Process Architecture
Disusun oleh :
Malabay

Sumebr dari : https://www.utdallas.edu/~chung/SA/Project2-sun-zhao-lin.doc

1. Introduction

In the new era of Internet and information highway, electronic commerce is one of the fastest growing and evolving area. It typically involves use of an electronic commerce system, which enables buyers and sellers to exchange commodities and services through electronic processes. Electronic business operations have been the privilege of large, sizable companies that had the knowledge, technology and sufficient capital to invest in electronic infrastructures that support business transactions

electronically. To support such an advanced concept, an intelligent architecture should be adopted. A B2B-OPS (business to business order processing system) is being considered as part of the next generation of electronic commerce system.

B2B-OPS shall invite existing business customers to log into the system. Customers can browse categories or search key words through the system, and then select any products and services collected by the B2B-OPS through system suppliers. Any selections can be stored in customer's shopping cart for further review, modification and cancellation by this customer, or negotiation with suppliers. The selected item(s) in the shopping cart can make up one to multiple final concrete orders. These orders will be sent to accounting department for generating customer credit check and invoice (the invoice process system (B2B-IPS) discussed in Project I). The shipping department shall be notified at the end to generate shipping slips and attach them to the ordered products. The shipping method was pre-selected by the customer in the final order.

In order to build this B2B-OPS system as part of an electronic commerce system, our team will consider five alternative software architectural designs for this project: *one for object-oriented structure (abstract data type), two for object-oriented designs (Unified Modeling Language),* one for implicit invocation structure, and one for modified pipe-and-filters structure. The best architecture of B2B-IPS from previous project is selected to build 3 of 5 current B2B-OPS architectures. Based on a detailed analysis of the advantages and disadvantages of our new designs, a rational decision is made to select one of the five architectures for this project.

2. Process Architecture

When system requirement document is available, software architecture design can be initiated. This architecture design is a process of defining architectures for specific software or multiple components to meet real world goals. It can be divided into four sub-processes:

- Requirement analysis (functional and non-functional requirements (FRs and NFRs)
- Architecture alternative design based on FRs and NFRs followed by design rationale
- Trade-off analysis based on NFRs and design rationale
- Design selection and final architecture evaluation

These sub-processes do not take place in a strict sequential pattern as described above but rather happen concurrently with many feedback loops as the multiple stakeholders (customer, architects, requirements engineers, etc.) negotiating, striving for some consensus. There will be multiple parties involved in the development, including the requirement engineers, product managers, and project managers, quality assurance managers and architects. The detailed process architecture is described as follows

2.1 Team roles

- **Requirement Engineer:** Requirement Engineer is responsible for identifying, analyzing, and specifying requirements for the software system. When the design phase starts, the requirement engineers should provide requirement specifications to the design team. During the architectural design, the architects go back to the requirement engineers for clarification if there is any ambiguity about the requirements. The requirement engineers, in turn may have to discuss the issues with the customers/users before they get back to the architects.

- **Product Manager:** Product Manager is responsible for the production and release of the software once the development is done.

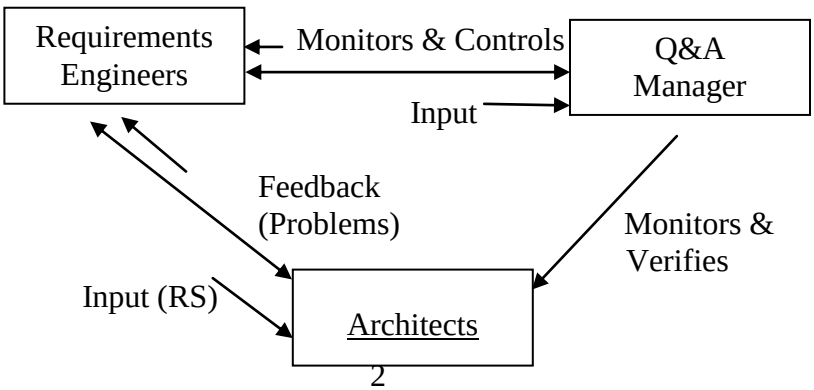
- **Project Manager:** Project Manager is Responsible for the management of requirements specification and architectural design and all the technical, financial, and personal aspects of this activity.

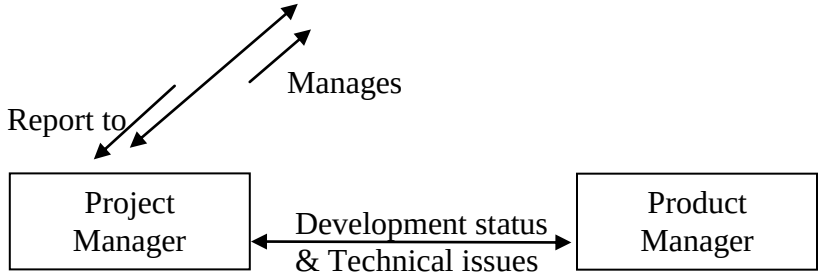
- **Quality Assurance Manager:** Quality Assurance Manager is responsible for verifying the architectural design against the requirement specification. He/she will ensure that the products and services are delivered at the required quality level, and that the project scope, cost, and time functions are fully integrated. His/her responsibility includes incorporation of quality assurance, a continuous process in every phase of the software life cycle.

- **Architect:** Architect is responsible for delivering a conceptual architecture design of the system based on both the functional and non-functional requirements. This architectural design will be the starting point for detailed design activities later in the development life cycle.

2.2 Process Architecture

Based on the responsibilities of the various parties involved and interactions between each party, the diagram below shows an overall team architecture in the architectural design process.





Team Architecture

2.3 Role Playing

Each of the three team members: Jihong Zhao, Steve Lin, and Wenping Sun, is assigned different roles during the architectural design activity based on the individual's strengths and past experience. We believe this way everyone can contribute the most to the project, and we will have a final product of the highest possible quality. The roles are assigned as following:

Wenping Sun played the roles of **customer**, **requirements engineer**, and **architect** in the project team. Her responsibility is:

(1) Analyze the project requirements,
(2) Design two Object-Oriented architectural alternatives.

She has very good understanding of real world goals, customer's needs and deep analysis of system requirements. She is excellent in object-oriented designs, especially in UML (Unified Modeling Language).

Jihong Zhao played the roles of **architect**, **product manager** and **project manager**. Her responsibility is:

(1) Design modified pipe and filter architectural alternatives, and OOD-UML.
(2) Resolve any issues that occurred during the architectural design process.
(3) Organize team meeting, monitor the status of the project, and sign contracts.

