers.

### OrderFillerQueue

### POEndpointBean

### Price

### ProcessManagerBean

This session bean provides operations to retrieve and update the overall status of a purchase order and the status of the individual supplier orders.

### PurchaseOrderBean

This entity bean persists the details of a purchase order (user who placed the order, date of the order, total price, head count, departure and arrival dates, departure city, etc.). The status of the order as it's processed is not stored within this entity bean; it's ManagerBean that keeps the order status.

### SendRequest

sendRequest - This interface can be used to contact external suppliers and place orders with them.

### TransportationBean

This entity bean persists the details of a flight reservation (e.g., origin, destination, carrier, flight id, departure time, arrival time, travel class, fare)

### WebServiceBroker

WebServiceBroker - This is a web service that is used by the external suppliers to submit invoices back to the OPC.

### WebServiceBrokerQueue

### WorkflowManagerBean

invoice message. This is a message that came from one of the external suppliers in response for a booking order. When an invoice message is received, this component basically interacts with ProcessManagerBean to update the status of the corresponding order. If the message confirms the last invoice that is part of a travel package order, WorkFlowManagerBean sends a JMS message to CrmBean to notify the customer via email.

purchase order message. When processing such messages, this component interacts with ProcessManagerBean to insert the order in the database in the Pending state. Then it interacts synchronously with the Bank external service provider to validate and charge the customer's credit card. If the credit card is OK, it sends a message to the OrderFillerQueue JMS queue to be processed by OrderFillerBean. Finally, it sends another message to CRMQueue, which will be processed by CrmBean (send email to the customer informing the order is being processed).

This message-driven bean is activated when there is a message in the WorkFlowMgrQueue. It processes two kinds of messages:

WofkFlowManagerBean is one of the two components in the Adventure Builder system that act as service user of external services (the other is BrokerRequestorBean).

WorkFlowManagerBean also sets a timer with the EJB container so that it is activated periodically to check the status of all pending orders.

