

Team Alpha Final Project

**E-commerce system becomes stable
Shopping Model**

CSCE 866
Spring 2002

April 30, 2002

Yun Feng
Kurt Weiss

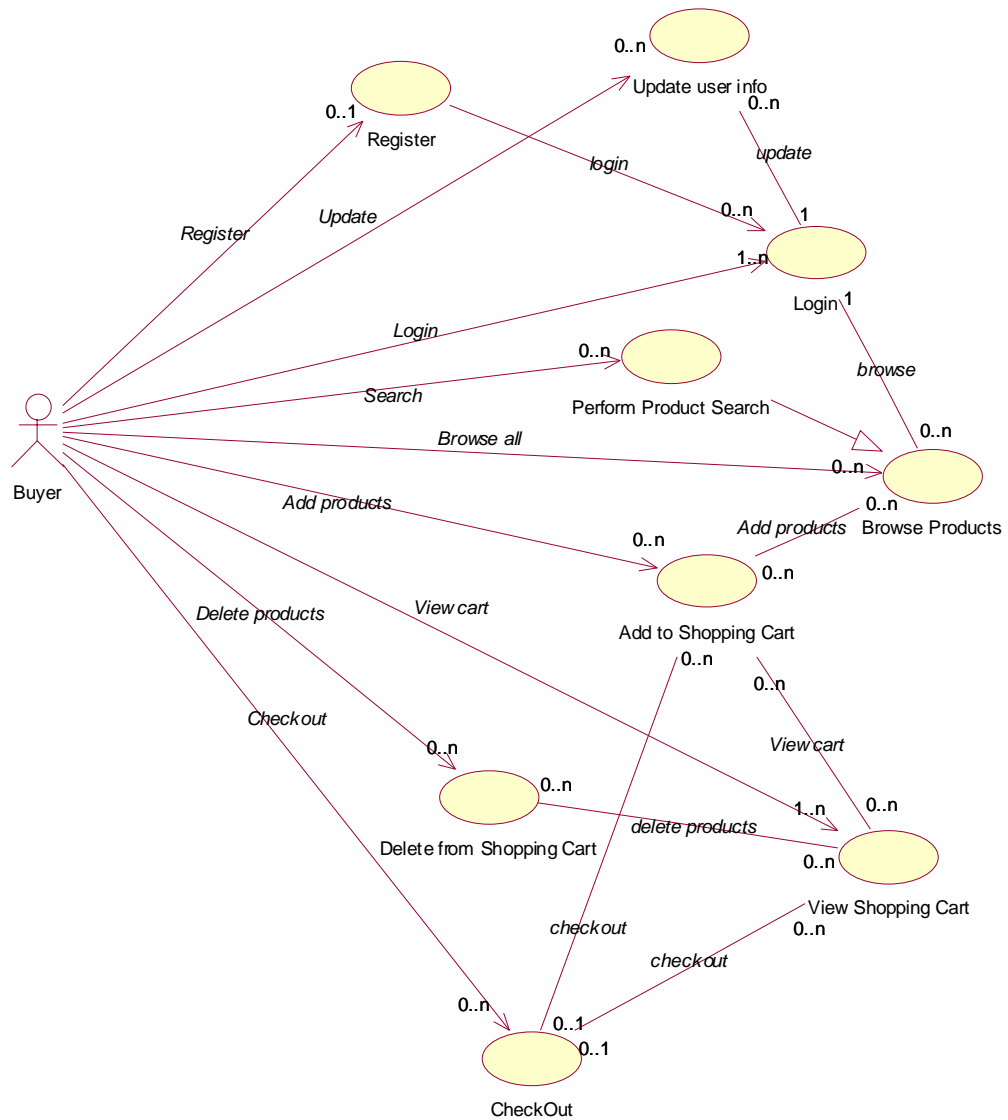
Table of Contents

Use Cases	3
Diagrams	3
Product Selection	3
Buying.....	4
Use Cases Description.....	5
CRC Cards	14
Class Diagram (Traditional Model)	18
Description:	18
Class Diagram (Stability Model).....	19
Description:	20
Behavior Model – State Transition Diagram (STD).....	21
Product	21
Description:.....	21
Shopping.....	22
Description:.....	22
Shopping Cart.....	23
Description:.....	23
Behavior Model – Activity Diagram.....	24
Activity diagram – buyer (customer) browsing activities	24
Description:.....	24
Activity diagram – buyer (customer) checkout activities	25
Description:.....	25
Activity diagram – Buyer (customer) login and update activities	26
Description:.....	26
OO Heuristics	27
Rewrite of Problem Statement	28
Pattern Documentation.....	29
Meta Model Shopping (EBT).....	29
Meta Model Buying (EBT)	30
Meta Model Invoice (BO).....	31
Meta Model Product (BO)	32
Reference.....	32

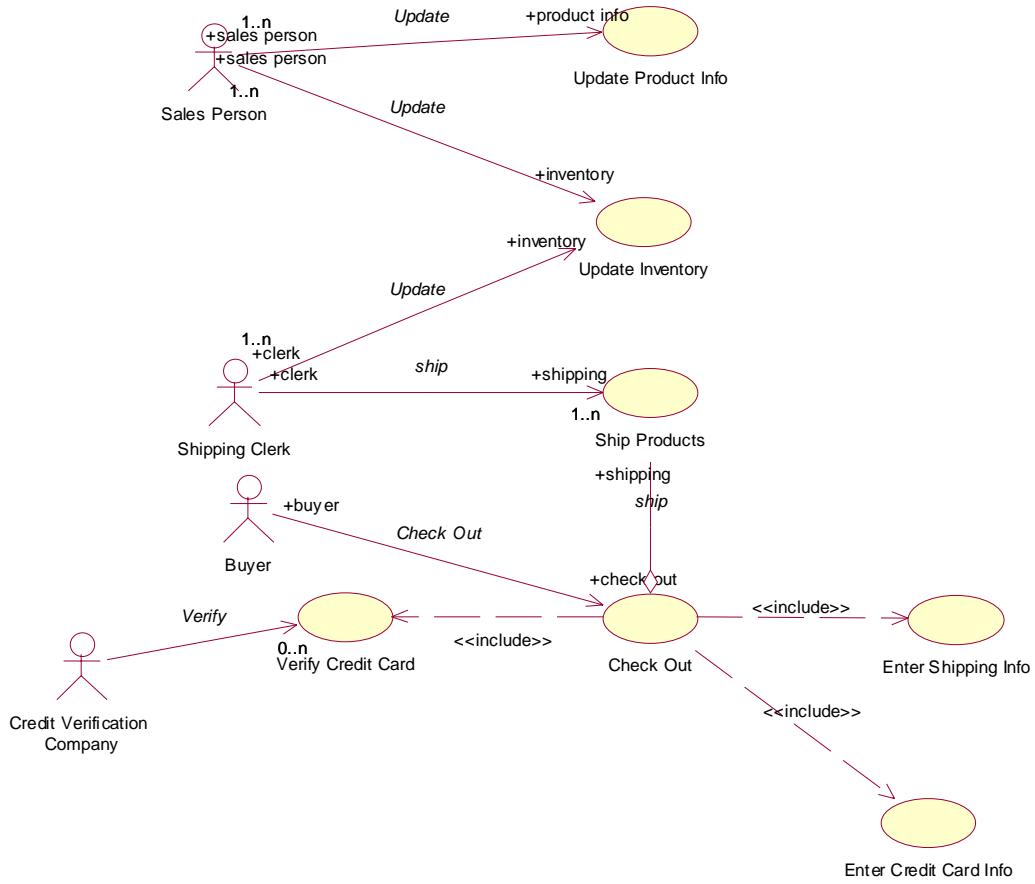
Use Cases

Diagrams

Product Selection



Buying



Use Cases Description

Use Case Id:	1
Use Case Title:	Register
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Buying
Corresponding Attributes:	Buyer
Corresponding Interfaces:	Buy
EBTs:	Buying
Attributes: Buyer Behaviors: Order(), Pay()	
Business Objects:	No
Industrial Objects:	No
Use Case Description:	<ol style="list-style-type: none"> 1. A new Buyer request to register; 2. The system display the register page and asks the new buyer to enter login name and password; 3. Buyer enters the required information; 4. The system precedes the register process.
Alternatives:	1.1 if it's not a new buyer , register is not necessary.