She has multi-talents in software architectural design, product and project management. The project benefits from her profound knowledge in the pros and cons of different architectures.

Steve Lin played the role of **architect**, **Q&A manager**. His responsibility is:

(1) Design implicit architectural alternatives.
(2) To check and make sure that the design meets the stated requirements, both

functionally, and non-functionally, and to make sure that the final document gets delivered on time and with quality.

He is a good manager in quality and assurance. He has thorough understanding in implicit invocation architecture.

2.4 Software Architecture Requirements

Functional Requirements

- Customer log-in
- Browse and search product/service
- Select and product/service into shopping cart
- Generate a final concrete order
- Performing accounting check
- Process B2B-IPS
- Generate shipping slip
- Ship products

Non-functional Requirements

- User-friendly
- Responsive
- Adaptable
- Reliable
- Fault-tolerant
- Accurate
- Customizable
- Affordable
- Portable

2.5 Meeting Schedule:

As the quality (non-functional requirements) of large systems can be highly constrained by a system's software architecture, it is in our best interest to determine, at the time a system's software architecture is specified, whether the software system will have the desired qualities.

We have our major meeting as following to discuss major issues for software architecture design. Between meetings, we have frequent information exchange via telephone calls, e-mails, etc. to make every effort for the success in this project.

First meeting: 10/5/00 Library

- Understand project description.
- Exchange knowledge about the application domain
- Work load distribution.
- Analyze modules in the system

Second meeting: 10/10/00 Library

- Discuss and distribute each member's work load
- Give comments and suggestions to teammates' works.

Third meeting: 10/17/00 Library

- Team members present their works and architecture designs.
- Discuss advantages and disadvantages of each design.
- Discuss the criteria of choosing the best architecture for this specific system.
- Come up with the agreement on the best architecture.
- Conduct decision point analysis.

Fourth meeting: 10/19/00 Library

- Present revised work
- Integrate our project
- Review project with comments and suggestions.
- Finish project

2.6 Selected B2B-IPS Architecture — OO-Unified Modeling Language (OO-UML)

In selected OO-UML architecture design (Figure 1) from Project I, we use class diagram model the static design view of Invoice processing system. Classes encapsulate data and information. They can be inherited and reused.

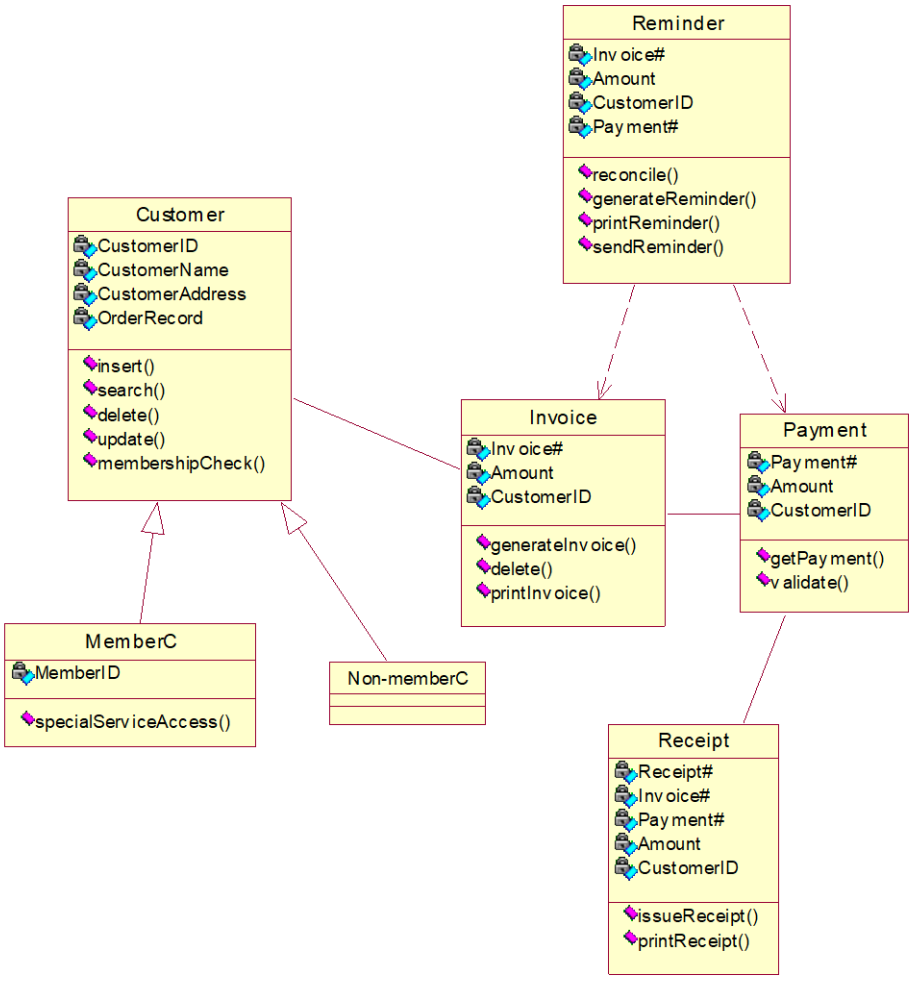


Figure 1. Object-Oriented - UML Architecture for B2B - IPS

Trade-off Analysis

Advantages:

- It is easy to add new features to the system, because changes in data representation or processing algorithms do not affect other objects, so it has high maintainability. (***maintainability* (+)**)
- The bundling of a set of accessing routines with the data they manipulate allows designers to decompose problem into collections of interacting agents. Therefore, the maintainability is very high.
- It provides reliable functionality and increases reusability, as modules make fewer assumptions about the others with which they interact. (***reliability* (+) *reusability* (+)**)
- Highly portable for the other systems. (***portability* (+)**)

Disadvantages:

- It usually has longer run-time when compared with other architecture styles. (***run-time performance* (-)**)
- It takes more time to design than other traditional structural designs.
- The architecture tends to use more space than traditional structure design styles. (***space* (-)**)

3. Architecture Design Alternatives

In this section, five different design alternatives are presented: one for Object-Oriented Design (ADT), two for Object-Oriented Designs (OOD-UML), and one each for Implicit Invocation Design and Modified Pipe-and-Filter. The Implicit Invocation Design and two UMLs are based on the selected B2B-IPS UML architecture. The detail description of each design is presented as following.

