Designing SOA Case Study^(*)

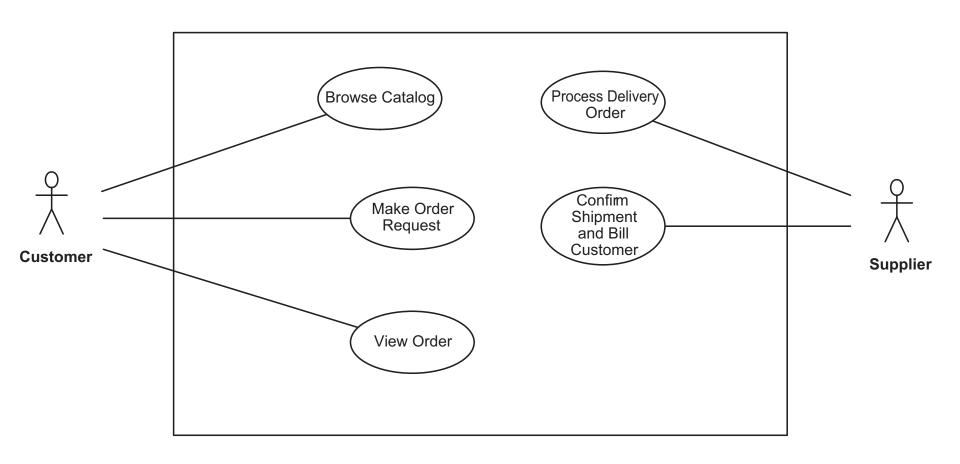
(*) Hassan Gomaa, Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures, Cambridge, 2011.

Web-based Online Shopping System

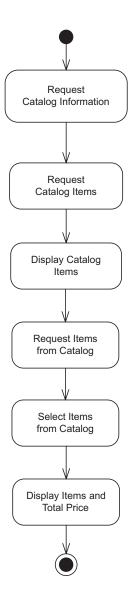
- In the Web-based Online Shopping System, customers can request to purchase one or more items from the supplier
- The customer provides personal details, such as address and credit card information
- This information is stored in a customer account
- If the credit card is valid, then a delivery order is created and sent to the supplier
- The supplier checks the available inventory, confirms the order, and enters a planned shipping date
- When the order is shipped, the customer is notified, and the customer's credit card account is charged

Use Case Modeling

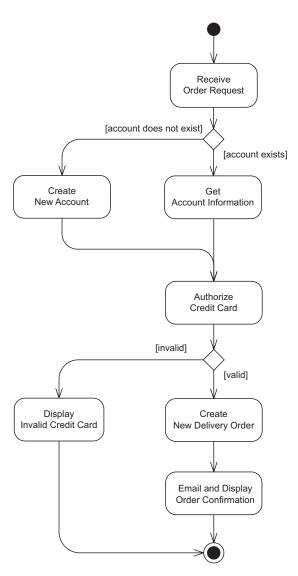
Use Case Diagram



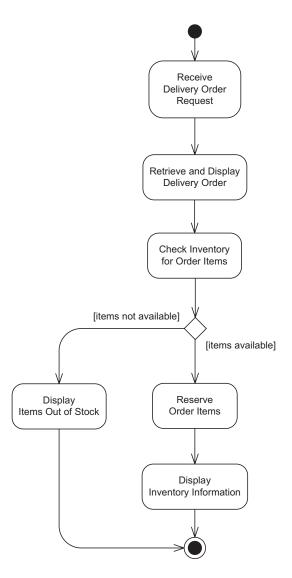
Activity diagram for Browse Catalog UC



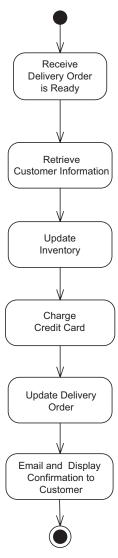
Activity diagram for Make Order Request UC