Use Case Id:	2
Use Case Title:	Login
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Buying
Corresponding Attributes:	Buyer
Corresponding Interfaces:	Buy
EBTs:	Buying
Attributes: Buyer Behaviors: Order(), Pay()	
Business Objects:	No
Industrial Objects:	No
Use Case Description:	<ol style="list-style-type: none"> 1. Buyer enter his/her user name and password; 2. the system verify the buyer name & password; 3. If the name and password match, login.
Alternatives:	3.1. if the name and password do not match, error message is shown.

Use Case Id:	3
Use Case Title:	Update user info
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Buying
Corresponding Attributes:	Buyer
Corresponding Interfaces:	Buy
EBTs:	Buying
Attributes: Buyer Behaviors: Order(), Pay()	
Business Objects:	No
Industrial Objects:	No
Use Case Description:	<ol style="list-style-type: none"> 1. old Buyer login the system with username and password; 2. The Buyer request to update user info; 3. the system displays the current user info; 4. user updates the buyer info and save; 5. system will store the updated user info.
Alternatives:	4.1 buyer can cancel the update.

Use Case Id:	4
Use Case Title:	Browse Products
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Shopping, Product
Corresponding Attributes:	Seller, Buyer, Product, productName, productPrice, productNumber, brand, description
Corresponding Interfaces:	Shop
EBTs:	Shopping
Attributes: Seller, Buyer, Product Behaviors: examineProduct(), compareProduct(), selectProduct()	
Business Objects:	Product
Attributes: productName, productPrice, productNumber, brand, description Behaviors: displayProduct(), updateProduct(), createProduct(), destroyProduct(), getProduct(), searchProduct()	
Industrial Objects:	No
Use Case Description:	<ol style="list-style-type: none"> 1. Buyer requests to view the products in a product category; 2. The system returns the shopping information for all the available products.
Alternatives:	2.1 if no available products, show "no record" info.

Use Case Id:	5
Use Case Title:	Perform Product Search
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Shopping, Product
Corresponding Attributes:	Seller, Buyer, Product, productName, productPrice, productNumber, brand, description
Corresponding Interfaces:	Shop
EBTs:	Shopping
Attributes: Seller, Buyer, Product	
Behaviors: examineProduct(), compareProduct(), selectProduct()	
Business Objects:	Product
Attributes: productName, productPrice, productNumber, brand, description	
Behaviors: displayProduct(), updateProduct(), createProduct(), destroyProduct(), getProduct(), searchProduct()	
Industrial Objects:	No
Use Case Description:	<ol style="list-style-type: none"> 1. Buyer inputs the product name he/she wants to search for shopping; 2. The system returns all available products that fit the search name.
Alternatives:	2.1 if no matched products, show "can not find"

Use Case Id:	6
Use Case Title:	Add to Shopping Cart
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Shopping, Product, Shopping Cart
Corresponding Attributes:	Seller, Buyer, Product, productName, productPrice, productNumber, brand, description, shoppingCartID, quantity, totalAmount
Corresponding Interfaces:	Shop, store
EBTs:	Shopping
Attributes: Seller, Buyer, Product	
Behaviors: examineProduct(), compareProduct(), selectProduct()	
Business Objects:	Product
Attributes: productName, productPrice, productNumber, brand, description	
Behaviors: displayProduct(), updateProduct(), createProduct(), destroyProduct(), getProduct(), searchProduct()	

Industrial Objects:	Shopping Cart
Attributes: ShoppingCartID, quantity, totalAmount Behaviors: displayCart(), addProductOrder(), removeProductOrder(), getCart(), checkout(), createCart(), destroyCart()	
Use Case Description:	<ol style="list-style-type: none"> 1. When the Buyer finds the products he/she wants, he/she adds them to the shopping cart; 2. The system will store and keep the information of the products that have been added into shopping cart.
Alternatives:	No.

Use Case Id:	7
Use Case Title:	Delete Products from Shopping Cart
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Shopping, Product, Shopping Cart
Corresponding Attributes:	Seller, Buyer, Product, productName, productPrice, productNumber, brand, description, shoppingCartID, quantity, totalAmount
Corresponding Interfaces:	Shop, store
EBTs:	Shopping
Attributes: Seller, Buyer, Product Behaviors: examineProduct(), compareProduct(), selectProduct()	
Business Objects:	Product
Attributes: productName, productPrice, productNumber, brand, description Behaviors: displayProduct(), updateProduct(), createProduct(), destroyProduct(), getProduct(), searchProduct()	
Industrial Objects:	Shopping Cart
Attributes: ShoppingCartID, quantity, totalAmount Behaviors: displayCart(), addProductOrder(), removeProductOrder(), getCart(), checkout(), createCart(), destroyCart()	
Use Case Description:	<ol style="list-style-type: none"> 1. After the buyer has added some products into shopping cart, if he/she later doesn't want it (them) any more, he/she selects to delete it(them) from shopping cart; 2. The system will delete the product(s) and its relating information from the shopping cart.
Alternatives:	No.

Use Case Id:	8
Use Case Title:	View Shopping Cart
Actors & Corresponding Roles:	Buyer
Corresponding Classes:	Shopping, Product, Shopping Cart
Corresponding Attributes:	Seller, Buyer, Product, productName, productPrice, productNumber, brand, description, shoppingCartID, quantity, totalAmount
Corresponding Interfaces:	Shop, store
EBTs:	Shopping
Attributes: Seller, Buyer, Product	
Behaviors: examineProduct(), compareProduct(), selectProduct()	
Business Objects:	Product
Attributes: productName, productPrice, productNumber, brand, description	
Behaviors: displayProduct(), updateProduct(), createProduct(), destroyProduct(), getProduct(), searchProduct()	
Industrial Objects:	Shopping Cart
Attributes: ShoppingCartID, quantity, totalAmount	
Behaviors: displayCart(), addProductOrder(), removeProductOrder(), getCart(), checkout(), createCart(), destroyCart()	
Use Case Description:	<ol style="list-style-type: none"> 1. Once the Buyer has add some products into shopping cart, he/she can request to view the shopping cart; 2. The system show all the products and their information in the shopping cart and total price; 3. After review, the user can continue shopping.
Alternatives:	<ol style="list-style-type: none"> 3.1 the Buyer can select to checkout if he/she finish shopping; 3.2 The buyer can select to delete one or more products in the shopping cart.