3.1 Modified Pipe and Filter Architecture

The Modified Pipe-and-Filter has Pipe-and-Filter's feather: each element does a local transformation using the input and producing output. Each glue serves as a conduit for the data stream, transmitting output of one process to input of another. And it has important new feature--a two-layer architecture: the upper layer is a control layer. It allows user to interactively control the process of filters.

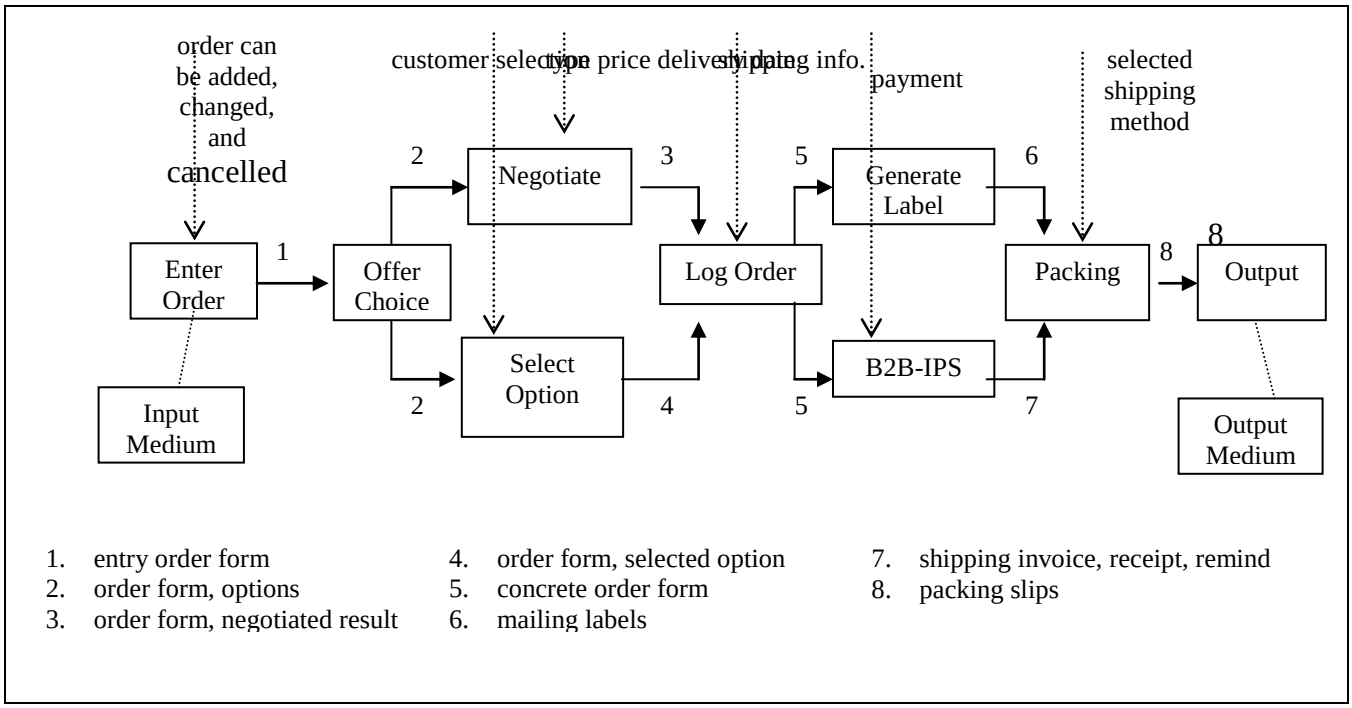


Figure 2. Modified Pipe and Filter Architecture

3.1.1 Architecture Style: Pipe and filters (Figure 2).

3.1.2 Components/elements: There are nine filters as the components in pipe-and-filter architectural style, as shown in the Figure 6.

- ***filter Enter Order***

operation **Read**:	reads data from the input medium. Data includes customers' order online, customer information.
operation **modifyOrder:**	change customer's order according to customer's requests and pre-set date.
operation **Output**:	outputs customer information entry order form.

- ***filter Offer Choice***

operation **Read**:	read entry order form.
operation **displayCatalogue**:	search product data base, find out products which fits customer's requirements.
operation **offerOption**:	for each selected item, display the purchase options.
operation **Output**:	output order form and options.

- ***filter Negotiate***

operation **Read**:	read order form and options
operation **toCustomer**:	get type, price and delivery date according to customer's requests and display supplier's response to customer.
operation **toSupplier:**	send customer's requests to supplier and get reply.
operation **getSelection:**	get final option according to negotiation.
operation **Output:**	output order form, negotiated result

- ***filter Select Option***

operation **Read**:	read order form and options
operation **getSelection:**	get final option according to customer's requests.
operation **Output:**	output order form, selected option

- ***filter Log Order***

operation **Read**:	read order form and selected options
operation **Log**:	log product according to the order
operation **gShippingAddr**:	generate shipping address
operation **gShippingDate**:	generate shipping date
operation **gShippingList**:	generate shipping list
operation **Output**:	output concrete order form (shipping information)

- ***filter Generate Label***

operation **Read**:	read shipping address
operation **gLabel:**	generate mailing label.
operation **Output:**	output mailing label

- ***filter B2B-IPS***

operation **Input**:	read shipping list
operation **gInvoice:**	generate shipping invoice.
operation **gPayment**:	get pre-payment from customer
operation **gReceipt:**	generate receipt for paid invoice.
operation **Reconcile**:	checks invoice and payment, makes payment match
operation **gReminder**:	generate reminder for unpaid invoice.
operation **Output:**	output shipping invoice, receipt, and reminder.

- ***filter Packing***

operation **Read**:	read mailing label and shipping invoice
operation **gPackage:**	generate packing slips.
operation **Output:**	output packing slips

- ***filter Output***

operation **Read**:	read packing slips
operation **shipOrder:**	ship order back to customer.
operation **Output:**	output all related data to output medium

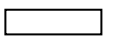
3.1.3 Interaction/connection:

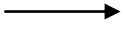
The interactions in the modified pipe-and-filter are pipes, control and system I/O.

3.1.4 Constrains:

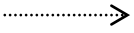
Each filter processes the input and produces output data. Each filter can run whenever it has the data needed to compute. Processes do not share states. They do not know the identity of it's upstream and down stream processes. Processes are independent from each other. And filters can be control by upper control layer.

3.1.5 Patterns


Filter


Pipe

System I/O


Control

3.2 Object-Oriented Design 1 — Unified Modeling Language (OOD-UML) (extension of B2B-IPS in OOD-UML)

In UML architecture design, we use class diagram model the static design view of Invoice processing system. Classes encapsulate data and information. They can be inherited and reused.