### WorkflowMgrQueue

### arrivalCity

### arrivaltime

### carrier

### creditCardNumber

### dateTime

### departurCity8

### departurTime

### departureTime

### destination

### expirationDate

### flight\_id

### headCount

### headCount

### headCount

### hotel\_id

### id

### id

### id

### location

### numberOfNight

### numberOfRooms

### orderDateTime

### orderOverallStatus

### order\_id

### ordersStatus

### origin

### ratePernight

### startDate

### totalPrice

### travelClass

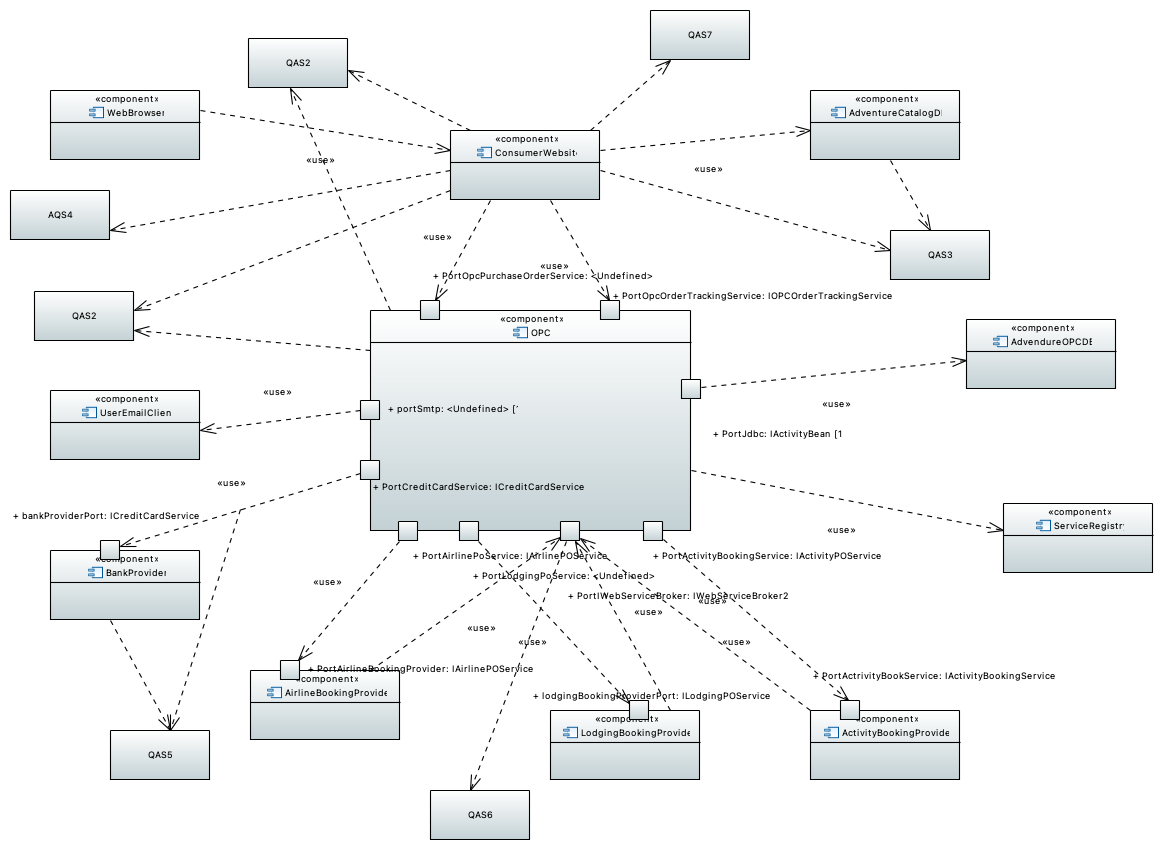
### type

### user\_id

### validationCode

# D-C&C

## !Top Level SOA View



### AQS4

The Consumer Web site sent a purchase order request to the order processing center (OPC). The OPC processed that request but didnât reply to Consumer Web site within five seconds, so the Consumer Web site resends the request to the OPC. The OPC receives the duplicate request, but the consumer is not double-charged, data remains in a consistent state, and the Consumer Web site is notified that the original request was successful in one hundred percent of the times.

### ActivityBookingProvider

This component represents an external application hosted by an activity provider partner company. The application provides a SOAP web service to book adventure activities, such as hot air ballooning, surf classes, and mountain climbing expeditions.

### AdvendureOPCDB

This relational database stores purchase orders, invoices coming from the external suppliers and related information. The database server is MySQL configured to use the InnoDB engine. Variability guide To configure data storage such that it supports different database vendors, we don't want to use any SQL statements in our queries that use vendor specific syntax and can potentially hinder migrating databases. We also use the DAO pattern to abstract away clients from accessing data directly.

### AdventureCatalogDB

To configure data storage such that it supports different database vendors, we don't want to use any SQL statements in our queries that use vendor specific syntax and can potentially hinder migrating databases. We also use the DAO pattern to abstract away clients from accessing data directly.

This is a relational database that stores the adventure builder catalog containing various adventure packages. It also stores information about users for user authentication and authorization. The database server is MySQL Cluster 7.0 configured to use the InnoDB engine.

Configurability of Data Access

### AirlineBookingProvider

This component represents an external application hosted by an airline partner company. The application provides a SOAP web service to book air travel.

### BankProvider

This component represents an external application hosted by a bank or credit card administrator. The application provides a SOAP web service to verify customers' credit card information.

### ConsumerWebsite

The ConsumerWebsite is a multi-tier application implemented using Java EE technology. It's the client facing part of the Adventure Builder system. It is implemented using GWT code, a number of JSP and html pages, and standard components of the WAF framework. Its primary responsibility is to process the http requests coming from customers browsing the catalog or placing/tracking orders. Requests to place or track orders are relayed to the OPC application via SOAP

### LodgingBookingProvider

This component represents an external application hosted by a lodging partner company. The application provides a SOAP web service to book hotel rooms.

### OPC

\* CreditCardService - used to validate the credit card transaction with a bank.

\* OpcOrderTrackingService - used to track the status of a purchase order.

\* Accept purchase order requests from the ConsumerWebsite for processing.

\* Provide a mechanism for the Consumer Website to query the current status of a purchase order.

#Rationale

- 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.