Use Case Id:	9
Use Case Title:	Update Product Info
Actors & Corresponding Roles:	Sales Person, sales person
Corresponding Classes:	Selling, Product
Corresponding Attributes:	Seller ProductName ProductPrice Brand Description productNumber
Corresponding Interfaces:	Update
EBTs:	Shopping, Selling
Attributes: seller, buyer, product Operations: examineProduct, compareProduct, selectProduct, deliver, collectPayment	
Business Objects:	Product
Attributes: productName, productPrice, Brand, description, productNumber Operations: displayProduct, updateProduct, createProduct, destroyProduct, getProduct, searchProduct	
Industrial Objects:	None
Use Case Description:	1. buyer enters system with ID 2. buyer changes product information 3. buyer saves new product information
Alternatives:	3.1. buyer aborts changes

Use Case Id:	10
Use Case Title:	Update Inventory
Actors & Corresponding Roles:	Sales Person, sales person; Shipping Clerk, clerk
Corresponding Classes:	Inventory, Product
Corresponding Attributes:	InventoryID, dateReceived, dateShipped; ProductName, ProductPrice, Brand, Description, productNumber
Corresponding Interfaces:	Update
EBTs:	Shopping
Attributes: Seller, Buyer, Product Operations: examineProduct(), compareProduct(), selectProduct()	
Business Objects:	Product
Attributes: productName, productPrice, Brand, description, productNumber Operations: displayProduct, updateProduct, createProduct, destroyProduct, getProduct, searchProduct	

Industrial Objects:	Inventory
Attributes:	InventoryID, dateReceived, dateShipped
Operations:	updateInventory, createInventory, destroyInventory, checkInventory, getInventory
Use Case Description:	4. Buyer enter system with ID 5. buyer changes inventory record 6. buyer saves changes
Alternatives:	3.1. buyer aborts changes

Use Case Id:	11
Use Case Title:	Ship Products
Actors & Corresponding Roles:	Shipping Clerk, clerk
Corresponding Classes:	Shipping Info
Corresponding Attributes:	ShippingName, shippingAddress
Corresponding Interfaces:	ship
EBTs:	Selling
Attributes:	Seller
Operations:	deliver(), collectPayment()
Business Objects:	Delivery
Attributes:	customerInfo, deliveryType, deliveryDate
Operations:	processDelivery()
Industrial Objects:	No
Use Case Description:	1. buyer gets shipping information 2. buyer directs shipment to proceed
Alternatives:	none

Use Case Id:	12
Use Case Title:	Check Out
Actors & Corresponding Roles:	Buyer, buyer
Corresponding Classes:	Shopping Cart
Corresponding Attributes:	ShoppingCartID, quantity, totalAmount
Corresponding Interfaces:	Check Out
EBTs:	Buying
Attributes:	Buyer
Operations:	Order(), Pay()

Business Objects:	Order
Attributes:	productName, productQuantity, totalPrice, customerInfo, orderDate
Operations:	requestOrder(), processOrder()
Industrial Objects:	Order Form
Attributes:	orderNumber, date
Operations:	createOrderForm, destroyOrderForm, calculateTotal
Use Case Description:	<ol style="list-style-type: none"> 1. buyer completes orders 2. buyer selects checkout 3. buyer logs off
Alternatives:	3.1. buyer continues shopping

Use Case Id:	13
Use Case Title:	Enter Shipping Info
Actors & Corresponding Roles:	Buyer, buyer
Corresponding Classes:	Shipping Info
Corresponding Attributes:	ShippingName, shippingAddress
Corresponding Interfaces:	<<include>> in Check Out, Check Out
EBTs:	Selling
Attributes:	Seller
Operations:	deliver(), collectPayment()
Business Objects:	Delivery
Attributes:	customerInfo, deliveryType, deliveryDate
Operations:	processDelivery()
Industrial Objects:	Shipping Info
Attributes:	shippingName, shippingAddress
Operations:	createShipInfo(), destroyShipInfo(), getShipInfo()
Use Case Description:	<ol style="list-style-type: none"> 7. buyer enters customer info 8. buyer enters shipping address
Alternatives:	none

Use Case Id:	14
Use Case Title:	Enter Credit Card Info
Actors & Corresponding Roles:	Buyer, buyer
Corresponding Classes:	Credit Card
Corresponding Attributes:	CreditCardNumber, creditCardType, expirationDate
Corresponding Interfaces:	<<include>> in Check Out, Check Out

EBTs:	Buying
Attributes: Buyer Operations: Order(), Pay()	
Business Objects:	Payment
Attributes: paymentAmount, paymentType Operations: requestPayment(), processPayment()	
Industrial Objects:	CreditCard
Attributes: CreditCardNumber, creditCardType, expirationDate Operations: verifyCreditCard(), createCreditCard(), destroyCreditCard(), getCreditCard(), chargeCreditCard()	
Use Case Description:	<ol style="list-style-type: none"> 1. buyer enters credit card number 2. buyer enters credit card expiration date 3. buyer enters credit card type
Alternatives:	none

Use Case Id:	15
Use Case Title:	Verify Credit Card
Actors & Corresponding Roles:	Credit Verification Company
Corresponding Classes:	Credit Card
Corresponding Attributes:	CreditCardNumber, creditCardType, expirationDate
Corresponding Interfaces:	Verify
EBTs:	Buying
Attributes: Buyer Operations: Order(), Pay()	
Business Objects:	Payment
Attributes: paymentAmount, paymentType Operations: requestPayment(), processPayment()	
Industrial Objects:	Approval, Credit Card
Attributes: CreditCardNumber, creditCardType, expirationDate Operations: verifyCreditCard(), createCreditCard(), destroyCreditCard(), getCreditCard(), chargeCreditCard()	
Attributes: Status, limit, balance Operations: approve(), decline()	
Use Case Description:	<ol style="list-style-type: none"> 1. buyer checks status 2. buyer checks balance 3. buyer checks limit 4. buyer checks charge 5. buyer approves charge
Alternatives:	5.1 buyer declines charge

CRC Cards

Shopping (EBT) (Trader)		
RESPONSIBILITY	COLLABORATION	
Product transition	CLIENT	SERVER
	<ul style="list-style-type: none"> • Buying (EBT) • Selling (EBT) • Product (BO) 	<ul style="list-style-type: none"> • ExamineProduct() • CompareProduct() • SelectProduct()

Buying (EBT) (Buyer)		
RESPONSIBILITY	COLLABORATION	
Product purchasing	CLIENT	SERVER
	<ul style="list-style-type: none"> • Shopping (EBT) • Payment (BO) • Order (BO) 	<ul style="list-style-type: none"> • Order() • Pay()

Selling (EBT) (Seller)		
RESPONSIBILITY	COLLABORATION	
Product selling	CLIENT	SERVER
	<ul style="list-style-type: none"> • Shopping (EBT) • Delivery (BO) 	<ul style="list-style-type: none"> • Deliver() • CollectPayment()

Product (BO) (Product)		
RESPONSIBILITY	COLLABORATION	
Entity of trading	CLIENT	SERVER
	<ul style="list-style-type: none"> • Shopping (EBT) • Inventory (IO) 	<ul style="list-style-type: none"> • DisplayProduct() • ProductSearch() • UpdateProduct() • Create Product() • DestroyProduct() • SearchProduct ()

Invoice (BO) (Invoice)		
RESPONSIBILITY	COLLABORATION	
	CLIENT	SERVER
Record the purchase and payment information	<ul style="list-style-type: none"> • Buying (EBT) • Product (BO) • Order (BO) • Payment (IO) 	<ul style="list-style-type: none"> • SendInvoice() • ChangeInvoice()

Order (BO) (Order)		
RESPONSIBILITY	COLLABORATION	
	CLIENT	SERVER
Order product	<ul style="list-style-type: none"> • Buying (EBT) • Shopping Cart (IO) • Order Form (IO) 	<ul style="list-style-type: none"> • RequestOrder() • ProcessOrder()

Delivery (IO) (Delivery)		
RESPONSIBILITY	COLLABORATION	
	CLIENT	SERVER
Deliver products purchased	<ul style="list-style-type: none"> • Selling (EBT) • Shipping Info (IO) 	<ul style="list-style-type: none"> • ProcessingDelivery()

Inventory (IO) (Product store)		
RESPONSIBILITY	COLLABORATION	
	CLIENT	SERVER
Provide available products	<ul style="list-style-type: none"> • Product (BO) 	<ul style="list-style-type: none"> • UpdateInventory() • CreateInventory() • DestroyInventory() • CheckInventory() • GetInventory()