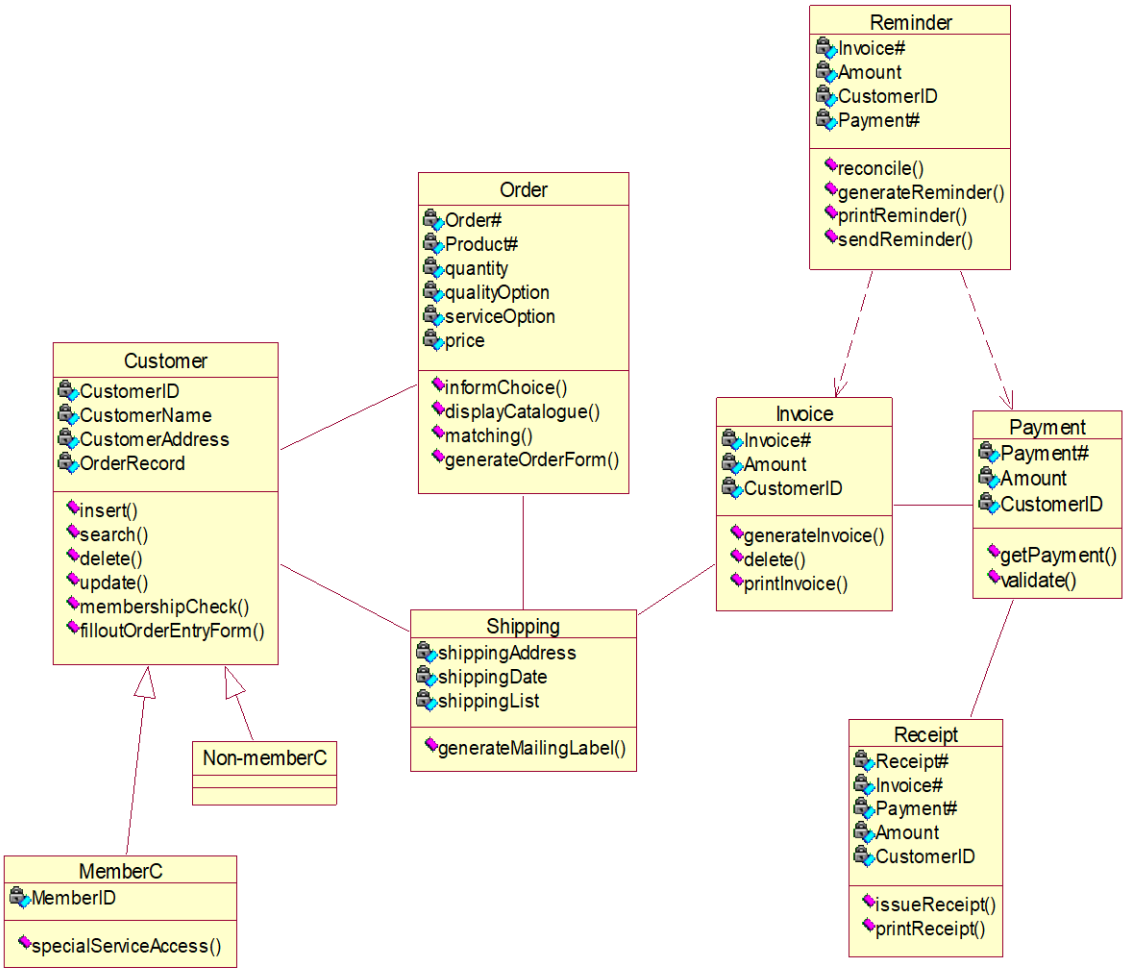


Figure 3. Object-Oriented (UML) Architecture

3.2.1 Architectural Style: Object-Oriented (UML) (Figure 3).

3.2.2 Components/Elements (class):

Customer: Read and store customer information, customer order information, and check customer's membership.

Order: Get customers' order information, match the needs of the customer against the goods and services the company offers, and generate concrete order form.

Shipping: Get order form and related customer information, generate mailing label, and shipping list.

Invoice: Get customer's order information and generate invoice.
Payment: Get payment information and validate payment.
Receipt: According to the payments, issue receipts.
Reminder: After reconcile the payments, generate remainders.

3.2.3 Interactions/Connections:

Class Customer

/* store customer information;
check customer membership, provide special feature service to member customer;
display order information, assign unique order# */

Class Order

/* get order entry form from Customer class;
inform the customer of the choices;
display catalogue;
match the needs of the customer against the goods and service the company offers;
generate concrete order form */

Class Shipping

/* log the order to generate the shipping address, shipping dates and shipping list;
based on the shipping address, generate mailing label */

Class Invoice

/* get customer information Customer class;
based on shipping list, issue invoice, and assign unique invoice# */

Class Payment

/* get invoice# and all relative information;
get payment information including:
the date the payment is made,
the amount paid,
the customer Id who made the payment,
and if it is by credit card, check or cash;
check payment is valid or not, if yes generate payment# */

Class Reminder

/* get invoice information and payment information;
compare invoice information with payment information;
create a reminder for the invoice that have passed due date */

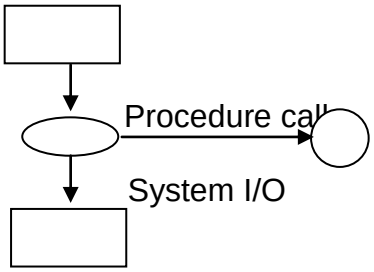
Class Receipt

/* issue receipt with unique receipt# according payment information:
invoice#: the invoice this receipt is related to,
amount: the amount received,
customerId: the customer the receipt is issued to */

3.2.4 Constraints:

Other components access data only by invoking the authorized functions.

3.2.5 Pattern:



3.3 Object-Oriented Design 2 -- Unified Modeling Language (UML) (extension of B2B-IPS in OOD-UML)

UML is the notations for classes, its data and member functions and the relationships among the classes are described in a standard format, i.e., through unified modeling language.

This architecture refined Object-Oriented Design 2 (UML), adding some interface and database classes.

3.3.1 Architectural Style: Object-Oriented (UML) (Figure 4, next page).

3.3.2 Components/Elements (class):

EntryOrder: Take customer order, create new order record and customer account. Update order according to customer's request.

OrderEntryForm: Get customer order information.

ConcreteOrder: Inform customer's choice by display part of the catalog and provide option lists. Issue negotiation, get customer selection and log order.

LoggedOrderForm: Get shipping information, setup pre-set-date and create logged order record.