OPC provides the following interfaces, which are exposed as SOAP web services:

OPC is the order processing center application. It's a Java EE application that communicates with external components using SOAP web services.Internally, it consists of loosely coupled EJBs that communicate with each other using primarily asynchronous messaging. The internal architecture follows the message channel design pattern [Hohpe 2003]. See the OPC C&C View for a description of the internal components of OPC. The core functionality of the Adventure Builder is implemented in this module. Its major functions are:

\* LodgingPOService - used to send purchase orders to external lodging suppliers.

- For example, the Airline Provider component may be implemented using Java, PHP, .NET, IBM CICS or any other technology that supports SOAP web services.

\* Upon completion of processing a purchase order, send an email to the customer reporting its success or failure.

\* AirlinePOService - used to send purchase orders to external airline suppliers.

- 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.

\* OpcPurchaseOrderService - used to create a purchase order.

-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.

\* Fill a purchase order by communicating with external suppliers.

\* WebServiceBroker - used by external suppliers to submit invoices back to OPC.

\* ActivityPOService - used to send purchase orders to external activity suppliers.

OPC requires the following interfaces, which are provided by external partners as SOAP web services:

### PortActivityBookingService

### PortActrivityBookService

### PortAirlineBookingProvider

### PortAirlinePoService

### PortCreditCardService

### PortIWebServiceBroker

### PortJdbc

### PortLodgingPoService

### PortOpcOrderTrackingService

### PortOpcPurchaseOrderService

### QAS2

A user places an order for an adventure travel package to the Consumer Web site. The user is notified on screen that the order has been successfully submitted and is being processed in less than five seconds.

### QAS3

Up to 500 users click to see the catalog of adventure packages following a random distribution over 1 minute; the system is under normal operating conditions; the maximal latency to serve the first page of content is under 5 seconds; average latency for same is less than 2 seconds.

### QAS5

Credit approval and payment processing are requested for a new order. In one hundred percent of the cases, the transaction is completed securely and cannot be repudiated by either party. The business goals are to provide customers and business partners confidence in security and to meet contractual, legal, and regulatory obligations for secure credit transactions.

### QAS6

The OPC experiences a flood of calls through the Web Service Broker endpoint that do not correspond to any current orders. In one hundred percent of the times, the system detects the abnormal level of activity, notifies the system administrator, and continues to service requests in a degraded mode.

### QAS7

The Consumer Web site is available to the user 24x7. If an instance of OPC application fails, the fault is detected and the system administrator is notified in 30 seconds; the system continues taking order requests; another OPC instance is created; and data remains in consistent state.

### ServiceRegistry

Data repository that works as a basic registry of the external services used by OPC. More specifically, it has name, location and metadata about all the SOAP web services offered by the banks, airline, lodging, and activity external partners. TBD: use a relational database or XML based files.

### UserEmailClient

Variability Guide

All email communication parameters (hostname, ports and protocol) should be configurable outside the code via a configuration file. The parameters are read at load time.

This is the email inbox of the end user (vacationer) who placed an adventure travel purchase request. OPC sends email notifications to the user via SMTP to inform of the status of the orders.

### WebBrowser

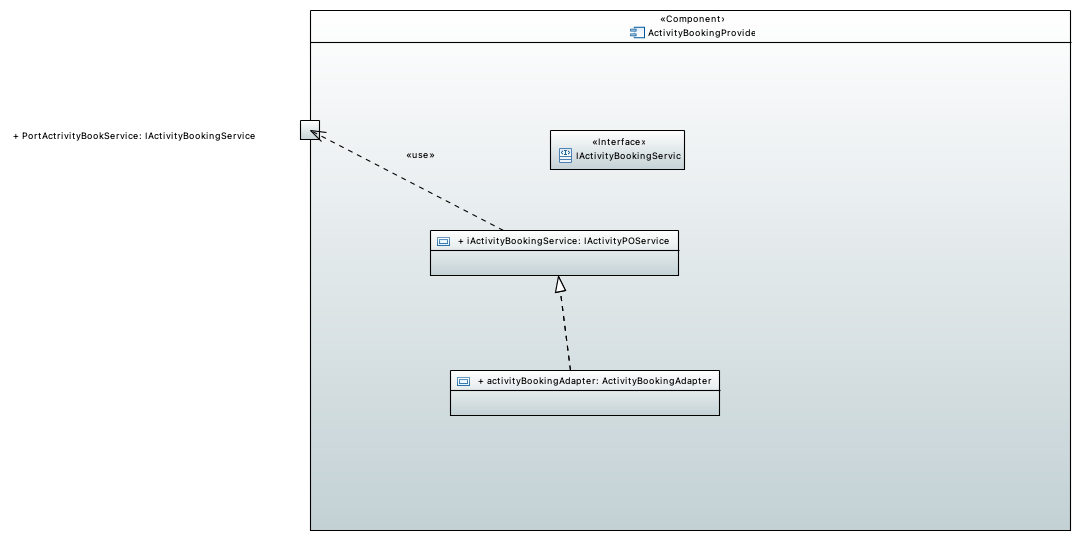
This component represents the user interface of the application running on a web browser. Consumer Website is a Web 2.0 application implemented using GWT. Therefore, in addition to HTML, the web browser runs JavaScript code that uses Ajax to communicate with the server. Using the web browser, a customer of Adventure Builder opens the web site, browses through the existing catalog of adventure packages, places orders, and tracks the status of existing orders.