Credit Card (IO) (Payment type)		
RESPONSIBILITY	COLLABORATION	
Provide a way to pay	CLIENT	SERVER
	<ul style="list-style-type: none"> • Payment (BO) • Approval Credit Card (IO) 	<ul style="list-style-type: none"> • VerifyCreditCard() • CreateCreditCard() • DestroyCreditCard() • GetCreditCard() • ChargeCreditCard()

Approval Credit Card (IO) (Credit Card approval)		
RESPONSIBILITY	COLLABORATION	
Provide Credit Card verification	CLIENT	SERVER
	<ul style="list-style-type: none"> • Credit Card (IO) 	<ul style="list-style-type: none"> • Approve()

Payment (IO) (Payment)		
RESPONSIBILITY	COLLABORATION	
Pay for the product	CLIENT	SERVER
	<ul style="list-style-type: none"> • Buying (EBT) • Credit Card (IO) • Check (IO) 	<ul style="list-style-type: none"> • RequestPayment() • ProcessPayment()

Check (IO) (Payment type)		
RESPONSIBILITY	COLLABORATION	
Provide way to pay	CLIENT	SERVER
	<ul style="list-style-type: none"> • Payment (BO) 	<ul style="list-style-type: none"> • SignCheck() • ClearCheck() • DepositCheck()

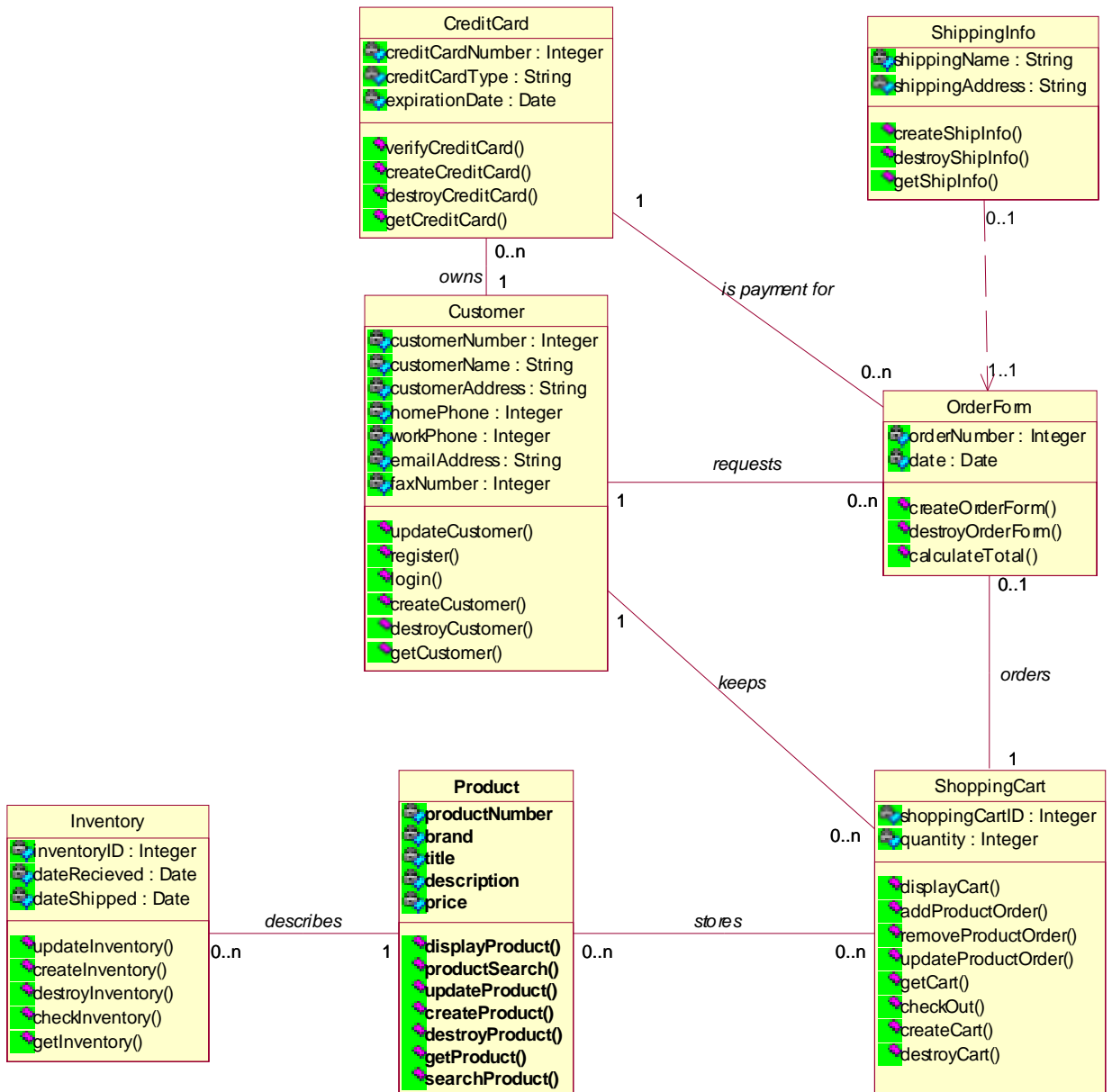
Shopping Cart (IO) (Product holder)		
RESPONSIBILITY	COLLABORATION	
Holding Products	CLIENT	SERVER
	<ul style="list-style-type: none"> • Order (BO) 	<ul style="list-style-type: none"> • DisplayCart() • AddProductOrder() • RemoveProductOrder()

		<ul style="list-style-type: none"> • UpdateProductOrder() • GetCart () • Checkout() • CreateCart() • DestroyCart ()
--	--	--

Order Form (IO) (Order recorder)		
RESPONSIBILITY	COLLABORATION	
Record order information	CLIENT	SERVER
	<ul style="list-style-type: none"> • Order (BO) 	<ul style="list-style-type: none"> • CreateOrderForm() • DestroyOrderForm() • CalculateTotal() • UpdateOrderForm() • VerifyOrderForm()

Shipping Info (IO) (Shipping record)		
RESPONSIBILITY	COLLABORATION	
Record shipping information	CLIENT	SERVER
	<ul style="list-style-type: none"> • Delivery (BO) 	<ul style="list-style-type: none"> • CreateShipInfo() • DestroyShipInfo() • GetShipInfo()

Class Diagram (Traditional Model)



Description:

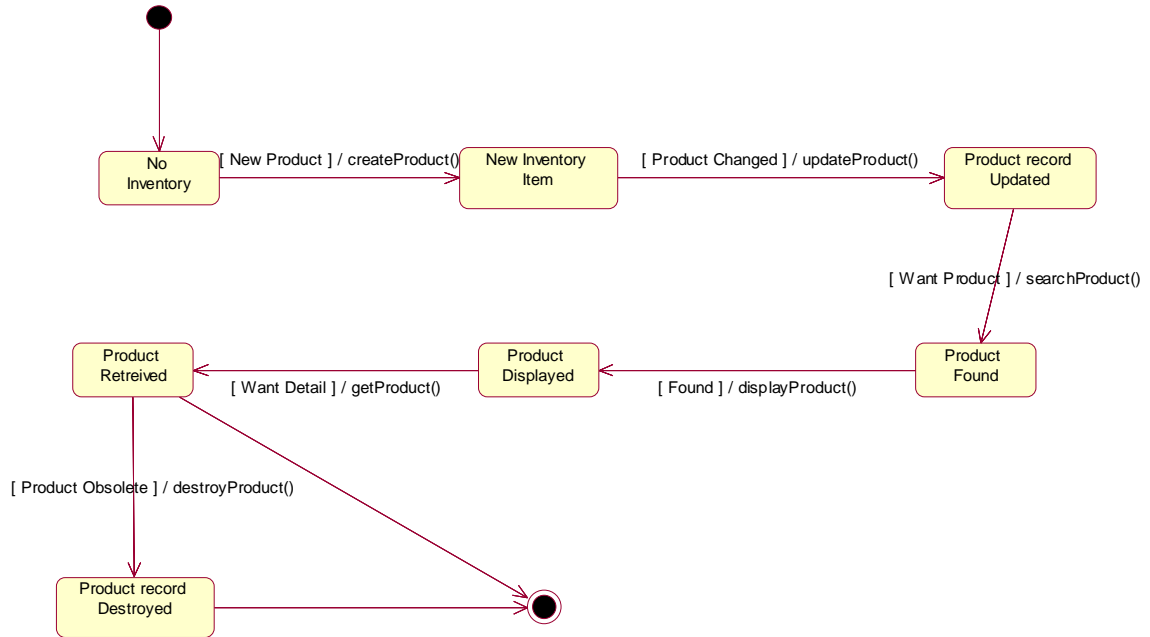
The traditional model is made up of mostly industrial objects as is typical of a model developed without stability in mind. It shows the relationships among the various traditional classes. These classes are self explanatory. Naming of the relationships among the classes makes the traditional model more clear and understandable.

Description:

In the stability model we have made all traditional model classes industrial objects with the singular exception of “Product”. We feel that Product is properly a business object because anything can be a product. In the specific problem domain a product is something sold by the electronics store. However with a stable and reusable core, product could become any thing that is bought and sold (e.g. a car, a house, a dog, etc.). The enduring business themes are all quite stable over time. Shopping, buying and selling apply to any purchase relationship or item. The business objects are stable externally as they relate to the enduring business themes, and change only with their internal implementation. The industrial objects are a specific implementation of the model for our specific problem domain.

Behavior Model – State Transition Diagram (STD)

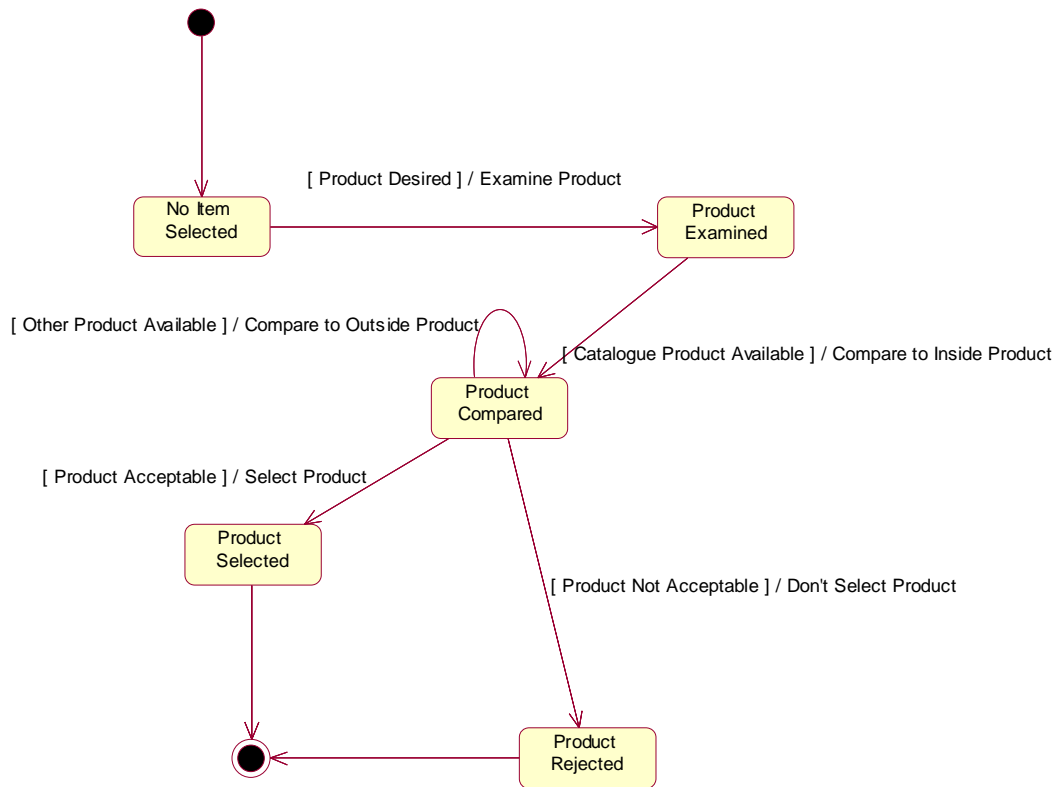
Product



Description:

Product transitions through several states in the system as it is entered in inventory, updated, and destroyed. Additionally the product class can be displayed or retrieved from the catalogue which is implicit in our system.

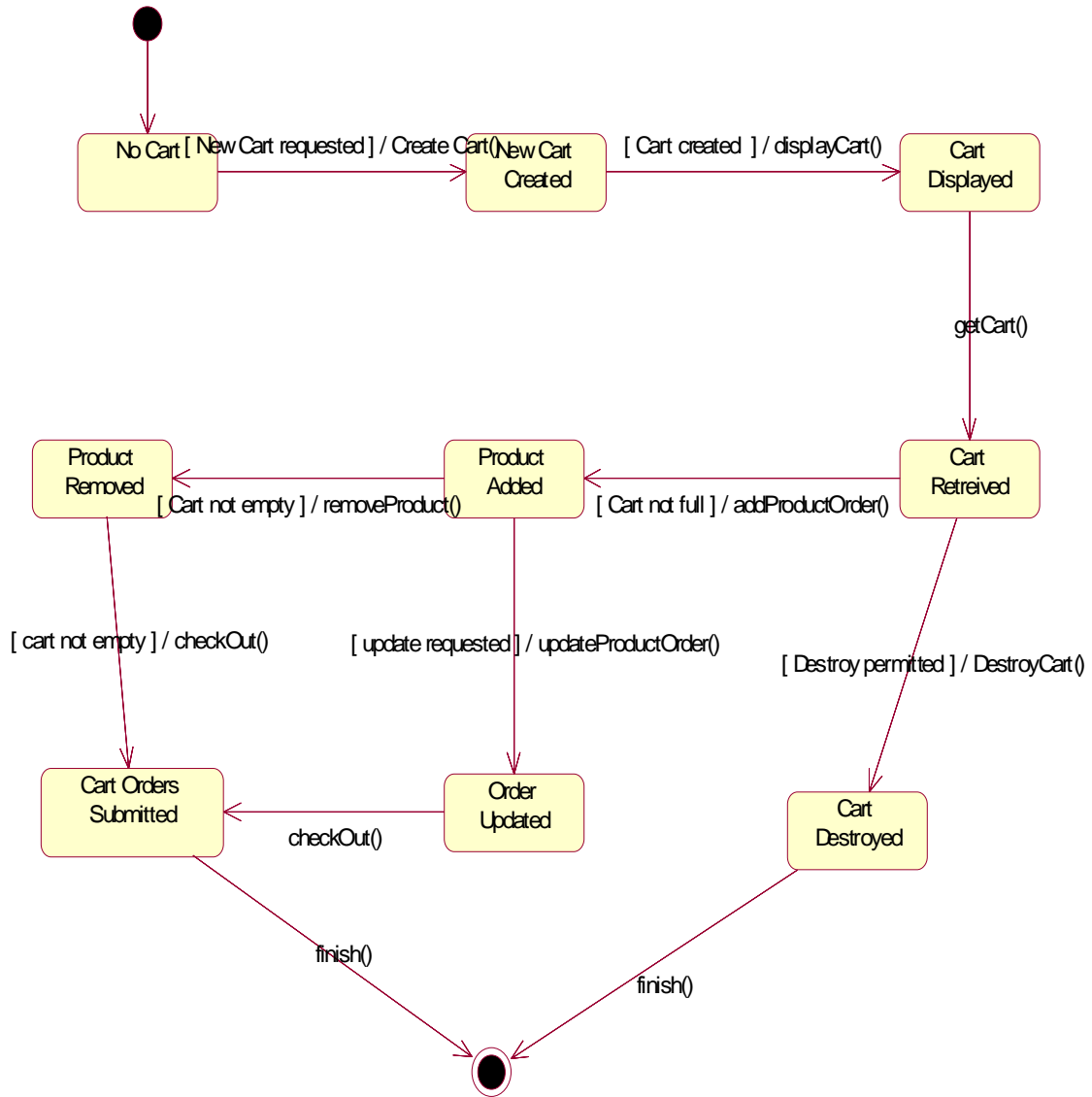
Shopping



Description:

Shopping is one of our enduring business themes it allows browsing of products, comparison of products with others in our system and selection or rejection of any product.