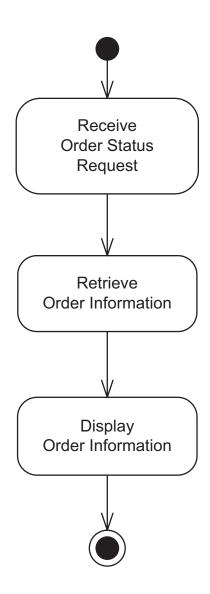
Activity diagram for *Process Delivery Order* UC



Activity diagram for Confirm Shipment and Bill Customer UC

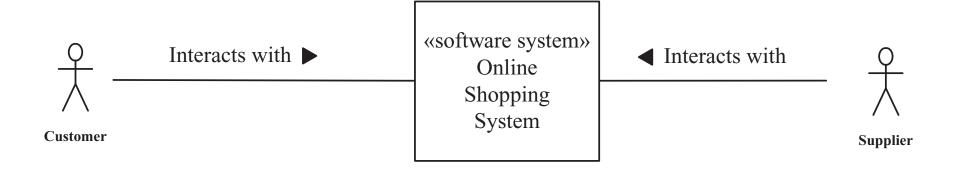


Activity diagram for View Order UC

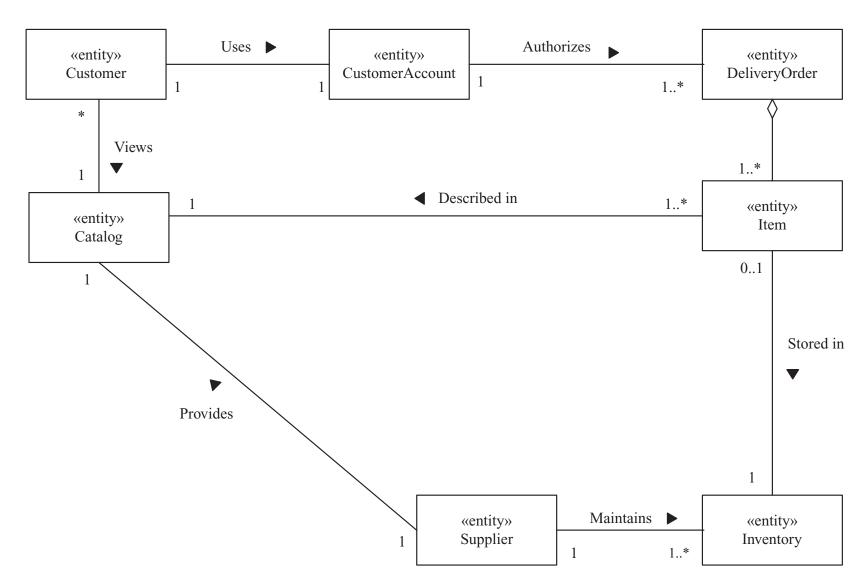


Static Modeling

Software System Context Class Diagram



Entity Class Diagram



Entity Classes

«entity» DeliveryOrder

orderId: Integer

order Status: Order status Type

accountId: Integer amountDue: Real authorizationId: Integer supplierId: Integer creationDate: Date plannedShipDate: Date actualShipDate: Date paymentDate: Date

«entity» Inventory

item ID: Integer

itemDescription : String quantity : Integer

price : Real
reorderTime : Date

«entity» Customer

customerId : Integer customerName : String

address: String

telephoneNumber: String

faxNumber : String emailId : EmailType

«entity» Item

itemId : Integer
unitCost : Real
quantity : Integer

«entity» Catalog

itemId: Integer

item Description: String

unitCost : Real supplierId : Integer itemDetails : linkType

«entity» Supplier

supplierId : Integer supplierName: String

address: String

telephoneNumber : String faxNumber : String email : EmailType

«entity» CustomerAccount

accountId : Integer cardId : String cardType : String expirationDate: Date

Class Structuring

- The entity classes determined in the previous section are integrated into a service-oriented architecture by means of service classes
- Catalog Service, Customer Account Service, Delivery Order Service, and Inventory Service are service classes that provide access to the entity classes