Shipping: According the shipping address, generate mailing label; based on shipping lists, generate invoice.

NewOrder: Store new order information (incomplete).

LoggedOrder: Store logged order information (concrete).

Product: Store the catalog the company maintains and purchase options.

B2B-IPS classes: Customer, Invoice, Payment, Receipt, and Reminder

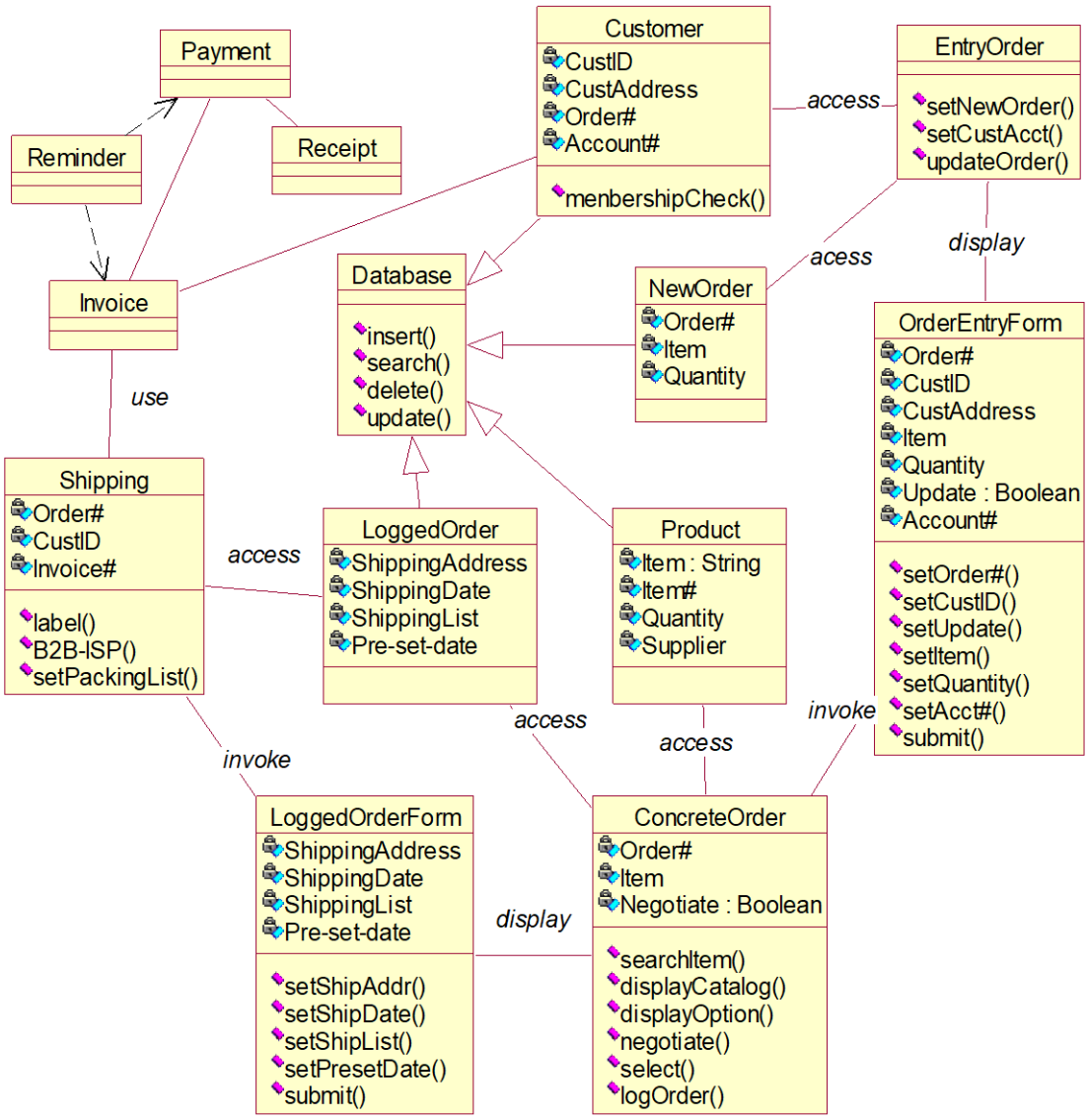


Figure 4. Object-Oriented Architectural Style (3)
Unified Modeling Language (UML)

3.3.3 Interactions/Connections:

Class Customer

/* Store customer information, include CustID, CustAddress, Order#, and Account#, etc;
Check customer membership and provide special feature service to member customer;
Display and update account information */

Class EntryOrder

/* Get customer information and order information from OrderEntryForm class;
Issue update, access to Customer class */

Class OrderEntryForm

/* Set customer information and order information including:
CustID, CustAddress, Order#, Account#, Item, Quantity
and Update (true or false);
Submit order, invoke ConcreteOrder class */

Class ConcreteOrder

/* Access Product database to get catalog and option, and update Product;
Issue negotiation and select option;
Get shipping information from LoggedOrderForm class;
Access LoggedOrder database to store the order */

Class LoggedOrderForm

/* Set shipping information including:
ShippingAddress, ShippingDate, ShippingList, and Pre-set-date:
Submit order, invoke Shipping class */

Class Shipping

/* Access LoggedOrder database, get ShippingAddress and generate mailing label;
Generate invoice by using B2B-IPS;
Set packing lists */

Class NewOrder

/* Store entry order information including:
Order#, Item, Quantity;
Perform basic database operations, used by EntryOrder class */

Class LoggedOrder

/* Store logged order information including:
ShippingAddress, ShippingDate, ShippingList, and Pre-set-date;
Perform basic database operations, used by ConcreteOrder class and Shipping class */

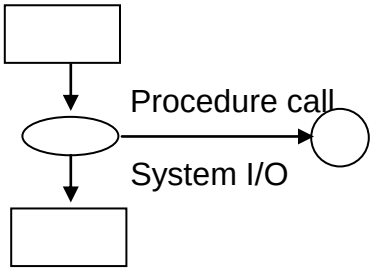
Class Product

/* Store the catalog the company maintains and purchase options;
Perform basic database operations, used by ConcreteOrder class */

3.3.4 Constraints:

Other components access data only by invoking the authorized functions. Classes can share same storage.

3.3.5 Pattern:



3.4 Object-Oriented Design 3 – Abstract Data Type (ADT)

In ADT architecture design, the components are objects. The system is divided into ADT objects, each handling a specific aspect of system functions. Each ADT object provides interface to communicate with other objects. An object is responsible for preserving the integrity of its representation that is hidden from other object. So data are not directly shared by different objects, but through explicitly invoking interfaces.