Shopping Cart

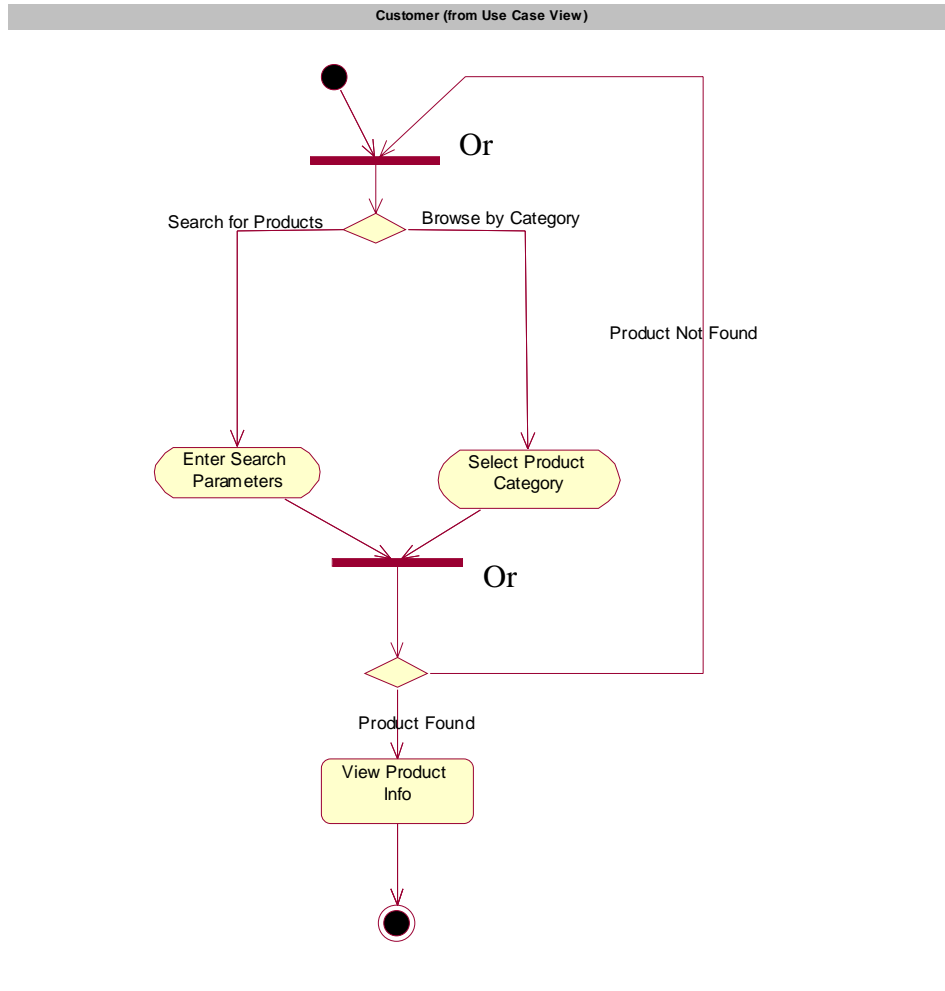


Description:

Shopping cart is one of our industrial objects. It allows creation and destruction of instantiations. Other states include adding and removing products, updating orders, display or retrieval of the cart, and submission of the entire cart of orders.

Behavior Model – Activity Diagram

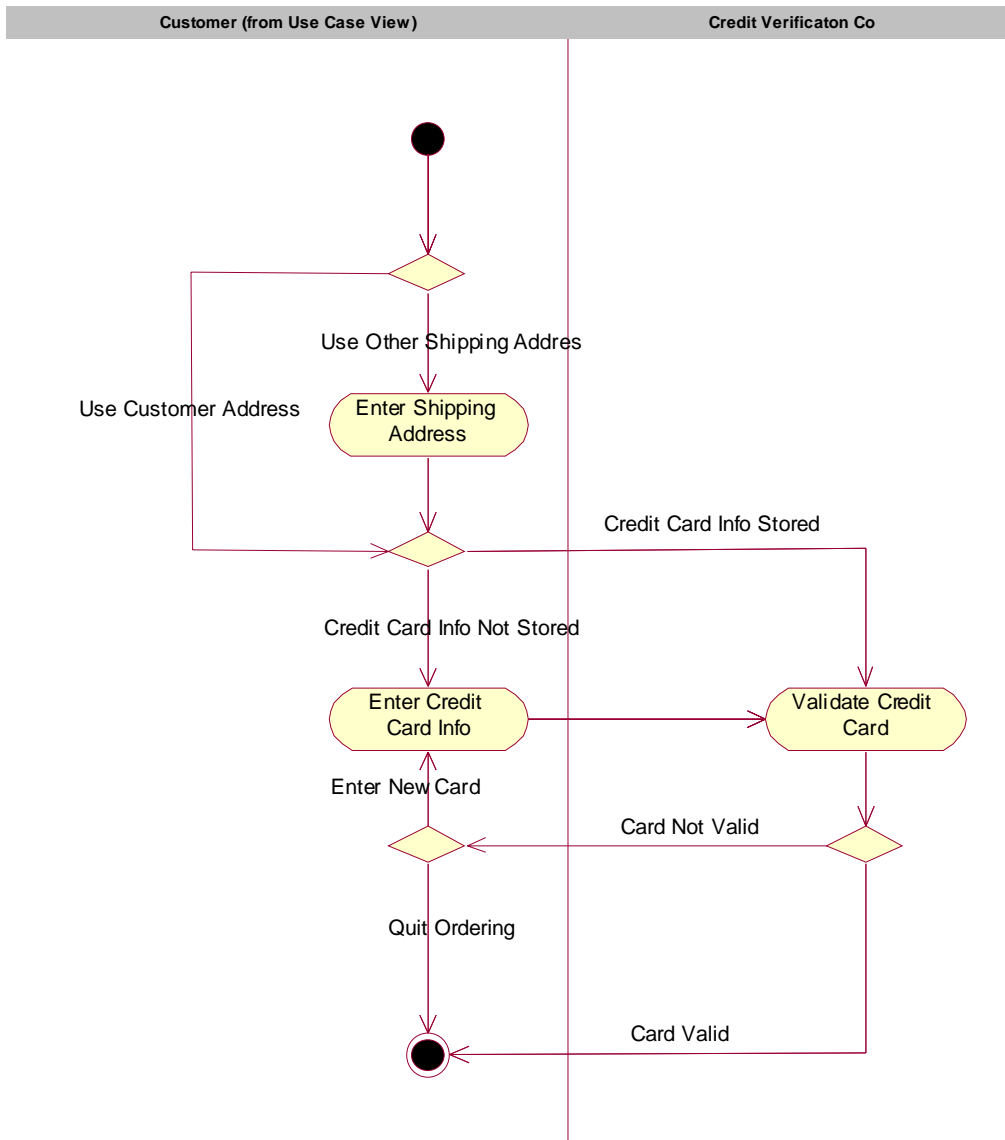
Activity diagram – buyer (customer) browsing activities



Description:

Customer browsing allows a search for a specific product or an examination of an entire product category. If a product is found it can be viewed, if not found the customer is allowed to search or brows again.