### bankProviderPort

### lodgingBookingProviderPort

### portSmtp

## !Activity Booking Provider Component Detail



### IActivityBookingService

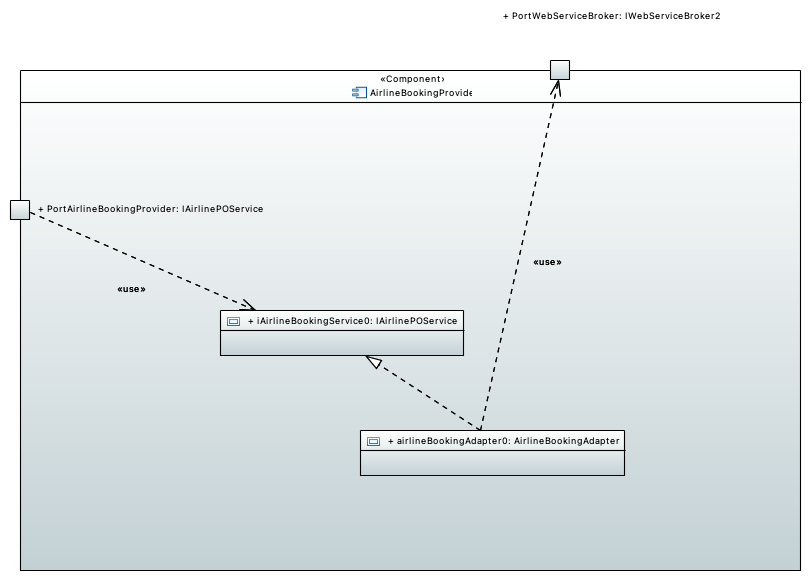
### PortActrivityBookService

### activityBookingAdapter

### iActivityBookingService

## !Activity Booking Provider Component Detail

## !Airline Booking Provider component detail



### PortAirlineBookingProvider

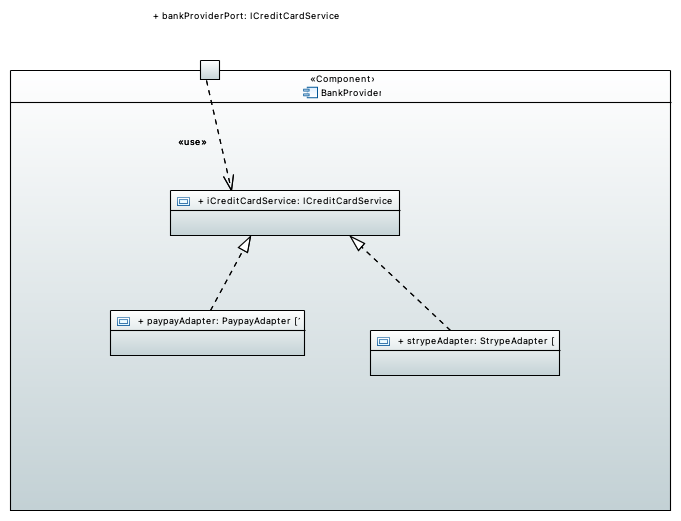
### PortWebServiceBroker

### airlineBookingAdapter0

### iAirlineBookingService0

## !Airline Booking Provider component detail

## !Bank Provider Component detail



### bankProviderPort

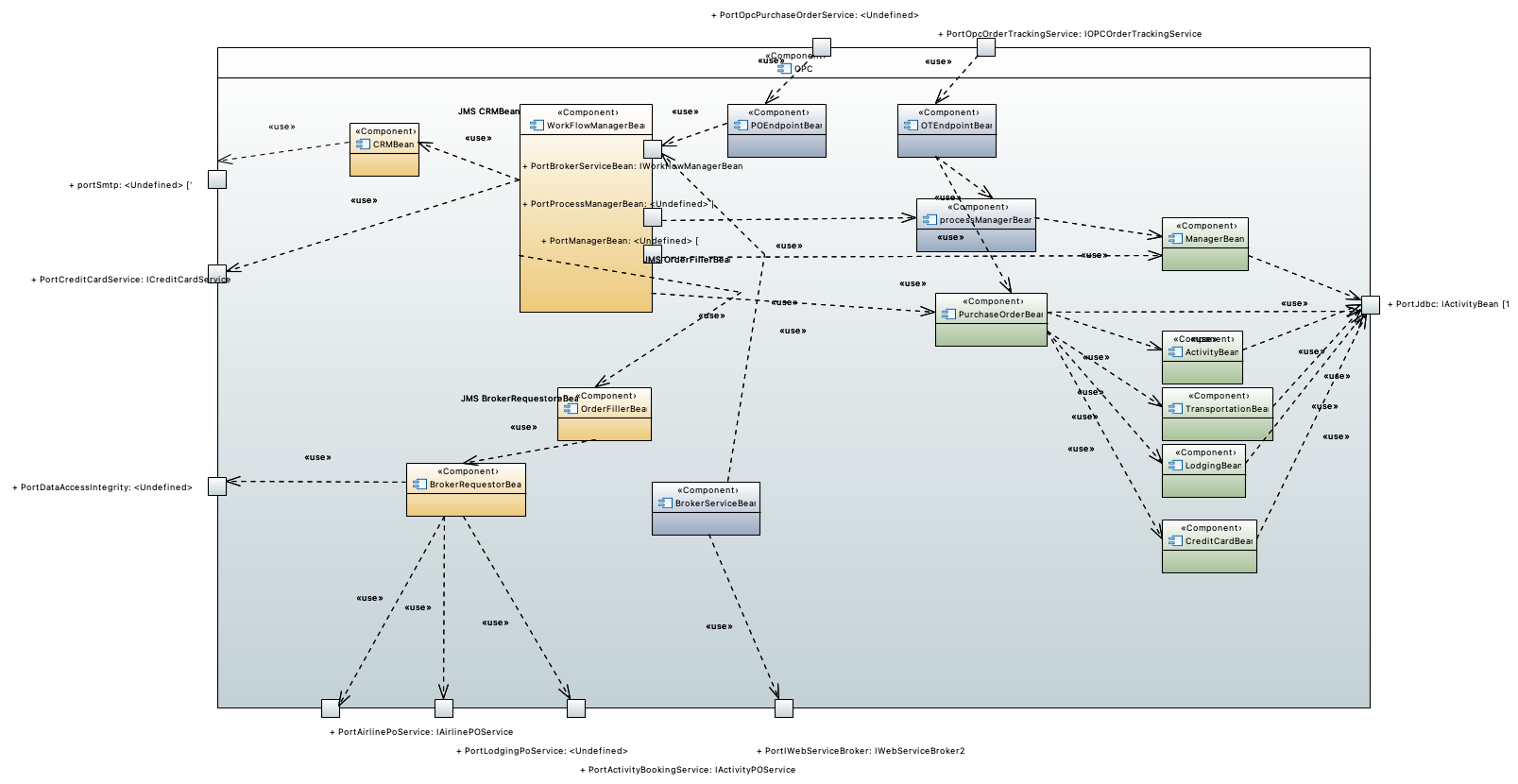
### iCreditCardService

### paypayAdapter

### strypeAdapter

## !Bank Provider Component detail

## !OPC C&C View



### ActivityBean

This entity bean persists the details of an activity reservation (activity id, location, price, date/time, head count).

### BrokerRequestorBean

This message driven bean makes requests to external suppliers. It requires the following interfaces: - AirlinePOService - This is used to send purchaseorders to external airline suppliers. - ActivityPOService - This is used to send purchaseorders to external activity suppliers. - LodgingPOService - This is used to send purchaseorders to external lodging suppliers.