Service and Entity Classes

«service»
DeliveryOrderService

«entity»
DeliveryOrder

Item

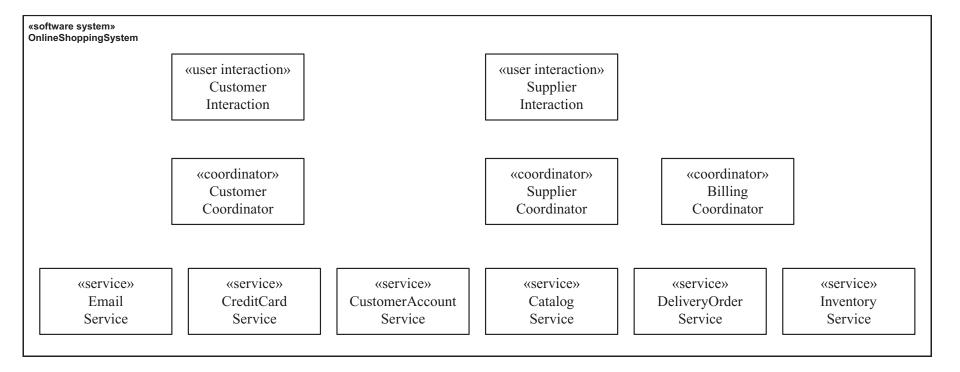
«service»
InventoryService

«entity»
Inventory

Other classes

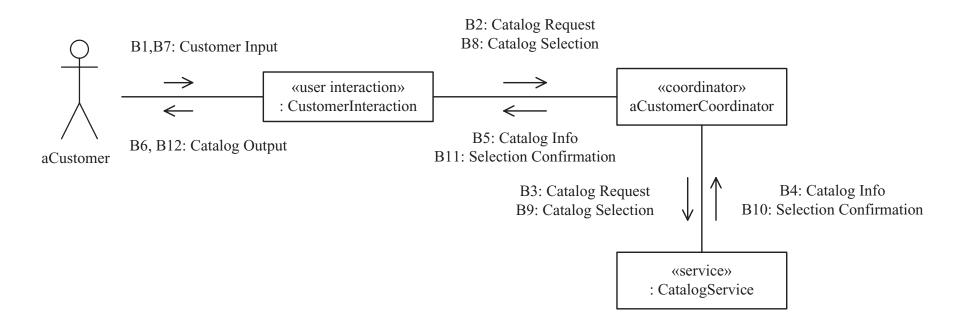
- There is also a service class, Credit Card Service, which deals with credit card authorization and charging
- Another service class is Email Service, which enables the Online Shopping System to send email messages to customers.
- User interaction classes are needed to interact with the external users – in particular, Customer Interaction and Supplier Interaction, which correspond to the actors in the use cases
- In addition, to coordinate and sequence the customer and supplier access to the online shopping services, two coordinator classes, Customer Coordinator and Supplier Coordinator, are provided,. A third autonomous coordinator, Billing Coordinator, is needed to deal with billing customers.