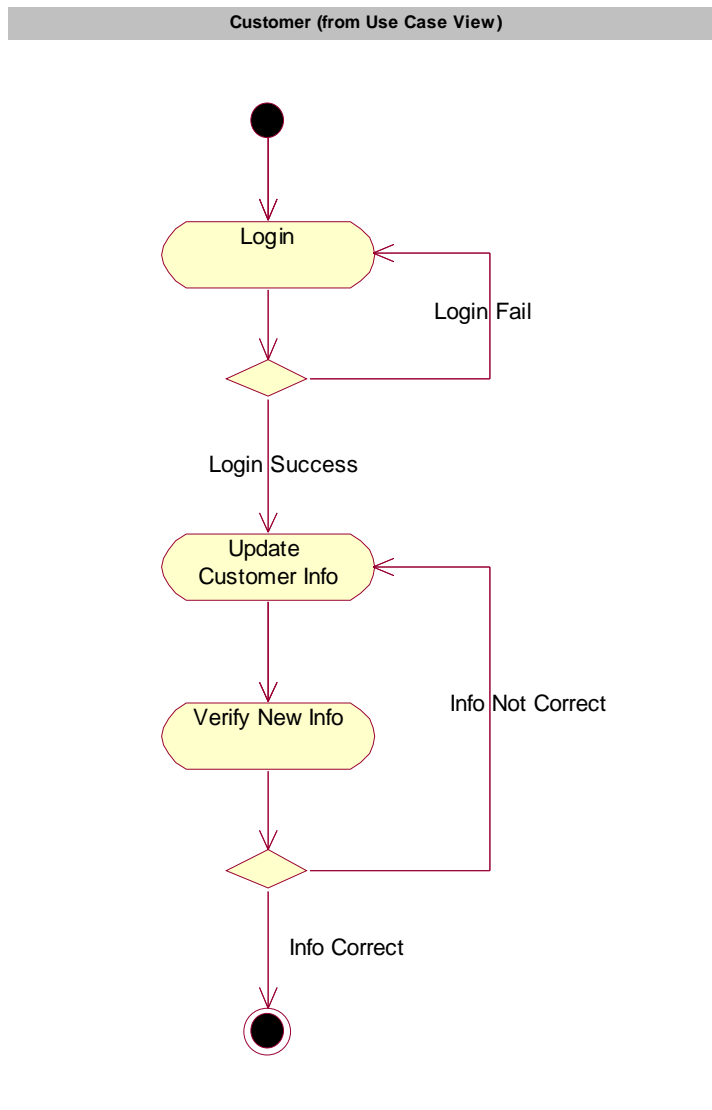
Activity diagram – buyer (customer) checkout activities



Description:

Customer checkout first allows identification and entry of the shipping address, then entry or retrieval of credit card information, and finally validation of the credit card.

Activity diagram – Buyer (customer) login and update activities



Description:

Customer login and update allows an update and verification of customer information after a valid login process.

OO Heuristics

The design heuristics provide a guide to what is and what is not appropriate in the system design. They are “rules of thumb” to which we can compare our design elements.

Our design includes about the same number of methods for each class, intelligence is distributed (heuristic 3.1). There is no macho class (heuristic 3.2), and each class performs needed operations.

Heuristic 2.8 A class should capture one and only one key abstraction.

Our classes are simple with one responsibility each. Each plays a role as a main entity within the domain model. No key abstraction maps to more than one class.

Heuristic 2.9 Keep related behavior and data in one place.

All related data is kept within one class. Policy is a good example the data needed for the contract and all of the needed data is contained within policy.

Heuristic 2.11 Be sure the abstractions that you model are classes and not simply the roles objects play.

The use of the four tests helped to ensure that each class qualifies as a class rather than an instance of a class or a specific role.

Heuristic 3.1 Distribute system intelligence horizontally as uniformly as possible, that is, the top level classes in a design should do the work uniformly.

Heuristic 3.4 Beware of classes that have too much noncommunicating behavior, that is, methods that operate on a proper subset of the data members of a class. God classes often exhibit much noncommunicating behavior.

Our classes have no behaviors that are noncommunicating. Each operation must necessarily communicate with at least one other class.

Heuristic 4.1 Minimize the number of classes with which another class collaborates.

We have kept our design simple and understandable with only four high level classes. We feel that communicating with three other classes meets this heuristic. Note: there is the potential of conflict with heuristic 3.1.

Heuristic 4.2 Minimize the number of message sends between a class and its collaborator. Only necessary messages are sent or received.

Heuristic 5.5 In practice, inheritance hierarchies should be no deeper than an average person can keep in short term memory. The text goes on to suggest no more than 6 deep. We feel an arbitrary numbers is not needed and too specific. In practice our inheritance hierarchies are no more than three levels deep which enhances understandability and clarity.

Heuristic 6.1 No multiple inheritance. We have carefully avoided multiple inheritance in our stability model.

This is just a sampling of the heuristics we applied. We could go on for a long time listing each heuristic separately.

Rewrite of Problem Statement

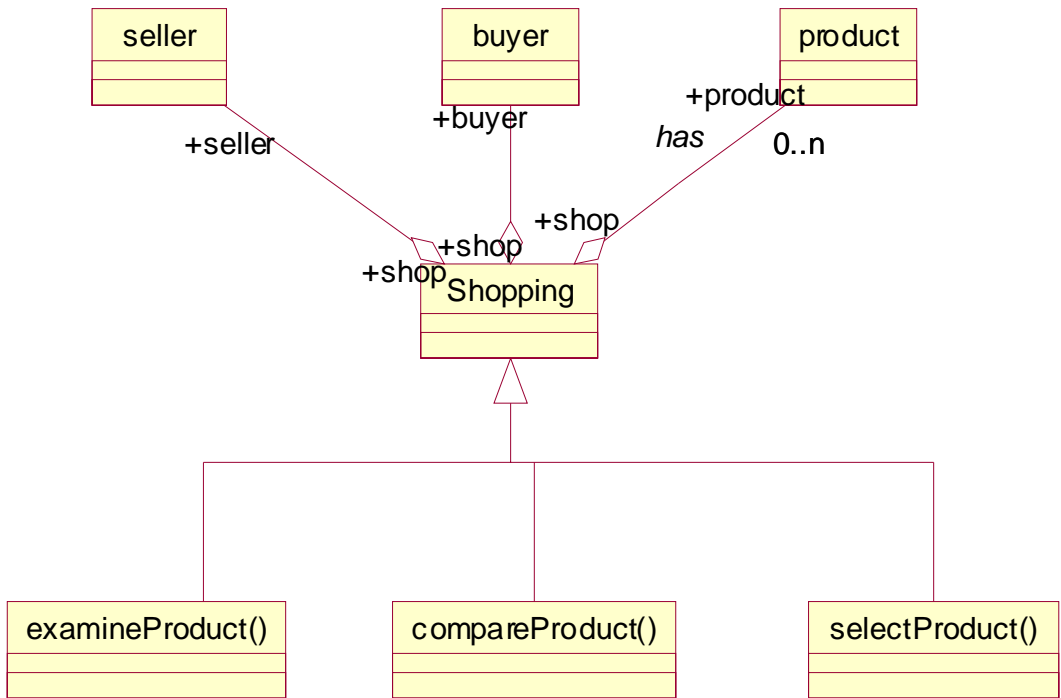
Our original problem statement dealt with the specific problem domain in a traditional manner. After analyzing our model with stability in mind we see that a stable model has little to do directly with Doe's Electronics e-commerce. Rather the problem becomes the stable, abstract, and intuitive "shopping". Shopping does not necessarily require a purchase, but does require both buying and selling which are our other two enduring business themes. Shopping involves some type of product, any type of product, which is one of our business objects. If a purchase is made then an order, invoice and delivery are required which are our other three business objects.