### BrokerServiceBean

This component is the OPC's window to external suppliers for activities, lodging and transportation. It provides the following interfaces: - WebServiceBroker - This is a web service that is used by the external suppliers to submit invoices back to the OPC. - getInvoice - This interface can provide the invoices that are sent by external suppliers. - sendRequest - This interface can be used to contact external suppliers and place orders with them.

### CRMBean

This component is used for customer relationship management. For this application it is only used to communicate with the user. It reads messages from a queue, creates the corresponding email messages according to templates and I18N requirements, and sent them to users.

### CreditCardBean

This entity bean persists the credit card information (credit card number, type, expiration date, etc.) of a user.

### LodgingBean

This entity bean persists the details of a hotel reservation (hotel id, start date, number of nights, rate per night, number of rooms).

### ManagerBean

This entity bean is used by WorkFlowManagerBean to persist the status of the purchase order containing order id, order overall status, and the status of each separate supplier order.

### OTEndpointBean

This stateless session bean implements the SOAP web services interface called OpcOrderTrackingService. Requests for information about an order are handled by interacting with ProcessManagerBean. Order information is retrieved using the PurchaseOrderBean entity bean.

### OrderFillerBean

his component is used to process the purchaseorder. It reads the purchaseorder and splits it into smaller purchaseorders, one each for activities, transportation and lodging. It then sends each of these smaller purchaseorders to the webservicebroker that in turn sends them to external suppliers. - It provides an interface called sendPO that is used by the workflowmanager to send purchaseorders. - It requires an interface sendRequest that it uses to send purchaseorders to each individual supplier.

### POEndpointBean

This stateless session bean implements the SOAP web services interface called OpcPurchaseOrderService. When a purchase order request arrives, it simply validates the order and, if OK, sends the order to the WorkFlowMgrQueue using JMS.

### PortActivityBookingService

### PortAirlinePoService

### PortBrokerServiceBean

### PortCreditCardService

### PortDataAccessIntegrity

### PortIWebServiceBroker

### PortJdbc

### PortLodgingPoService

### PortManagerBean

### PortOpcOrderTrackingService

### PortOpcPurchaseOrderService

### PortProcessManagerBean

### PurchaseOrderBean

This entity bean persists the details of a purchase order (user who placed the order, date of the order, total price, head count, departure and arrival dates, departure city, etc.). The status of the order as it's processed is not stored within this entity bean; it's ManagerBean that keeps the order status.

### TransportationBean

This entity bean persists the details of a flight reservation (e.g., origin, destination, carrier, flight id, departure time, arrival time, travel class, fare)

### WorkFlowManagerBean

This message-driven bean is activated when there is a message in the WorkFlowMgrQueue. It processes two kinds of messages: - purchase order message. When processing such messages, this component interacts with ProcessManagerBean to insert the order in the database in the Pending state. Then it interacts synchronously with the Bank external service provider to validate and charge the customer's credit card. If the credit card is OK, it sends a message to the OrderFillerQueue JMS queue to be processed by OrderFillerBean. Finally, it sends another message to CRMQueue, which will be processed by CrmBean (send email to the customer informing the order is being processed). - invoice message. This is a message that came from one of the external suppliers in response for a booking order. When an invoice message is received, this component basically interacts with ProcessManagerBean to update the status of the corresponding order. If the message confirms the last invoice that is part of a travel package order, WorkFlowManagerBean sends a JMS message to CrmBean to notify the customer via email. WorkFlowManagerBean also sets a timer with the EJB container so that it is activated periodically to check the status of all pending orders. WofkFlowManagerBean is one of the two components in the Adventure Builder system that act as service user of external services (the other is BrokerRequestorBean). The life cycle of a purchase order can be summarized by the following UML state machine diagram. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=146280255