Class Structuring

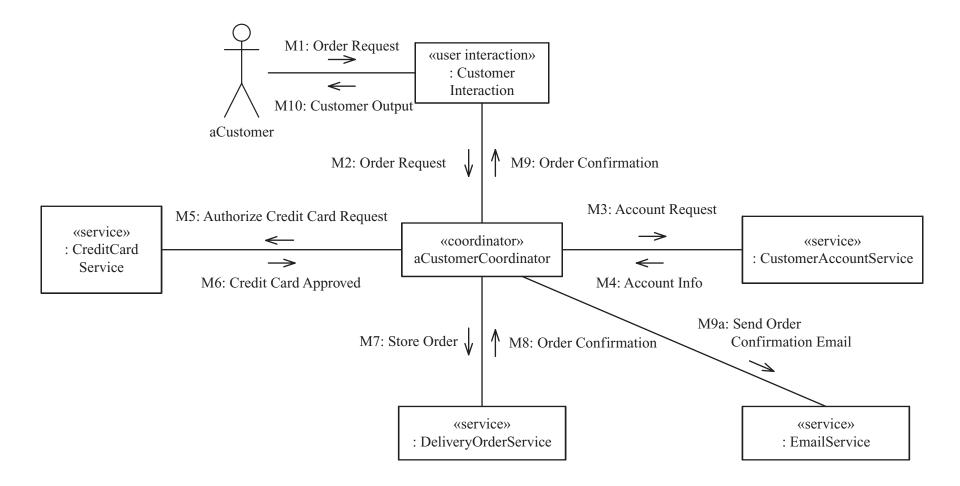


Dynamic Modeling

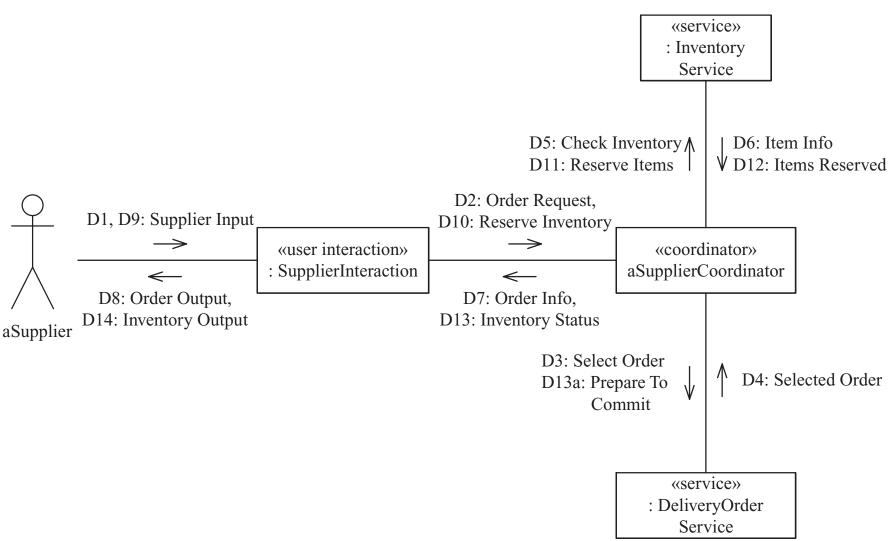
Communication diagram for the *Browse Catalog* UC



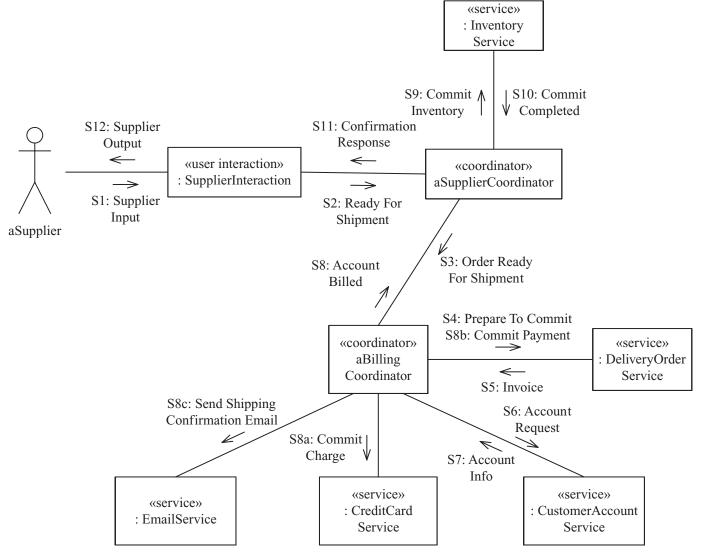
Communication diagram for the *Make Order Request* UC