3.4.1 Architectural Style: Object-Oriented (ADT) (Figure 5, next page).

3.4.2 Components/Elements:

Input: Get information through input medium.
Customer: Store new customer information. Read and check existing customer information, order information, Check customer membership.
Order process: Get customer order information, matching the needs of the customer against the goods and services the company offers, generate concrete order form.
Shipping Handling: Get order form and related customer information, generate mailing label and shipping list.
Invoice process: According shipping list generate invoices.
Payment Handling: Get payment information and validate payment.
Receipt Handling: According payment issue receipt.
Reminder Process: After reconcile generate remainder
Output: Print invoice, receipt and remainder through output medium.

3.4.3 Interactions/Connections:

*Module **Input:***

Get customer information and order information from customer and store in Customer File.

*Module **Customer:***

Procedure **Set Cust Info (id, name, addr, creditPeriod):**

/* Create customer based on
id: customer id;
name: customer name;
addr: customer address;
creditPeriod: credit period; */

Procedure **MembershipCheck (id)**

/* check customer membership;
display special feature service option to member customer;
provide special feature service to member customer;
accept customer membership application */

Procedure **Orer Infor (item#)**

/* display order information, assign unique order# */

Function **Get Cust Info(id):**

/* return customer information including
customer id,
customer name,
customer address,
customer credit period */

*Module **Order process***

Procedure **setup (OrderInfo):**

/* return customer order information including
product id,
quantity */

Function **InformChoice ()**

/* return customer choice including
displayCatalogue
chooseServices
quality
delivery
warranty plan */

Function **Matching ()**

/* matching the needs of the customer against the goods and services the company offers */

Procedure **OrderForm**

/* Generate a concrete order form which is satisfied by the customer */

*Module **Shipping handling***

Procedure **setup (CustInfo, OrderForm)**

/* System get customer information from customer module and final order information from order processing module */

Function **mailingLabel ()**

/* According the information get from setup, generate mailing label including:
shipping #,
shipping address,
shipping date */

Function **shippingList ()**

/* shipping #,
product id,
quantity */

*Module **Invoice process***

Procedure **setup (CustInfo, OrderInfo):**

/* System get customer information from Customer module and shipping list from shipping handling module */

Procedure **IssueInvoice (order#):**

/* According the information issue invoices, assign unique invoice# */

*Module **Payment Handling***

Procedure **setup (invoice#, payment):**

/* System get invoice# and all relative information */

Function **Get Payment ():**

/* return payment information including:
the invoice related to the payment,
the date the payment is made,
the amount paid,
the customer Id who made the payment,
and if it is by credit card, check or cash. */

Function **Validate Payment():**

/* Check payment is valid or not, if yes generate payment# */

*Module **Reminder Process***

Procedure **setup (invoice#, payment#):**

/* Get invoice information and payment information */

Procedure **Reconcile (invoice#, payment#):**

/* perform once a week in a predefined time;
compare invoice information with payment information;

create a reminder for the invoice that have passed due date. */

Function **Generate Reminder ():**
/* return reminder information */

*Module **Receipt Handling***
Procedure **setup (invoice#, payment#):**
/* create receipt with information:
invoice#: the invoice this receipt is related to,
amount: the amount received,
customerId: the customer the receipt is issued to. */

Function **Issue Receipt():**
/* return receipt with receipt# */

*Module **Output***
Operation **PrintInvoice:** get invoice from Invoice Processing and print invoice.

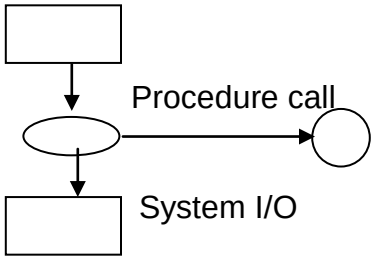
Operation **PrintReceipt:** get receipt from Receipt Handling and print receipt.

Operation **PrintReminder:** get Reminder from Reminder Processing and print reminder.

*Module **Master Control***
Synchronize the activities of all the other modules by using procedure calls of the modules.

3.4.4 Constraints: Each object provide interface that permit other components to access data only by invoking procedures in that interface

3.4.5 Pattern:



3.5 Implicit Invocation Design (extension of B2B-IPS in OOD-UML)

The idea behind implicit invocation is that instead of invoking a procedure directly, any components in the software system can announce one or more events. Other components in the system can register an interest in one or more particular event(s) by associating their procedures with it. When the event is announced, the system itself invokes all of the procedures that have been registered for this event. Thus an event announcement "implicitly" causes the invocation of procedures in other modules.

It has two important differences comparing to other architectural styles. First, the interface of the data is more abstract. Rather than exposing the storage formats to the computing modules, this solution accesses data abstractly. Second, computations are invoked implicitly as data is modified. Thus interaction is based on an "active data" model.