Only when we get to the actual implementation of the system do we then deal with the original Doe's Electronics traditional problem domain. There we implement our design (business objects) with industrial objects from our traditional model. The implementation is specific to the immediate problem domain.

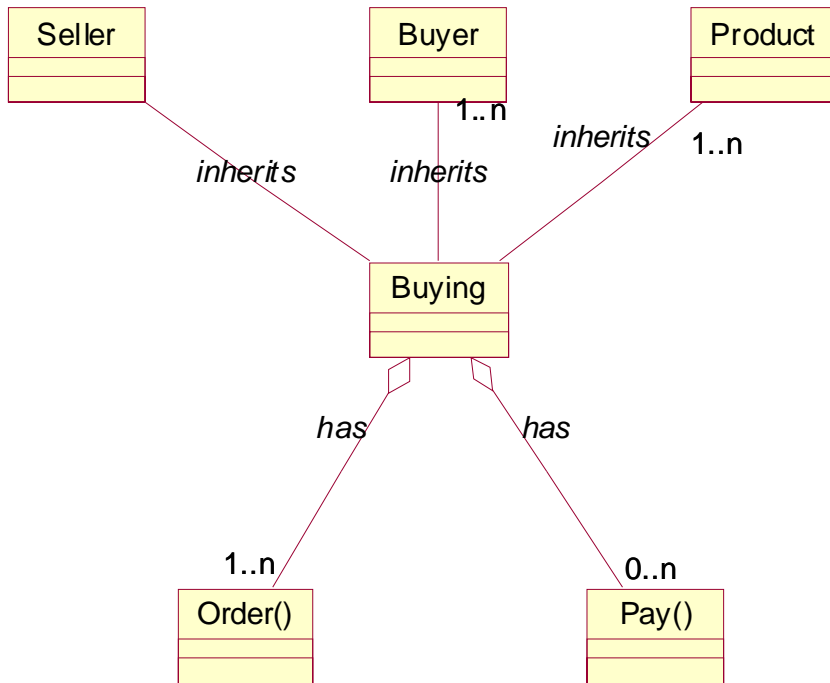
The stable model is enduring, adaptable, reusable, and scalable. Doe's Electronics can begin selling products other than electronics, or could change its business entirely from retail electronics to wholesale commodities, or real estate, or any number of other selling activities.

Pattern Documentation

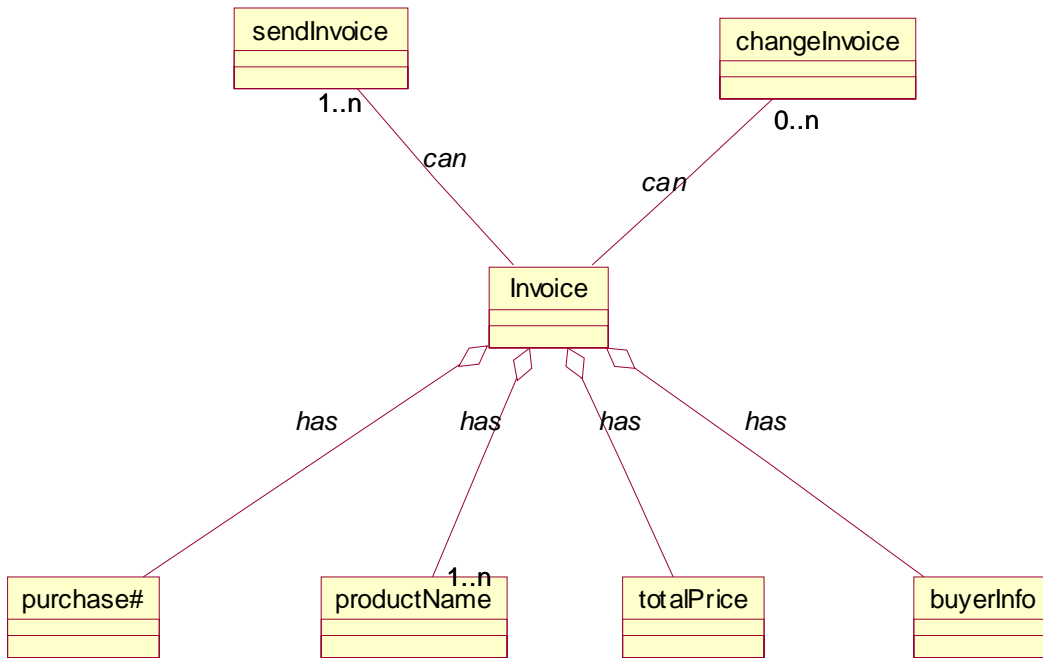
Meta Model Shopping (EBT)



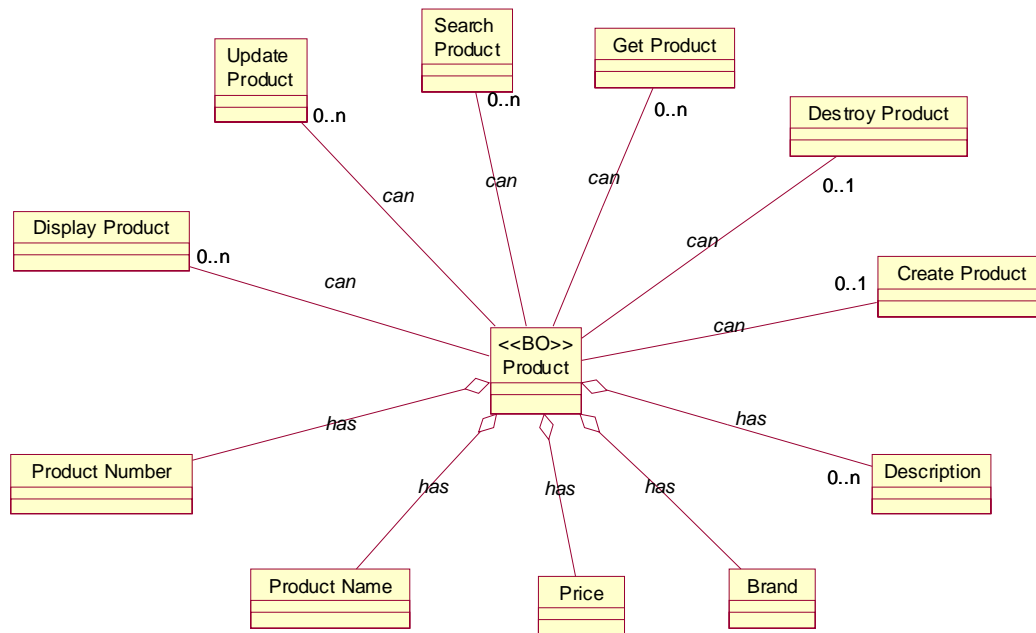
Meta Model Buying (EBT)



Meta Model Invoice (BO)



Meta Model Product (BO)



Reference

Riel, Arthur J., Object-Oriented Design Heuristics, Addison Wesley Longman, Inc. 1996.

Oestereich, Bernd, Developing Software with UML, Addison Wesley Longman Ltd, 2001.