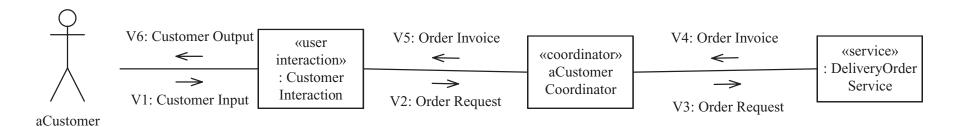
Communication diagram for the *Process Delivery Order* UC



Communication diagram for the *Confirm Shipment and Bill Customer* UC



Communication diagram for the *View Order* UC

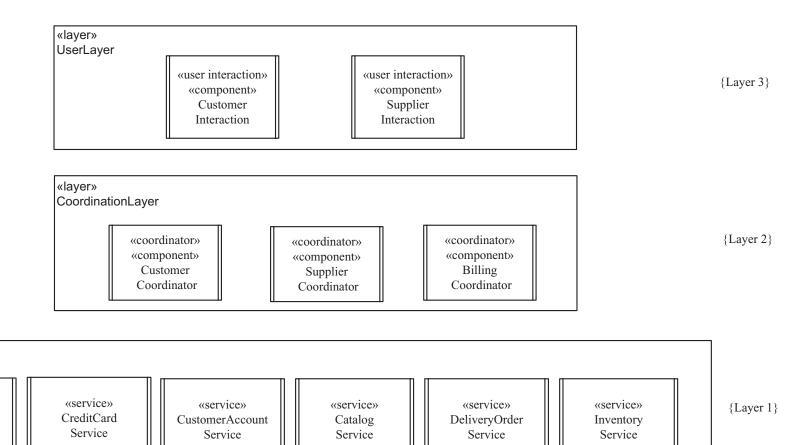


Design Modeling

Design Modeling Overview

- The Online Shopping System is designed as a layered architecture based on the Layers of Abstraction architecture pattern
- The software architecture consists of three layers a (data) service layer, a coordinator layer, and a user interaction layer
- Furthermore, because this system needs to be highly flexible and distributed, the decision is made to design a serviceoriented architecture, in which distributed components can discover services and communicate with them
- Each component is depicted with the component stereotype (what kind of component it is, as specified by the component structuring criteria)
- The design of the component and service interfaces are determined by analysis of the communication diagrams for each use case

Layered Architecture



«layer» ServiceLayer

«service»

Email

Service

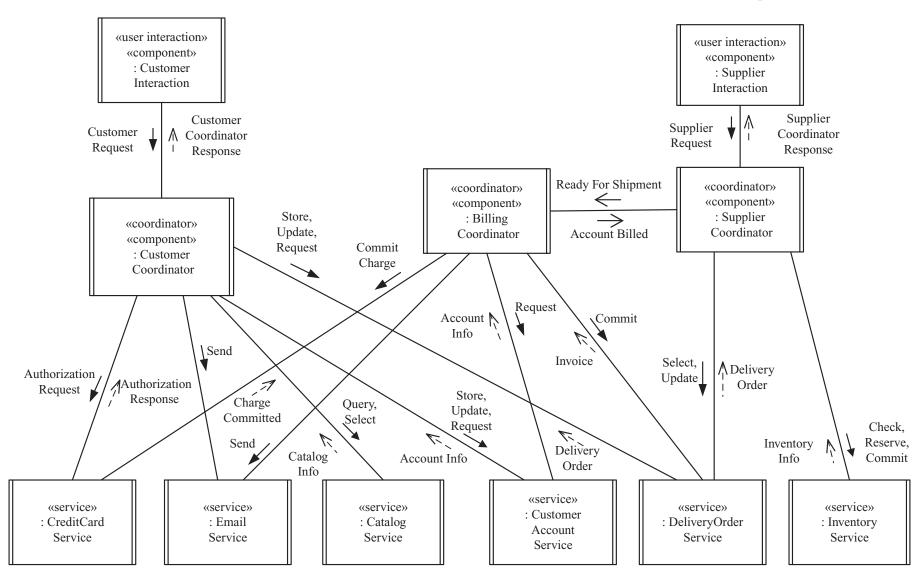
Architectural Communication Patterns

- Synchronous Message Communication with Reply
- Broker Handle
- Service Discovery
- Bidirectional Asynchronous Message
 Communication
- Two-Phase Commit