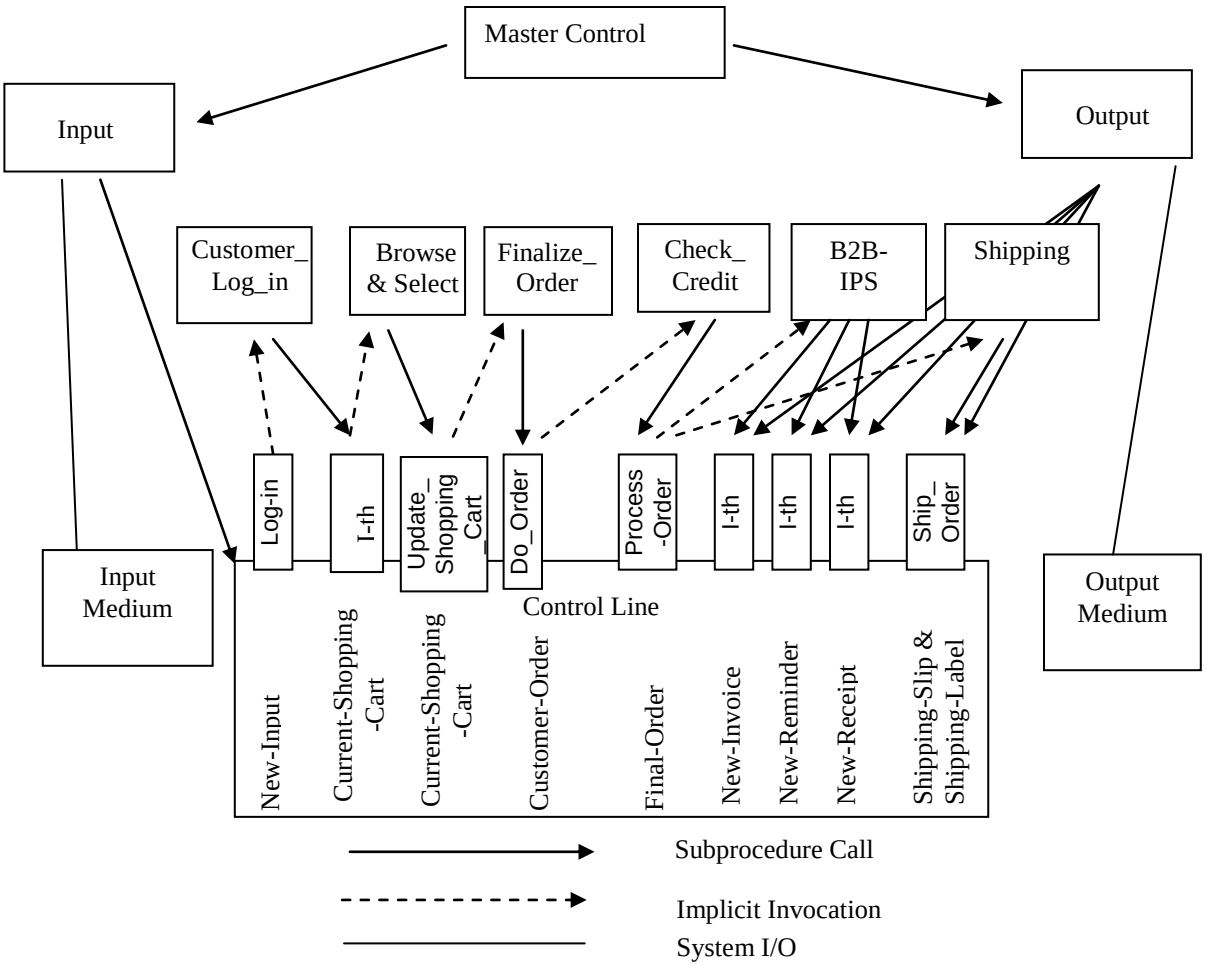


Figure 6. Implicit Invocation Architecture 1

3.5.1 Architectural Style: Hybrid implicit invocation architecture (Figure 6).

3.5.2 Components/Elements:

Input: Get information through input medium.

Customer_Log_in: Store new customer information. Read and check existing customer information, shopping cart content, order/payment information, and allow log-on into B2B-OPS.

Browse & Select: Search or browse system catalogues. B2B-OPS will display all keyword matches or complete list of same product or services at different prices or service qualities by suppliers. Customers will be allowed to make any selections and save into their shopping carts.

Finalize_Order: Get customers' selections from any items in their respective shopping carts and then generate Customer-Order.

Check_Credit: Get customer's order information. Check their account credit status for further purchases.

B2B-IPS: Get customer's order information and generate invoice, account reminder and payment as Project I. This architecture design of B2B-OPS is the extension of OO-UML design of B2B-IPS of Project I.

Shipping: Get customers' final order information, generate shipping slip, shipping label, and ship out products.

3.5.3 Interactions/Connections:

Module ***Input:***

Operation Read: customer information from input medium

/* through Operation Log-in, customer information like id, name, address are generated into New-Input */

/* → implicitly invoke Module Customer_Log_in */

Module ***Customer_Log_in***

/* through Operation I-th: read New-Input. Identify existing customers or create new customer and insert into Customer_File. Retrieve current content of logged-in customer's shopping cart into Current_Shopping_Cart. */

/* → implicitly invoke Module Browse & Select. */

Module ***Browse & Select***

/* through Operation Update_Shopping_Cart: read content Current_Shopping_Cart and perform modification based on customer's input. Browse B2B-OPS catalogues or display keyword search results. Insert customer selected products or services into Current_Shopping_Cart. */

/* → implicitly invoke Module Finalize_Order. */

Module ***Finalize_Order***

/* through Operation Do_Order: read content Current_Shopping_Cart. Select items in Current_Shopping_Cart to generate Curtomer-Order. Save Customer-Order and content of Current_Shopping_Cart into customer's account file for future reference. */

/* → implicitly invoke Module Check_Credit. */

Module ***Check_Credit***

/* through Process-Order: read Curtomer-Order. Perform customer credit check. Prove Customer-Order and generate Final-Order. */

/* → implicitly invoke Module B2B-IPS and Module Shipping. */

*Module **B2B-IPS***

/* through Operation (2^{nd}) I-th: read Final-Order and, then, generate New-Invoice as described in Project I. Other operation I-th will follow in B2B-IPS to produce New-Reminder, and New-Receipt. */

*Module **Shipping***

/* through Operation Ship-Order: retrieve information from Final-Order and generate Shipping-Slip and Shipping-Label. The content of Final-Order, product or service, will be deliver to customer using the method specified in the order. */

*Module **Output***

/* access Final-Order, New-Invoice, New-Receipt, New-Reminder, Shipping-Slip & Shipping-Label and print out */

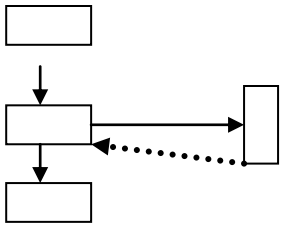
*Module **Master Control***

/* synchronize the activities of all the other modules by using procedure calls of this modules and explicitly triggers input and output. */

3.5.4 Constraints: Computations are invoked implicitly as data is modified, based on active data model.

3.5.5 Pattern:

Subprogram Call
System I/O



4. Tradeoff Analysis

When we decide which architecture design to use, we need to consider the following types of non-functional requirements for trade-off analysis.

- User Friendly --- how easy it is for user to use the system.
- Enhanceability --- accommodate new function to system.
- Performances --- space and time.
- Reusability --- to what extents can the components serve as reusable entities.
- Robustness --- behaves reasonably.
- Maintainability --- repairability and evolvability.
- Portability --- can be used in different platforms.

The advantages and disadvantages of the five architectural styles are discussed as follows.