### portSmtp

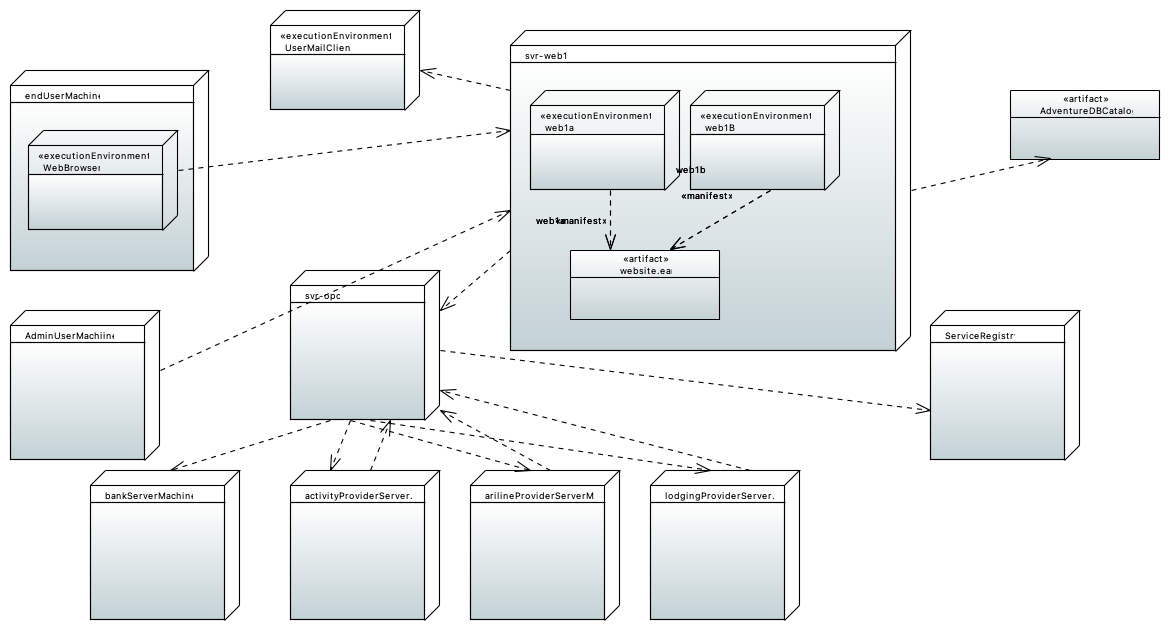
### processManagerBean

This session bean provides operations to retrieve and update the overall status of a purchase order and the status of the individual supplier orders.

## !OPC C&C View

# E-Allocation

## !singleDeployement



### AdminUserMachiine

### AdventureDBCatalog

This is a relational database that stores the adventure builder catalog containing various adventure packages. It also stores information about users for user authentication and authorization. The database server is MySQL Cluster 7.0 configured to use the InnoDB engine.

### ServiceRegistry

Data repository that works as a basic registry of the external services used by OPC. More specifically, it has name, location and metadata about all the SOAP web services offered by the banks, airline, lodging, and activity external partners. TBD: use a relational database or XML based files.

### UserMailClient

This is the email inbox of the end user (vacationer) who placed an adventure travel purchase request. OPC sends email notifications to the user via SMTP to inform of the status of the orders. Email communication All email communication parameters (hostname, ports and protocol) should be configurable outside the code via a configuration file. The parameters are read at load time.

### WebBrowser

This component represents the user interface of the application running on a web browser. Consumer Website is a Web 2.0 application implemented using GWT. Therefore, in addition to HTML, the web browser runs JavaScript code that uses Ajax to communicate with the server. Using the web browser, a customer of Adventure Builder opens the web site, browses through the existing catalog of adventure packages, places orders, and tracks the status of existing orders.

### activityProviderServerMachine

TODO. This node is external to the system. It represents the server hosted by an activities supplier that provides services to place an activities purchase order.

### arilineProviderServerMachine

TODO. This node is external to the system. It represents the server hosted by an airlines supplier that provides services to place a transportation purchase order.

### bankServerMachine

TODO. This node is external to the system. It represents the server hosted by a bank that provides services to verify a user's account.

### endUserMachine

This node is used to represent a user who can access the Adventure Builder application using a PC. It is most likely that he is using a web browser to place or track a purchaseorder.

### lodgingProviderServerMachine

TODO. This node is external to the system. It represents the server hosted by a lodging supplier that provides services to place a lodging purchase order.

### svr-opc

### svr-web1

### web1B

### web1a

### website.ear

website.ear This is an enterprise archive artifact that is deployed on an application server. It contains the consumer website application.

# F-Comportement