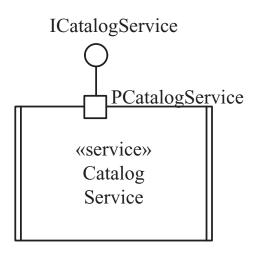
Concurrent Software Design

- To keep the design simple, the Synchronous Message Communication with Reply pattern has been widely used in this case study
- This approach has the disadvantage of suspending the client while it awaits a response from the service
- An alternative design to avoid suspending the client is to use the Asynchronous Message Communication with Callback pattern
- The Bidirectional Asynchronous Communication pattern is used for Supplier Coordinator and Billing Coordinator to communicate with each other in both directions

Concurrent Communication Diagram



Component Ports and Interfaces for Services



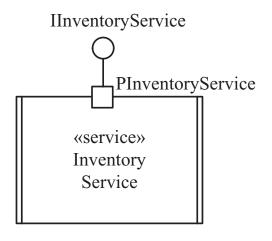
«service»

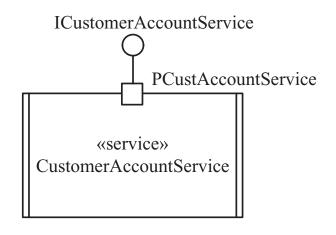
DeliveryOrder

Service

PDOService





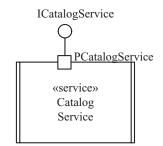


IDeliverOrderService

Service Interface for Catalog Service

«interface» ICatalogService

requestCatalog (in catalogType, out catalogInfo)
requestSelection (in itemId, out itemInfo)



«entity» CatalogInfo

catalog Id: Integer

catalog Description: String

supplierId : Integer

catalog Type: Catalog Type

«entity» ItemInfo

1..*

itemId : Integer

itemDescription: String

unitCost : Real supplierId : Integer itemDetails : URL «enumeration» CatalogType

Books

Computers

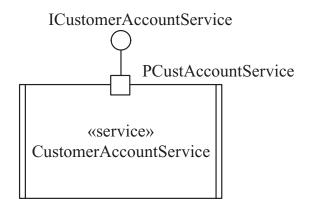
Home

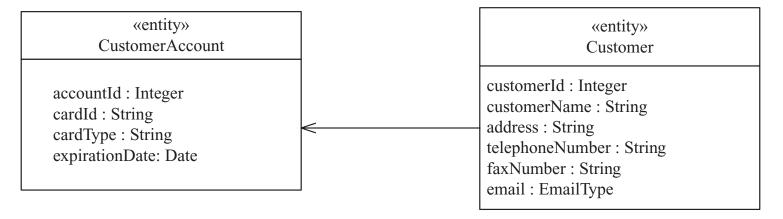
Toys

Service Interface for *Customer Account* Service

«interface» ICustomerAccountService

requestAccount (in accountId, out account)
createAccount (in cardId, in cardType, in expirationDate,
out accountId)
updateAccount (in accountId, in cardId, in cardType, in
expirationDate)