4.1 Modified Pipe and Filter Architecture

4.1.1 Advantages:

- The architecture design is easy to understand overall input/output behavior of a system as a simple composition of the behaviors of the individual filters. (***understandability* (+)**)
- Systems are easy to maintain and enhance: new filters can be added to existing systems at the appropriate points and old filters can be replaced by improved ones. (***maintain* (+) *enhance* (+)**)
- It will support component/filter reuse: any two filters can be linked together. (***reusability* (+)**)
- Permit certain kinds of specialized analysis, such as throughput/bottleneck and deadlock analysis.
- Support concurrent execution: each filter can be implemented as a separate task and potentially executed in parallel with other filters. (***concurrent execution* (+)**)
- Easy to use: This modified pipe & filter solution makes the interactive control
- possible. Using this kind of system, the user can provide all sorts of parameters to the system directly and provide a collection of settings that determine what aspect can be modified dynamically by user. (***user friendly* (+)**)

4.1.2 Disadvantages:

- Often lead to a batch organization of processing. (***time performance* (-)**)
- Different filters may run at radically different speeds: it is unacceptable to slow some filter down because that another filter is still processing its data. (***time* (-)**)
- Hampered by having to maintain correspondences between two separate but related streams. (***maintainability* (-)**)
- Space is being used inefficiently, since almost all the data must be copied over. (***Space* (-)**)

4.2 Object-Oriented Design 2 — Unified Modeling Language

4.2.1 Advantages:

- It is easy to add new features to the system, because changes in data representation or processing algorithms do not affect other objects, so it has high maintainability. (***maintainability* (+)**)

- The bundling of a set of accessing routines with the data they manipulate allows designers to decompose problem into collections of interacting agents. Therefore, the maintainability is very high.
- It provides reliable functionality and increases reusability, as modules make fewer assumptions about the others with which they interact. (***reliability* (+) *reusability* (+)**)
- Highly portable for the other systems. (***portability* (+)**)

4.2.2 Disadvantages:

- It usually has longer run-time when compared with other architecture styles. (***run-time performance* (-)**)
- It takes more time to design than other traditional structural designs.
- The architecture tends to use more space than traditional structure design styles. (***space* (-)**)

4.3 Object-Oriented Design 3 — Unified Modeling Language

4.3.1 Advantages:

- GUI makes the system easy to use. (***user friendly* (+)**)
- It is easy to add new features to the system, because changes in data representation or processing algorithms do not affect other objects, so it has high maintainability. (***maintainability* (+)**)
- The bundling of a set of accessing routines with the data they manipulate allows designers to decompose problem into collections of interacting agents. Therefore, the maintainability is very high.
- It provides reliable functionality and increases reusability, as modules make fewer assumptions about the others with which they interact. (***reliability* (+) *reusability* (+)**)
- Highly portable for the other systems. (***portability* (+)**)
- Database usage allows efficient space utilization when compared with other architecture styles. (***space* (+)**)

4.3.2 Disadvantages:

- It usually has longer run-time when compared with other architecture styles. (***run-time performance* (-)**)
- It takes more time to design than other traditional structural designs. (***simplicity* (-)**)

4.4 Abstract Data Type

4.4.1 Advantages:

- The design style can easily accommodate new functionality without the

need for changing the existing modules. This is because that each module is designed to access other modules only through the well-defined interface procedures/functions. As an example, this design can be updated to be an inventory system by adding appropriate modules and appropriate interfaces. (***enhanceability* (+)**)

- Because each module in the design is self-encapsulated, and providing well defined functionality. Most components from this design style can easily accommodate with a new system, the reusability and portability are very high. (***reusability* (+)** and ***portability* (+)**)
- The system is easy for end-user to navigate. (***User Friendly* (+)**)
- If part of the system malfunction, it will not affect the whole system drastically as others would. So the robustness is strong. (***robustness* (+)**)
- This design is easier to maintain (+) compared to other design style. This is because all modules interact with each other only through clearly defined interface procedures/functions. Bug fixing inside one module will not result in change in other modules and interfaces. (***maintainability* (+)**)

4.4.2 Disadvantages:

- This design will consume more space than other design styles due to duplicated information (e.g. invoice Id, amount…). (***space* (-)**)
- Reconcile payments and invoices, reading and writing invoice/payment from/to database can be poor due to need for construction and destruction of the objects. (***performance* (-)**)
- One object must know the identity of the called object.

4.5 Implicit Invocation Architecture

4.5.1 Advantages

- This system supports functionality enhancement by addition of new components and interfaces. (***enhanceabiliby* (+)**)
- Insulate computations from changes in data representation. This will bring easy modification of individual modules without changing entire system. (***modification* (+)**)
- The system evolves easily: Components may be replaced or added by other newly designed components without affecting existing system or by establishing new interfaces for new components with current system. (***evolution* (+)**)
- It usually provides a user-friendly interface. (***user-friendliness* (+)**)
- If some components malfunction, the system might still perform some functions. (***robustness* (+)**)

4.5.2 Disadvantages

- Difficult to control the processing order of the implicitly invoked modules because components relinquish control over the computation performed by the system. (***time performance*** **(-)**)
- It usually requires more space due to transaction data/files creating. (***space performance*** **(-)**)
- Most of the system components have to be used in similar system due to data sharing. The reusability is low. (***reusability*** **(-)**)
- Reasoning about correctness can be problematic, since the meaning of a procedure that announces events will depend o the context of bindings in which it is invoked. (***understandability*** **(-)**)
- Low portability: some modules can be implemented in same architecture designs as a whole. (***portability*** **(-)**)

Table 1. Tradeoff analysis Summary

	Priority	OO Model 1 ADT	OO Model 2 UML	OO Model 3 UML	Implicit Invocation	Modified Pipe & Filter
Performance Time Space	 1 1	 +/- -	 +/- -	 + +/-	 - +	 - -
Enhanceability	1	+	+ +	+ +	+	+
Maintainability	2	+	+ +	+ +	+	+/-
User Friendliness	2	+	++	+ +	+	+
Robustness	3	+	+	+	+	-
Reusability	3	+ +	+ +	+ +	+/-	+
Portability	3	+	+	+	+/-	-
Total Credit		16	21	23	8	-1

Credit calculation rule: (+): 1; (-): -1; (+/-): 0 at Priority 1. (+): 2 at Priority 2 and (+): 3 at Priority 3.

5. The Selected Architectural Design

According to our tradeoff analysis for each design alternative (Table 1), Object-Oriented Design 3 — Unified Modeling Language is the best selection among architectural design list. The tool of object-oriented design is well developed and quite mature. The architecture design is clearly understood. The system is easily enhanceable and maintainable. The enhanceability and maintainability will be two important factors to be concerned during our architect design. Information hiding and encapsulation, the characteristics of the architecture, further increases system's security, robustness, portability and reusability. UML, a state of art requirement engineering tool, makes entire software process much more discipline oriented. The software system is even more enhanceable, modifiable, reusable.