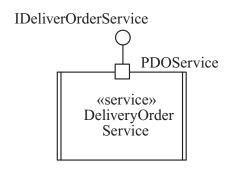


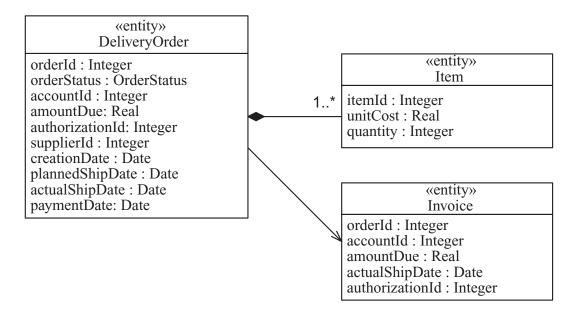


Service Interface for *Delivery Order* Service

«interface» IDeliveryOrderService

store (in order, out orderId)
select(in supplierId, out order)
update(in orderId, in order, out orderStatus)
orderShipped (in orderId, out orderStatus)
confirmPayment (in orderId, in amount, out orderStatus)
read (in orderId, out order)
requestInvoice (in orderId, out invoice)
prepareToCommitOrder (in orderId, out order)
commit (in orderId)
abort (in orderId)





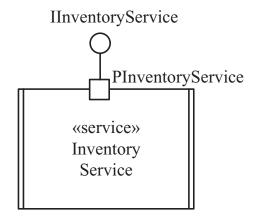
«enumeration» OrderStatus

NotYetShipped PreparedForShipment Shipped

Service Interface for *Inventory* Service

«interface» IInventoryService

checkInventory (in itemId, out inventoryStatus)
update (in itemId, in amount)
reserveInventory (in itemId, in amount)
commitInventory (in itemId, in amount)
abortInventory (in itemId, in amount)



«entity» Inventory

itemID: Integer

itemDescription: String

quantity: Integer

quantityReserved: Integer

price: Real

reorderTime: Date

«entity» InventoryStatus

itemID: Integer

currentAmount: Integer

quantityAfterShipped: Integer

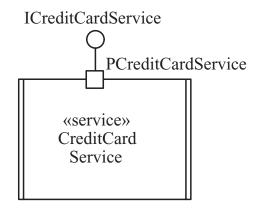
reorderTime: Date

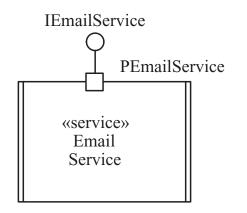
Service Interface for Credit Card and Email services

authorizeCharge (in creditcardId, in amount, out authorizationResponse) commitCharge (in creditcardId, in amount, out chargeResponse) abortCharge (in creditcardId, in amount, out chargeResponse)

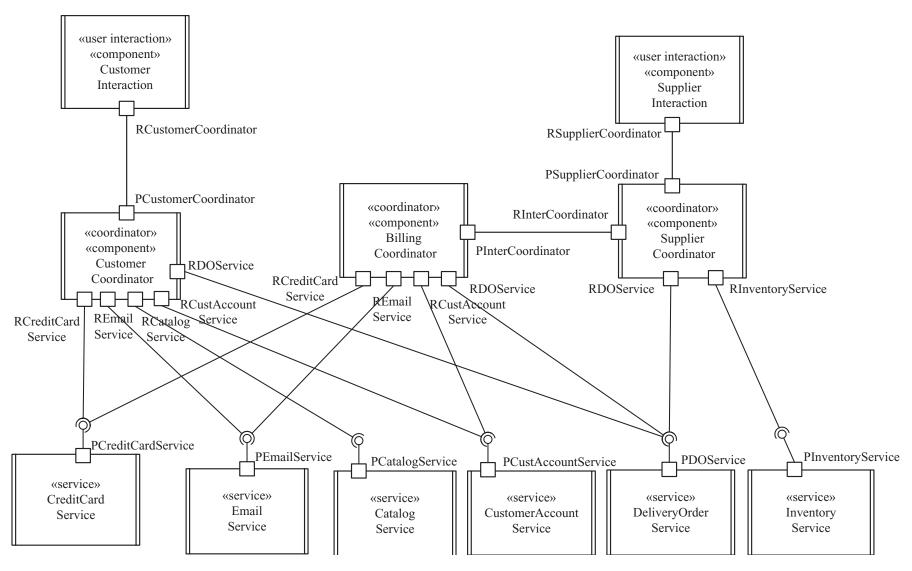
«interface» IEmailService

sendEmail (in emailId, in emailText)





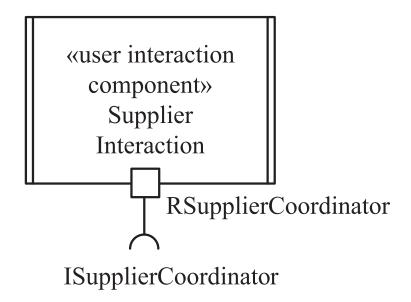
Service-Oriented Software Architecture



Component ports and interfaces for Customer Interaction and Supplier Interaction

«user interaction component»
CustomerInteraction
RCustomerCoordinator

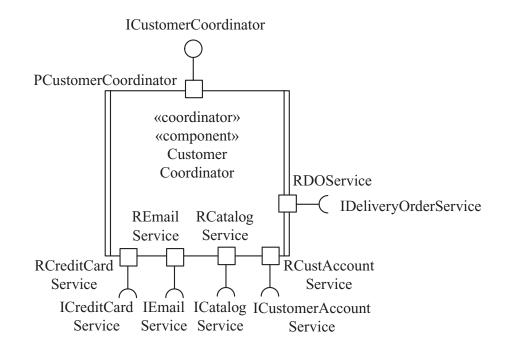
ICustomerCoordinator



Component ports and interfaces for Customer Coordinator

«interface» ICustomerCoordinator

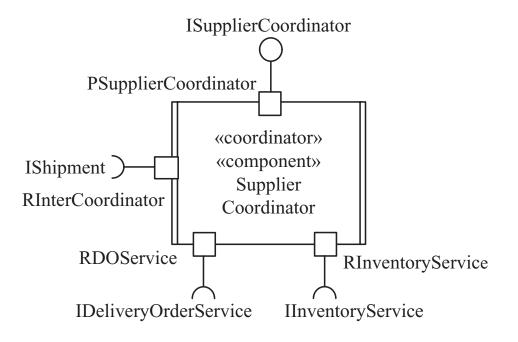
requestCatalog(in selectedCatalog, out catalogInfo)
requestSelectionFromCatalog(in itemId, out itemInfo)
requestPurchase(in purchaseInfo, out orderInfo)
requestAccount(in accountId, out accountInfo)
requestOrderStatus(in orderId, out orderInfo)



Component ports and interfaces for Supplier Coordinator

«interface» ISupplierCoordinator

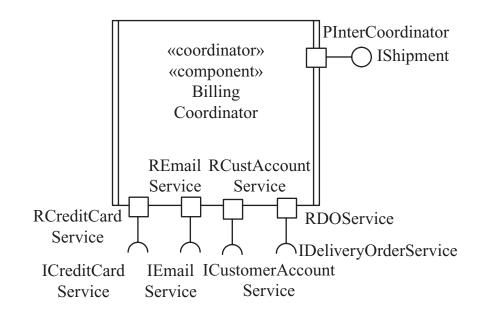
confirmShipment (in orderId)
requestOrder (in supplierId, out orderId)
checkInventory (in orderId, out inventoryInfo)
reserveInventory (in orderId, out inventoryInfo)



Component ports and interfaces for *Billing* Coordinator

«interface» IShipment

orderReadyForShipment (in orderId, in orderStatus)



Service Reuse

- With the SOA paradigm, once the services have been designed and their interfaces specified, the service interface information can be registered with a service broker
- Services can be composed into new applications
- This case study has described an Online Shopping System
- However, other electronic commerce systems could be designed that would reuse the services provided by the Online Shopping System, such as Catalog Service, Delivery Order Service, and Inventory Service